A Fortran-Python Interface for Integrating Machine Learning Parameterization into Earth System Models

Tao Zhang[1], Cyril Morcrette[2,7], Meng Zhang[3], Wuyin Lin[1], Shaocheng Xie[3], Ye Liu[4], Kwinten Van Weverberg[5,6], Joana Rodrigues[2]

1. Brookhaven National Laboratory, Upton, NY, USA
2. Met Office, FitzRoy Road, Exeter, EX13PB, UK
3. Lawrence Livermore National Laboratory, Livermore, CA, USA
4. Pacific Northwest National Laboratory, Richland, WA, USA
5. Department of Geography, Ghent University, Belgium
6. Royal Meteorological Institute of Belgium, Brussels, Belgium
7. Department of Mathematics and Statistics, Exeter University, Exeter, UK

Correspondence to: Tao Zhang (taozhang.ccs@gmail.com)

Abstract

Parameterizations in Earth System Models (ESMs) are subject to biases and uncertainties arising from subjective empirical assumptions and incomplete understanding of the underlying physical processes. Recently, the growing representational capability of machine learning (ML) in solving complex problems has spawned immense interests in climate science applications. Specifically, ML-based parameterizations have been developed to represent convection, radiation and microphysics processes in ESMs by learning from observations or high-resolution simulations, which have the potential to improve the accuracies and alleviate the uncertainties. Previous works have developed some surrogate models for these processes using ML. These surrogate models need to be coupled with the dynamical core of ESMs to investigate the effectiveness and their performance in a coupled system. In this study, we present a novel Fortran-Python interface designed to seamlessly integrate ML parameterizations into ESMs. This interface showcases high versatility by supporting popular ML frameworks like PyTorch, TensorFlow, and Scikit-learn. We demonstrate the interface's modularity and reusability through two cases: a ML trigger function for convection parameterization and a ML wildfire model. We conduct a comprehensive evaluation of memory usage and computational overhead resulting from the integration of Python codes into the Fortran ESMs. By leveraging this flexible interface, ML parameterizations can be effectively developed, tested, and integrated into ESMs.

Plain Language

Earth System Models (ESMs) are crucial for understanding and predicting climate change. However, they struggle to accurately simulate the climate due to uncertainties associated with parameterizing sub-grid physics. Although higher-resolution models can reduce some uncertainties, they require significant computational resources. Machine learning (ML) algorithms offer a solution by learning the important relationships and features from high-resolution models. These ML algorithms can then be used to develop parameterizations for coarser-resolution models, reducing computational and memory costs. To incorporate ML parameterizations into ESMs, we develop a Fortran-Python interface that allows for calling Python functions within Fortran-based ESMs. Through two case studies, this interface demonstrates its feasibility, modularity and effectiveness.

1. Introduction

Earth System Models (ESMs) play a crucial role in understanding the mechanism of the climate system and projecting future changes. However, uncertainties arising from parameterizations of sub-grid processes pose challenges to the reliability of model simulations (Hourdin et al., 2017). Kilometer-scale high-resolution models (Schär et al., 2020) can potentially mitigate the uncertainties by directly resolving some key subgrid-scale processes that need to be parameterized in conventional low-resolution ESMs. Another promising method, superparameterization – a type of multi-model framework (MMF) (D. Randall et al., 2003; D. A. Randall, 2013), explicitly resolves sub-grid processes by embedding high-resolution cloud-resolved models within the grid of low-resolution models. Consequently, both high-resolution models and superparameterization approaches have shown promise in improving the representation of cloud formation and precipitation. However, their implementation is challenged by exceedingly high computational costs.

In recent years, machine learning (ML) techniques have emerged as a promising approach to improve parameterizations in ESMs. They are capable of learning complex patterns and relationships directly from observational data or high-resolution simulations, enabling the capture of nonlinearities and intricate interactions that may be challenging to represent with traditional parameterizations. For example, Zhang et al. (2021) proposed a ML trigger function for a deep convection parameterization by learning from field observations, demonstrating its superior accuracy compared to traditional CAPE-based trigger functions. Chen et al. (2023) developed a neural network-based cloud fraction parameterization, better predicting both spatial distribution and vertical structure of cloud fraction when compared to the traditional Xu-Randall scheme (Xu & Randall, 1996). Krasnopolsky et al. (2013) prototyped a system using a neural network to learn the convective temperature and moisture tendencies from cloud-resolving model (CRM) simulations. These tendencies refer to the rates of change of various atmospheric variables over one time step, diagnosed from particular parameterization schemes. These studies lay the groundwork for integrating ML-based parameterization into ESMs.

However, the aforementioned studies primarily focus on offline ML of parameterizations that do not directly interact with ESMs. Recently, there have been efforts to implement ML parameterizations that can be directly coupled with ESMs. Several studies have developed ML parameterizations in ESMs by hard coding custom neural network modules, such as O'Gorman & Dwyer (2018), Rasp et al. (2018), Han et al. (2020) and Gettelman et al. (2021). They incorporated a Fortran-based ML inference module to allow the loading of the pre-trained ML weights to reconstruct the ML algorithm in ESMs. The hard-coding has limitations Such hard-coding approach restricts the ML algorithm's ability to adapt to changes in the model dynamics over time, as the 'online' updating requires a two-way coupling between the dominantly Fortran-based ESMs and Python ML libraries.

Deleted: . Kochkov et al. (2023) presented an innovative ML parameterization that feeds back from the dynamics, in order to improve stability and reduce bias. However,

Deleted: s

Fortran-Keras Bridge (FKB; Ott et al. (2020)) and C Foreign Function Interface (CFFI; https://cffi.readthedocs.io) are two packages that support two-way coupling between Fortran-based ESM and Python based ML parameterizations. FKB enables tight integration of Keras deep learning models but is specifically bound to the Keras library, limiting its compatibility with other frameworks like PyTorch and Scikit-Learn. On the other hand, CFFI provides a more flexible solution that in principle supports coupling various ML packages due to its language-agnostic design. Brenowitz & Bretherton (2018) utilized it to enable the calling of Python ML algorithms within ESMs. However, the CFFI has several limitations. When utilizing CFFI to interface Fortran and Python, it uses global data structures to pass variables between the two languages. This approach results in additional memory overhead as variable values need to be copied between languages, instead of being passed by reference. Additionally, CFFI lacks automatic garbage collection for the unused memory within these data structures and copies. Consequently, the memory usage of the program gradually increases over its lifetime. In addition, when using CFFI to call Python functions from a Fortran program, the process involves several steps such as registering variables into a global data structure, calling the Python function, and retrieving the calculated result. These multiple steps can introduce computational overhead due to the additional operations required.

Additionally, Wang et al. (2022) developed a coupler to facilitate two-way communication between ML parameterizations and host ESMs. The coupler gathers state variables from the ESM using the Message Passing Interface (MPI) and transfers them to a Python-based ML module. It then receives the output from the Python code and returns them to the ESM. While this approach effectively bridges Fortran and Python, its use of file-based data passing to exchange information between modules carries some performance overhead relative to tighter coupling techniques. Optimizing the data transfer, such as via shared memory, remains an area for improvement to fully leverage this coupler's ability to integrate online-adaptive ML parameterizations within large-scale ESM simulations, which is the main goal for this study.

In this study, we investigate the integration of ML parameterizations into Fortran-based ESM models by establishing a flexible interface that enables the invocation of ML algorithms in Python from Fortran. This integration offers access to any Python codes from Fortran, including a diverse range of ML frameworks, such as PyTorch, TensorFlow, and Scikit-learn, which can effectively be utilized for parameterizing intricate atmospheric and other climate system processes. The coupling of the Fortran model and the Python ML code needs to be performed for thousands of model columns and over thousands of timesteps for a typical model simulation. Therefore, it is crucial for the coupling interface to be both robust and efficient. We showcase the feasibility and benefits of this approach through case studies that involve the parameterization of deep convection and wildfire processes in ESMs. The two cases demonstrate the robustness and efficiency of the coupling interface. The focus of this paper is on documenting the coupling between the Fortran ESM and the ML algorithms and systematically evaluating the computational efficiency and memory usage of different ML frameworks (such as Pytorch and TensorFlow), different ML algorithms, and different configuration of a climate model. The assessment of the scientific performance of the ML emulators will be addressed in follow-on papers. The showcase examples emphasize the potential for high modularity and reusability by separating the ML components into Python modules. This modular design facilitates independent development and testing of ML-based parameterizations by researchers. It enables easier code maintenance, updates, and the adoption of state-of-the-art ML techniques

Deleted: including

with only minimal disrupting the existing Fortran infrastructure. Ultimately, this advancement will contribute to enhanced predictions and a deeper comprehension of the evolving climate of our planet. It is important to note that the current interface only supports executing deep learning algorithms on CPUs and does not support running them on GPUs.

Deleted: out

Deleted: important to note

Deleted: ¶

The rest of this manuscript is organized as follows: Section 2 presents the detailed interface that integrates ML into Fortran-based ESM models. Section 3 discusses the performance of the interface and presents its application in two case studies. Finally, Section 4 provides a summary of the findings and a discussion of their implications.

2. General design of the ML interface

2.1 Architecture of the ML interface

We developed an interface using shared memory to enable two-way coupling between Fortran and Python (Figure 1). The ESM used in the demonstration in Figure 1 is the U.S. Department of Energy (DOE) Energy Exascale Earth System Model (E3SM; Golaz et al., 2019, 2022). Because Fortran cannot directly call Python, we utilized C as an intermediary since Fortran can call C functions. This approach leverages C as a data hub to exchange information without requiring a framework-specific binding like KFB. As a result, our interface supports invoking any Python-based ML package such as PyTorch, TensorFlow, and scikit-learn from Fortran. While C can access Python scalar values through the built-in PyObject_CallObject function from the Python C API, we employed Cython for its ability to transfer array data between the languages. Using Cython, multidimensional data structures can be efficiently passed between Fortran and Python modules via C, allowing for flexible training of ML algorithms within ESMs.

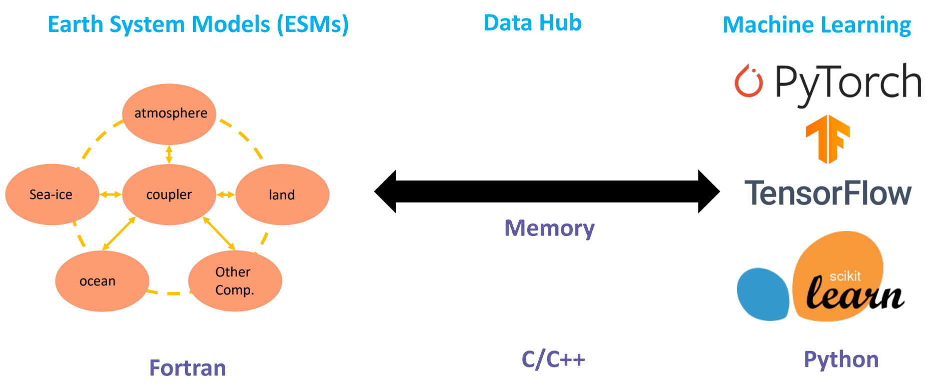
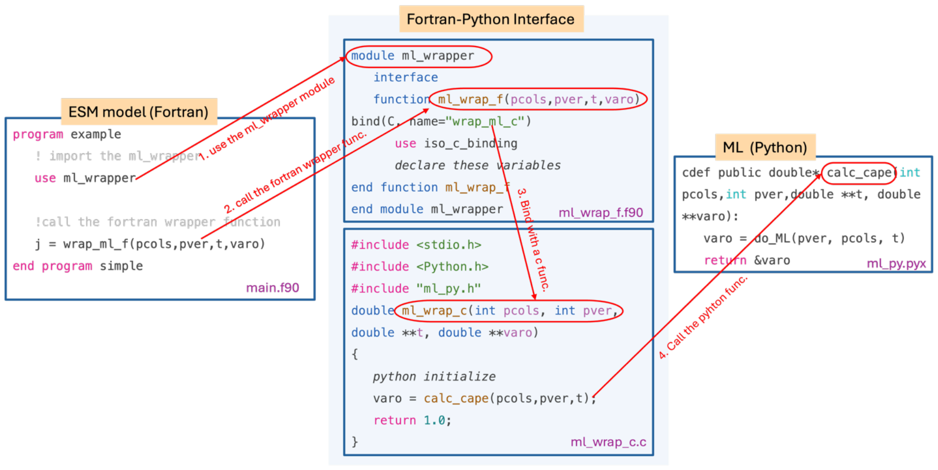


Figure 1. The interface of the ML bridge for two-way communication via memory between Fortran ESM and Python ML module.

2.2 Code structure

Figure 2 illustrates how the framework operates using toy code example. The Fortran-Python interface comprises a Fortran wrapper and C wrapper files, which are bound together. The Fortran-based ESM first imports the Fortran wrapper, allowing it to call wrapper functions with input and output memory addresses. The interface then passes these memory addresses to the Python-based ML module, which performs the ML predictions and returns the output address to the Fortran model.



```fortran
program example
   ! import the ml_wrapper
   use ml_wrapper

   !call the fortran wrapper function
   j = wrap_ml_f(pcols,pver,t,varo)
end program simple
```

```fortran
module ml_wrapper
   interface
   function ml_wrap_f(pcols,pver,t,varo)
bind(C, name="wrap_ml_c")
      use iso_c_binding
      declare these variables
end function ml_wrap_f
end module ml_wrapper
```

```c
#include <stdio.h>
#include <Python.h>
#include "ml_py.h"
double ml_wrap_c(int pcols, int pver,
double **t, double **varo)
{
   python initialize
   varo = calc_cape(pcols,pver,t);
   return 1.0;
}
```

```python
cdef public double* calc_cape(int
pcols,int pver,double **t, double
**varo):
   varo = do_ML(pver, pcols, t)
   return &varo
```

Figure 2. Toy code illustrating the Fortran-Python interface.

Formatted: Font: (Default) Arial, (Asian) Arial

Formatted: Centered

When coupling the Python ML module with the real model using the interface, additional steps should be considered: 1. The ML module should remain active throughout the model simulations, without any Python finalization calls, ensuring it is continuously available. 2. The Python module should load the trained ML model and any required global data only once, rather than at each simulation step. This one-time initialization process improves efficiency and prevents unnecessary repetition. On the Fortran ESM side, the init_ml() function is called within the atm_init_mct module to load the ML model and global data (shown in Figure 3). Then, similar to the toy code, we call the wrapper function, pass input variables to Python for ML predictions, and return the results to the Fortran side. 3. When compiling the complex system, which includes Python, C, Cython, and Fortran code, the Python path should be specified in the CFLAGS and LDFLAGS. It is important to note that without the position-independent compiling flag (-fPIC), the hybrid system will only work on a single node and may cause segmentation faults on multiple nodes. Including it can resolve this issue, allowing multi-node compatibility.

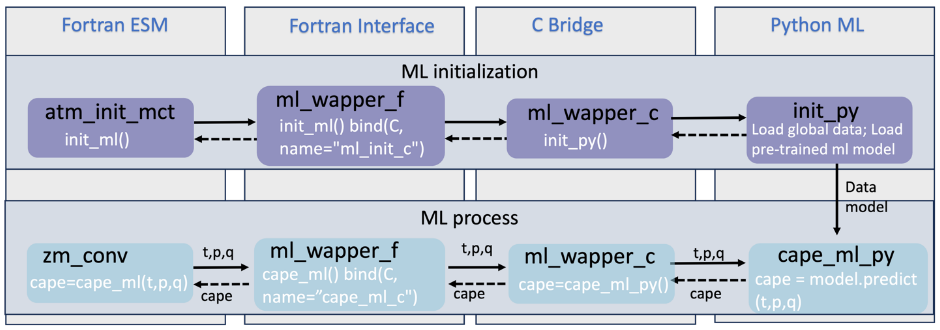


Figure 3. The code structure of the ML bridge interface using the ML closure in deep convection as an example.

In traditional ESMs, sub-grid scale parameterization routines such as convection parameterizations are often calculated separately for each vertical column of the model domain. Meanwhile, the domain is typically decomposed horizontally into 2D chunks that can be solved in parallel using MPI processes. Each CPU core/MPI process is assigned a number of chunks of model columns to update asynchronously (Figure 4). Our interface takes advantage of this existing parallel decomposition by designing the ML calls to operate over all columns simultaneously within each chunk, rather than invoking the ML scheme individually for each column. This allows the coupled model-ML system to leverage parallelism in the

Deleted: The interface consists of two stages. The first stage involves initializing the ML environment, which persists throughout the model simulations. On the Fortran ESM side, the init_ml() function is called in the atm_init_mct module. Through the Fortran Interface and C Bridge, the corresponding function in the Python ML component is invoked. This function loads the ML-related global data and the trained ML algorithm. This initialization process is performed only once to enhance efficiency and avoid unnecessary repetition during the simulations. The second stage involves the actual invocation of the ML process. The example here is an ML-based closure for the deep convection parameterization. We aim to utilize ML to calculate Convective Available Potential Energy (CAPE) by utilizing an ML emulator based on high-resolution cloud-resolving model simulations. We call the cape_ml function in the Fortran module zm_conv, providing temperature, pressure, and humidity as input variables, and defining the returned CAPE from the ML side. Through the Fortran Interface and C Bridge, these three variables are passed to the Python ML component. In the Python ML component, the received variables, along with other pre-loaded global data and the trained ML algorithm, are used to calculate the ML-based CAPE. The calculated result is then returned to the Fortran ESM. The Fortran ESM utilizes this ML-derived CAPE to determine how convection will evolve.

Deleted: ¶

Deleted: 2

Deleted: 3

neural network computations. If the ML were called separately for every column, parallel efficiencies would not be realized. By aggregating inputs over the chunk-scale prior to interfacing with Python, performance is improved through better utilization of multi-core and GPU-based ML capabilities during parameterization calculations.

Deleted: The Python, C, Cython and Fortran code components are compiled together into a unified executable file. Table 1 shows the detailed steps to enable the ML bridge interface in E3SM.

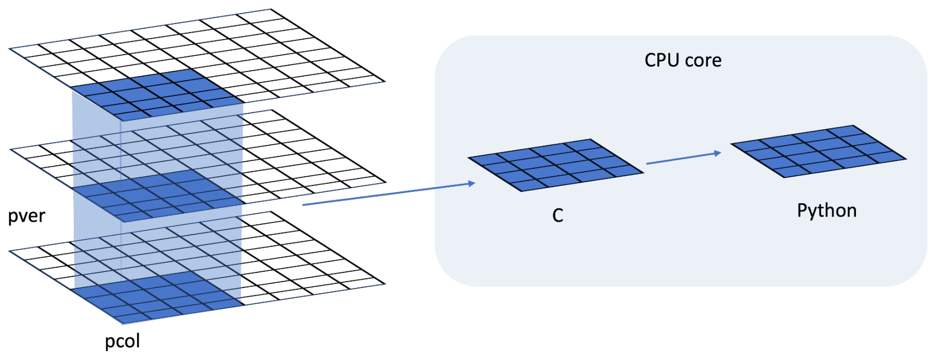


Figure 4. Data and system structure. The model domain is decomposed into chunks of columns. pver refers to number of pressure vertical levels. A chunk contains multiple columns (up to pcol). Multiple chunks can be assigned to each CPU core.

Deleted: 3

Deleted: **Table 1.** The steps to enable the ML bridge framework in E3SM

Deleted: ¶

Formatted: Font: (Default) Arial, (Asian) Arial

Formatted: Centered

3. Results

The framework explained in the previous section provides seamless support for various ML parameterizations and various ML frameworks, such as PyTorch, Tensorflow, and Scikit-learn. To demonstrate the versatility of this framework, we applied it in two distinct case applications. The first application replaces the conventional CAPE-based trigger function in a deep convection parameterization with a machine-learned trigger function. The second application involves a ML-based wildfire model that interacts bidirectionally with the ESM. We provide a brief introduction to these two cases. Detailed descriptions and evaluations will be presented in separate papers.

Deleted: t

The framework's performance is influenced by two primary factors: increasing memory usage and increasing computational overhead. Firstly, maintaining the Python environment fully persistent in memory throughout model simulations can impact memory usage, especially for large ML algorithms.

This elevated memory footprint increases the risk of leaks or crashes as simulations progress. Secondly, executing ML components within the Python interpreter inevitably introduces some overhead compared to the original ESMs. The increased memory requirements and decreased computational efficiency associated with these considerations can impact the framework's usability, flexibility, and scalability for different applications.

To comprehensively assess performance, we conducted a systematic evaluation of various ML frameworks, ML algorithms, and physical models. This evaluation is built upon the foundations established for evaluating the ML trigger function in the deep convection parameterization.

3.1 Application cases

3.1.1 ML trigger function in deep convection parameterization

In General Circulation Models, uncertainties in convection parameterizations are recognized to be closely linked to the convection trigger function used in these schemes (Bechtold et al., 2004; Xie et al., 2004, 2019; Xie & Zhang, 2000; Lee et al., 2007) . The convective trigger in a convective parameterization determines when and where model convection should be triggered as the simulation advances. In many convection parameterizations, the trigger function consists of a simple, arbitrary threshold for a physical quantity, such as convective available potential energy (CAPE). Convection will be triggered if the CAPE value exceeds a threshold value.

In this work, we use this interface to test a newly developed ML trigger function in E3SM. The ML trigger function was developed with the training data originating from simulations performed using the kilometer-resolution (1.5 km grid spacing). Met Office Unified Model Regional Atmosphere 1.0 configuration (Bush et al., 2020). Each simulation consists of a limited area model (LAM) nested within a global forecast model providing boundary conditions (Walters et al., 2017; Webster et al., 2008). In total 80 LAM simulations were run located so as to sample different geographical regions worldwide. Each LAM was run for 1 month, with 2-hourly output, using a grid-length of 1.5 km, a 512 x 512 domain, and a model physics package used for operational weather forecasting. The 1.5 km data is coarse-grained to several scales from 15 to 144 km.

A two-stream neural network architecture is used for the ML model. The first stream takes profiles of temperature, specific humidity and pressure across 72 levels at each scale as inputs and passes them through a 4-layer convolutional neural network (CNN) with kernel sizes of 3, to extract large scale

Deleted: Convection plays a vital role in atmospheric processes, such as precipitation formation, heat and moisture transport, and energy redistribution (Arakawa, 2004; Arakawa & Schubert, 1974). However, the deficiencies in convection parameterizations constitute one of the principal sources of uncertainties

Deleted: i

Deleted: (D. A. Randall, 2013) . Some

Deleted: Figure 4a illustrates how the CAPE-based trigger function works

Deleted: , such as 70 J/kg used in E3SM version 1

Deleted: and apply it to

Deleted:

Deleted: The ML trigger function was developed with

Deleted:

Deleted: t

Deleted:

Deleted: originating

Deleted:

Deleted: This physics package does not include a convective parameterization scheme, but does include a representation of fractional cloudiness (Bush et al., 2020).

Deleted: , comparable to the scale a global model might be run at

Deleted: At each scale,

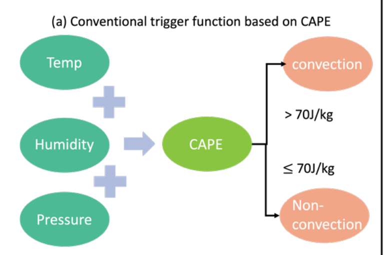
Deleted: we assess whether individual pixels can be considered to be buoyant cloudy updrafts (BCU, e.g. Hartmann et al., 2019; Swann, 2001). Here, the threshold for buoyant is local virtual temperature more than 0.1 K warmer than the average at that scale and height. Cloudy is defined whenever the fractional cloud cover is greater than 0.0 and updraft is defined as vertical ascent larger than 0.2 m/s. In each averaging region, the number of grid points that meet all three criteria are counted and saved as a profile of BCU fraction.

Deleted: ¶

features. The second stream takes mean orographic height, standard deviation of orographic height, land fraction and the size of the grid-box as inputs. The outputs of the two streams are then combined and fed into a 2-layer fully connected network to allow the ML model to leverage both atmospheric and surface features when making its predictions. The output is a binary variable indicating whether the convection happens, based on the condition of buoyant cloudy updrafts (BCU, e.g. Hartmann et al., 2019; Swann, 2001). If there are 3 contiguous levels where the predicted BCU is larger than 0.05, the convection scheme is triggered. Once trained, the CNN is coupled to E3SM and thermodynamic information from E3SM is passed to it to predict the trigger condition. Then, the predicted result is returned to E3SM.

Deleted: The output pf the ML model is a profile of BCU.

Deleted: the profile of BCU. If there are 3 contiguous levels where the predicted BCU is larger than 0.05, the convection scheme is triggered.



Deleted:

Deleted: Figure 4. Structure of traditional CAPE-based and the new ML BCU-based trigger function. The rectangles in LAM represent the LAM domains.

Deleted: ¶

Deleted: The ML trigger function is implemented using this two-stream architecture and coupled with the E3SM model using the framework described in Section 2.

Deleted:

Figure 5 shows the comparison of annual mean precipitation between the control run using the traditional CAPE-based trigger function and the run using the ML BCU trigger function. The ML BCU scheme demonstrates reasonable spatial patterns of precipitation, similar to the control run, with comparable root-mean-square error and spatial correlation. Additional experiments exploring the definition of BCU and varying the thresholds along with an in-depth analysis will be presented in a follow-up paper.

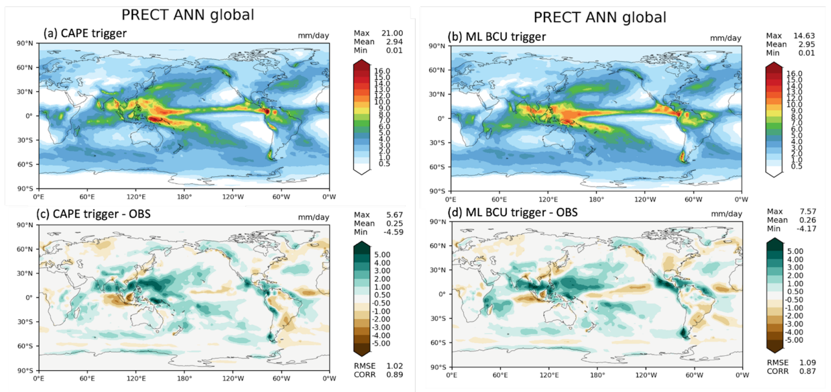


Figure 5. Comparison of annual mean precipitation between the control run using the CAPE-based trigger function (a, c) and the run using the ML BCU trigger function (b, d).

3.1.2 ML learning fire model

Predicting wildfire burned area is challenging due to the complex interrelationships between fires, climate, weather, vegetation, topography, and human activities (Huang et al., 2020). Traditionally,

Deleted: Wildfires in the United States have significantly increased in frequency and intensity in recent decades, resulting in substantial direct and indirect losses (Iglesias et al., 2022). …

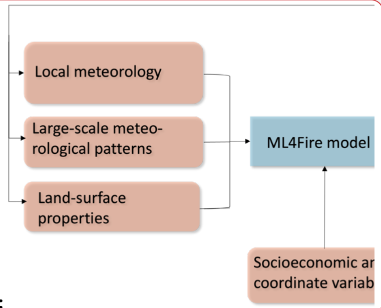
statistical methods like multiple linear regression have been applied, but are limited in the number and diversity of predictors considered (Yue et al., 2013). In this study, we develop a coupled fire-land-atmosphere framework that uses machine learning to predict wildfire area, enhancing long-term burned area projections and assessing fire impacts by enabling simulations of interactions among fire, atmosphere, land cover, and vegetation.

Deleted: Alternatively, ML algorithms that capture statistical relationships between the burned area and environmental factors have shown promising burned area prediction (Kondylatos et al., 2022; Li et al., 2023; Wang et al., 2022, 2023). However, improving long-term burned area projections and evaluating fire impacts requires the coupling of the fire model to an earth system model, which allows simulations of the interactions between the fire, atmosphere, land cover and vegetation (Huang et al., 2021). To achieve this, we develop a coupled fire-land-atmosphere framework using ML. …

The ML algorithm is trained using a monthly dataset, which includes the target variable of burned area, as well as various predictor variables. These predictors encompass local meteorological data (e.g., surface temperature, precipitation), land surface properties (e.g., monthly mean evapotranspiration and surface soil moisture), and socioeconomic variables (e.g., gross domestic product, population density), as described by Wang et al. (2022). In the coupled fire-land-atmosphere framework, meteorology variables and land surface properties are provided by the E3SM. We use the eXtreme Gradient Boosting algorithm implemented in Scikit-Learn to train the ML fire model. Figure 6 demonstrates that the ML4Fire model exhibits superior performance in terms of spatial distribution compared to process-based fire models, particularly in the Southern US region. Detailed analysis will be presented in a separate paper. The ML4Fire model has proven to be a valuable tool for studying vegetation-fire interactions, enabling seamless exploration of climate-fire feedbacks.

Deleted: , as illustrated in Figure 6

Deleted: 7



Deleted:

Deleted: Figure 6. Structure of ML fire model (ML4Fire) coupled into E3SM model.

Deleted: ¶

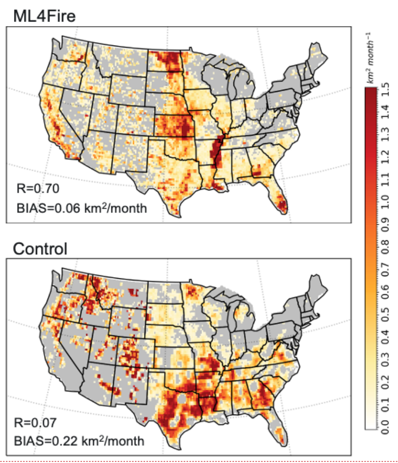


Figure 6. Comparison between ML4Fire model and process-based fire model against the historical burned area from Global Fire Emissions Database 5 from 2001-2020. R and BIAS are the spatial pattern correlation and difference against the observation, respectively.

Deleted: 7

3.2 Performance of different ML frameworks

The Fortran-Python bridge ML interface supports various ML frameworks, including PyTorch, TensorFlow, and scikit-learn. These ML frameworks can be trained offline using kilometer-scale high-resolution models (such as the ML trigger function) or observations (ML fire model). Once trained, they can be plugged into the ML bridge interface through different API interfaces specific to each framework. The coupled ML algorithms are persistently resident in memory, just like the other ESM components. During each step of the process, the performance of the full system is significantly affected by memory usage. If memory consumption increases substantially, it may lead to memory leaks as the number of time step iteration increases. In addition, Python, being an interpreted language, is typically considered to have slower performance compared to compiled languages like C/C++ and Fortran. Therefore, incorporating Python may decrease computational performance. We examine the memory usage and computational performance across various ML frameworks based on implementing the ML trigger function in E3SM. The ML algorithm is implemented as a two-stream CNN model using Pytorch and TensorFlow frameworks, as well as XGBoost using the Scikit-learn package. It should be noted that XGBoost, a boosting tree-based model, is a completely different type of ML model compared to the CNNs, which are the type of deep neural network.

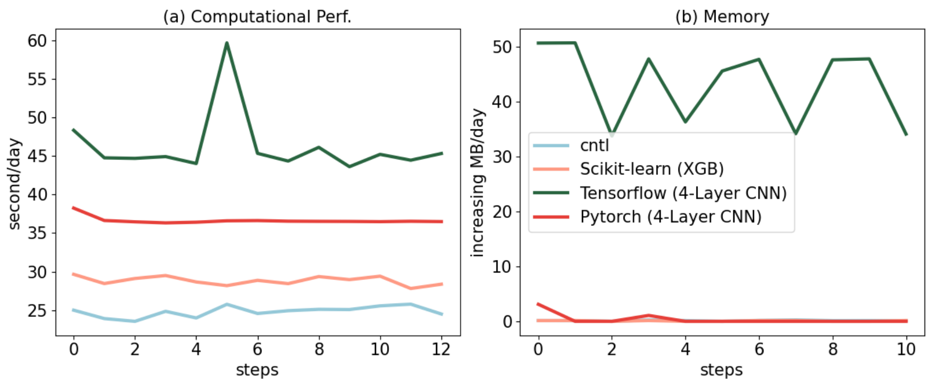


Figure 7. Computational and memory overhead as the simulation progresses for coupling the ML trigger function with the E3SM model. The x-axis represents the simulated time step. The y-axis of (a) represents the simulation speed measured in seconds per day (indicating the number of seconds required to simulate one day). The y-axis of (b) represents the relative increase in memory usage for Scikit-learn, TensorFlow,

Deleted: 8

and PyTorch compared with CNTL. CNTL represents the original simulation without using the ML framework.

Figure 7 illustrates the computational and memory overhead associated with the ML parameterization using different ML frameworks. It shows that XGBoost only exhibits a 20% increase in the simulation time required for simulating one day due to its simpler algorithm. For more complex neural networks, PyTorch incurs a 52% overhead, while TensorFlow's overhead is almost 100% – about two times as much as the overhead by PyTorch. In terms of memory usage, we use the highwater memory metric (Gerber & Wasserman, 2013), which represents the total memory footprint of a process. Scikit-learn and PyTorch do not show any significant increase in memory usage. However, TensorFlow shows a considerable increase up to 50MB per simulation day per MPI process element. This is significant because for a node with 48 cores, it would equate to an increase of around 2GB per simulated day on that node. This rapid memory growth could quickly lead to a simulation crash due to insufficient memory during continuous integrations, preventing the use in practical simulations. Our findings show that the TensorFlow prediction function does not release memory after each call. Therefore, we recommend using PyTorch for complex deep learning algorithms and Scikit-learn for simpler ML algorithms to avoid these potential memory-related issues when using TensorFlow.

Previous work, such as Brenowitz & Bretherton (2018, 2019) has utilized the CFFI package to establish communication between Fortran ESM and ML Python. As described in the Introduction, while CFFI offers flexibility in supporting various ML packages, it does have certain limitations. To pass variables from Fortran to Python, the approach relies on global data structures to store all variables, including both the input from Fortran to Python and the output returning to Fortran. Consequently, this package results in additional memory copy operations and increasing overall memory usage. In contrast, our interface takes a different approach by utilizing memory references to transfer data between Fortran and Python, avoiding the need for global data structures and the associated overhead. This allows for a more efficient data transfer process.

In Figure 8, we present a comparison between the two frameworks by testing the different number of elements passed from Fortran to Python. The evaluation is based on a demo example that focuses solely on declaring arrays and transferring them from Fortran to Python, rather than a real E3SM simulation. Figure 8a illustrates the impact of the number of passing elements on the overhead of the two interfaces. As the number of elements exceeds 10^4, the overhead of CFFI becomes significant. When the number surpasses 10^6, the overhead of CFFI is nearly ten times greater than that of our interface. Regarding

Deleted: 8

Deleted:

Deleted:

Deleted:

memory usage, our interface maintains a stable memory footprint of approximately 60MB. Even as the number of elements increases, the memory usage only shows minimal growth. However, for CFFI, the memory usage starts at 80MB, which is 33% higher than our interface. As the number of elements reaches 10^6, the memory overhead for CFFI dramatically rises to 180MB, twice as much as our interface.

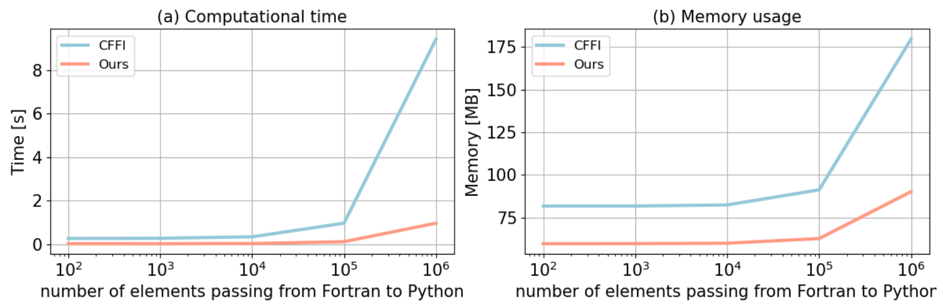


Figure 8. Comparison of our framework and the CFFI framework in terms of computational time and memory usage. The x-axis represents the number of elements transferred from Fortran to Python, while the y-axis displays the total time (a) and total memory usage (b) for a demonstration example. The evaluations presented are based on the average results obtained from 5 separate tests.

3.3 Performance of ML algorithms of different complexities

ML parameterizations can be implemented using various deep learning algorithms with different levels of complexity. The computational performance and memory usage can be influenced by the complexity of these algorithms. In the case of the ML trigger function, a two-stream four-layer CNN structure is employed. We compare this structure with other ML algorithms such as Artificial Neural Network (ANN) and Residual Network (ResNet), whose structures are detailed in Table 1. We selected these three ML algorithms because they are commonly used in previous ML parameterization approaches, such as (Brenowitz & Bretherton, 2019; Han et al., 2020; Wang et al., 2022). Systematically evaluating the hybrid system with these ML methods using our interface can help identify bottlenecks and improve the system computational performance. These algorithms are implemented in PyTorch. The algorithm's complexity is measured by the number of parameters, with the CNN having approximately 60 times more parameters than ANN, and ResNet having roughly 1.5 times more parameters than CNN.

Deleted:

Deleted: 9

Deleted: 2

Table 1. The structure and number of parameters of each ML algorithms.

Algorithms	Structure	# of parameters
ANN	3 x Linear	121,601
CNN	4 x Conv2d + 2 x Linear	7,466,753
ResNet	17 x Conv2d + 1 x Linear	11,177,025

Figure 9 presents a comparison of the memory and computational costs between the CNTL run without deep learning parameterization and the hybrid run with various deep learning algorithms. The same specific process-element layout (placement of ESM component models on distributed CPU cores) is used for all the simulations. Deep learning algorithms incur a significant yet affordable increase in memory overhead, with at least a 20% increase compared to the CNTL run (Figure 9a). This is primarily due to the integration of ML algorithms into the ESM, which persist throughout the simulations. Although there is a notable increase in complexity among the deep learning algorithms, their memory usage only shows a slight rise. This is because the memory increment resulting from the ML parameters is relatively small. Specifically, ANN requires 1MB of memory, CNN requires 60MB, and the ResNet algorithms requires 85MB, which are calculated based on the number of parameters in each algorithm. When comparing these values to the memory consumption of the CNTL run, which is approximately 3000MB, the additional parameters' incremental memory consumption is not substantial. However, when we use 128 MPI processes per node, it could bring the total memory requirement to approximately 460 GB per node. If the available hardware memory is less than this, the process layout must be adjusted accordingly.

In terms of computational performance, the Python-based ML calls inevitably introduce some overhead. However, as shown in Figure 9b, the performance decrease is not substantial. The simple ANN model reduces performance by only about 10% compared to the CNTL run, while even the more complex ResNet model results in a 35% decrease. In contrast, Wang et al. (2022) reported a 100% overhead in their interface, which transfers parameters via files. It is worth noting that in this study, the deep learning algorithms are executed on CPUs. To enhance computational performance, future work could consider utilizing GPUs for acceleration.

In addition, we develop a performance model to estimate computational performance for the hybrid model using different ML model sizes and complexities. This performance model, based on linear regression, predicts the computational ratio relative to the CNTL run by taking the number of ML

Deleted: 2

Deleted: 10

Deleted: A

Deleted: 10

Deleted: However, there is a significant decrease in computational performance as the complexity of the deep learning algorithms increases (Figure 10b). This is primarily due to the larger number of parameters in neural networks, which require more forward computations

Deleted: .

parameters as input, shown in Figure 9b. It provides a simple yet effective way to capture this relationship and serves as a valuable tool for performance prediction when incorporating more complicated ML models.

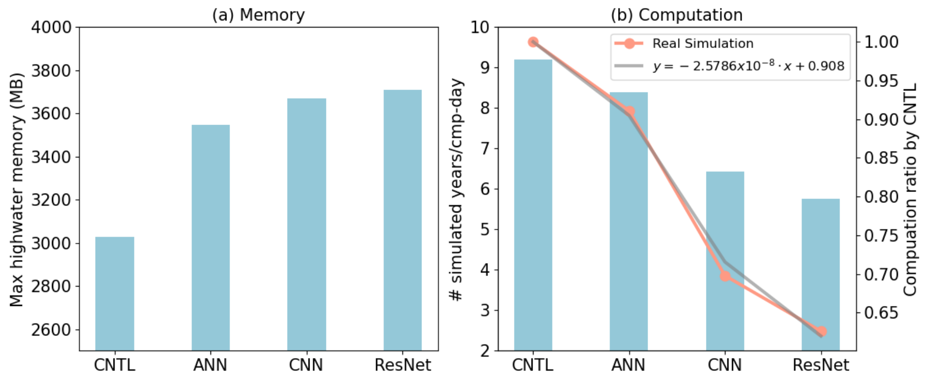


Figure 9. Comparison of CNTL and the hybrid model using various ML algorithms in terms of memory and computation. CNTL is the default run without ML parameterizations. In (b), the left y-axis represents the actual number of simulated years per day, while the right y-axis shows the relative performance compared to the CNTL run (orange line). The gray line illustrates the regression between the number of ML parameters (x) and the relative performance of the hybrid system (y).

Deleted: 10

3.4 Performance for physical models of different complexities

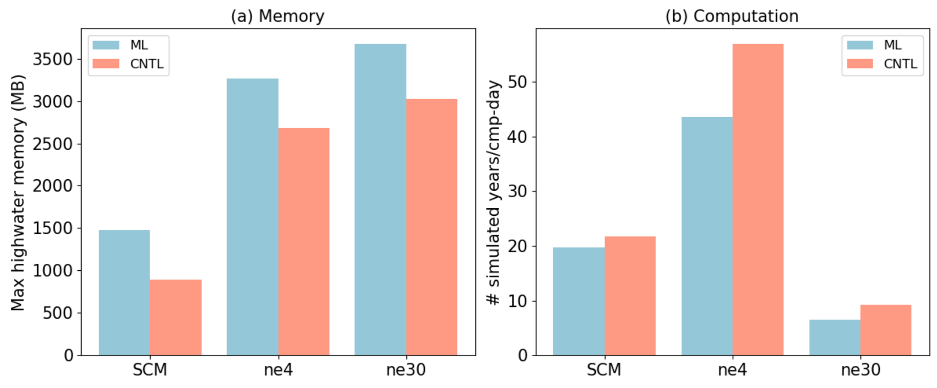


Figure 10. Compassion of CNTL and ML for various ESMs in terms of memory and computation. The ESM configuration include SCM, ultra-low resolution model (ne4) and nominal low-resolution model (ne30).

Deleted: 11

ML parameterization can be applied to various ESM configurations, for example, with the E3SM Atmosphere Model (EAM), we experiment with Single Column Model (SCM), the ultra low-resolution model of EAM (ne4), and the nominal low resolution model of EAM (ne30) configurations. The SCM consists of one single atmosphere column of a global EAM (Bogenschutz et al., 2020; Gettelman et al., 2019). ne4 has 384 columns, with each column representing the horizontal resolution of 7.5°. ne30 is the default resolution for EAM and comprises 21,600 columns, with each column representing the horizontal resolution of 1°. In the case of the ML trigger function, the memory overhead is approximately 500MB for all configurations due to the loading of the ML algorithm, which does not vary with the configuration of the ESM.

Regarding computational performance, SCM utilizes 1 process, ne4 employs 1 node with 64 processes, and ne30 utilizes 10 nodes with each node using 128 processes. In the case of SCM, the overhead attributed to the ML parameterization is approximately 9% due to the utilization of only 1 process. However, for ne4 and ne30, the overhead is 23% and 28% respectively (Figure 10). The increasing computational overhead is primarily due to resource competition when multiple processes are used within a single node. It is noted that although there is a significant computational gap between ML and CNTL for ne4, the relative performance between ML and CNTL for ne4 is approximately 76.7%, which is close to ne30 at 71.4%.

Deleted: 11

4. Discussion and Conclusion

ML algorithm can learn detailed information about cloud processes and atmospheric dynamics from kilometer-scale models and observations and serves as an approximate surrogate for the kilometer-scale model. Instead of explicitly simulating kilometer-scale processes, the ML algorithms can be designed to capture the essential features and relationships between atmospheric variables by training on available kilometer-scale data. The trained algorithms can then be used to develop parameterizations for use in models at coarser resolutions, reducing the computational and memory costs. By using ML parameterizations, scientists can effectively incorporate the insights gained from kilometer-scale models for coarser-resolution simulations. Through learning the complex relationships and patterns present in the

Deleted: In this study, we develop a novel Fortran-Python interface for developing ML parameterizations.

high-resolution data, the ML-based parameterizations have the potentials to more accurately represent cloud processes and atmospheric dynamics in the ESMs. This approach strikes a balance between computational efficiency and capturing critical processes, enabling more realistic simulations and predictions while minimizing computational resources. All these potential benefits in turn promote innovative developments to facilitate increasing and more efficient use of ML parameterizations.

In this study, we develop a novel Fortran-Python interface for developing ML parameterizations. This interface demonstrates feasibility in supporting various ML frameworks, such as PyTorch, TensorFlow, and Scikit-learn and enables the effective development of new ML-based parameterizations to explore ML-based applications in ESMs. Through two cases - a ML trigger function in convection parameterization and a ML wildfire model - we highlight high modularity and reusability of the framework. We conduct a systematic evaluation of memory usage and computational overhead from the integrated Python codes.

Based on our performance evaluation, we observe that coupling ML algorithms using TensorFlow into ESMs can lead to memory leaks. As a recommendation, we suggest using PyTorch for complex deep learning algorithms and Scikit-learn for simple ML algorithms for the Fortran-Python ML interface.

The memory overhead primarily arises from loading ML algorithms into ESMs. If the ML algorithms are implemented using PyTorch or Scikit-learn, the memory usage will not increase significantly. The computational overhead is influenced by the complexity of the neural network and the number of processes running on a single node. As the complexity of the neural network increases, more parameters in the neural network require forward computation. Similarly, when there are more processes running on a single node, the integrated Python codes introduce more resource competition.

Deleted: gradient

Although this interface provides a flexible tool for ML parameterizations, it does not currently utilize GPUs for ML algorithms. In Figure 3, it is shown that each chunk is assigned to a CPU core. However, to effectively leverage GPUs, it is necessary to gather the variables from multiple chunks and pass them to the GPUs. Additionally, if an ESM calls the Python ML module multiple times in each time step, the computational overhead becomes significant. It is crucial to gather the variables and minimize the number of calls. In the future, we will enhance the framework to support this mechanism, enabling GPU utilization and overall performance improvement.

Acknowledge

This work was primarily supported by the Energy Exascale Earth System Model (E3SM) project of the Earth and Environmental System Modeling program, funded by the US Department of Energy, Office of Science, Office of Biological and Environmental Research. Research activity at BNL was under the Brookhaven National Laboratory contract DE-SC0012704 (Tao Zhang, Wuyin Lin). The work at LLNL was performed under the auspices of the US Department of Energy by the Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. The work at PNNL is performed under the Laboratory Directed Research and Development Program at the Pacific Northwest National Laboratory. PNNL is operated by DOE by the Battelle Memorial Institute under contract DE-A05-76RL01830.

Author contribution

TZ developed the Fortran-Python Interface. CM and JR contributed the ML model for the trigger function. YL contributed the ML model for the wire fire model. TZ and MZ assessed the performance of the ML trigger function. TZ took the lead in preparing the manuscript, with valuable edits from CM, MZ, WL, SX, YL, KW, and JR. All the co-authors provided valuable insights and comments for the manuscript.

Conflict of Interest

The authors declare that they have no conflict of interest.

Data Availability Statement

The Fortran-Python interface for developing ML parameterizations can be archived at https://doi.org/10.5281/zenodo.11005103 (Zhang et al., 2024) and can be also accessed at https://github.com/tzhang-ccs/ML4ESM. The E3SM model can be accessed at https://zenodo.org/records/12175988. The dataset for machine learning trigger function can be accessed at https://zenodo.org/records/12205917. The dataset for machine learning wild fire can be accessed at https://zenodo.org/records/12212258.

Deleted: https://doi.org/10.11578/E3SM/dc.20240301.3 (E3SM Project, 2024)

References

Bechtold, P., Chaboureau, J.-P., Beljaars, A., Betts, A. K., Köhler, M., Miller, M., & Redelsperger, J.-L. (2004). The simulation of the diurnal cycle of convective precipitation over land in a global model. *Quarterly Journal of the Royal Meteorological Society*, *130*(604), 3119–3137. https://doi.org/10.1256/qj.03.103

Bogenschutz, P. A., Tang, S., Caldwell, P. M., Xie, S., Lin, W., & Chen, Y.-S. (2020). The E3SM version 1 single-column model. *Geoscientific Model Development*, *13*(9), 4443–4458. https://doi.org/10.5194/gmd-13-4443-2020

Brenowitz, N. D., & Bretherton, C. S. (2018). Prognostic validation of a neural network unified physics parameterization. *Geophysical Research Letters*, *45*(12), 6289–6298. https://doi.org/10.1029/2018gl078510

Brenowitz, N. D., & Bretherton, C. S. (2019). Spatially extended tests of a neural network parametrization trained by coarse-graining. *Journal of Advances in Modeling Earth Systems*, *11*(8), 2728–2744. https://doi.org/10.1029/2019ms001711

Bush, M., Allen, T., Bain, C., Boutle, I., Edwards, J., Finnenkoetter, A., Franklin, C., Hanley, K., Lean, H., Lock, A., Manners, J., Mittermaier, M., Morcrette, C., North, R., Petch, J., Short, C., Vosper, S., Walters, D., Webster, S., … Zerroukat, M. (2020). The first Met Office Unified Model–JULES Regional Atmosphere and Land configuration, RAL1. *Geoscientific Model Development*, *13*(4), 1999–2029. https://doi.org/10.5194/gmd-13-1999-2020

Chen, G., Wang, W., Yang, S., Wang, Y., Zhang, F., & Wu, K. (2023). A Neural Network-Based Scale-Adaptive Cloud-Fraction Scheme for GCMs. *Journal of Advances in Modeling Earth Systems*, *15*(6), e2022MS003415. https://doi.org/10.1029/2022MS003415

E3SM Project, D. (2024). *Energy Exascale Earth System Model v3.0.0* [Computer software]. [object Object]. https://doi.org/10.11578/E3SM/DC.20240301.3

Gerber, R., & Wasserman, H. (2013). *High Performance Computing and Storage Requirements for Biological and Environmental Research Target 2017* (LBNL-6256E). Lawrence Berkeley National Lab. (LBNL), Berkeley, CA (United States). https://doi.org/10.2172/1171504

Gettelman, A., Gagne, D. J., Chen, C.-C., Christensen, M. W., Lebo, Z. J., Morrison, H., & Gantos, G. (2021). Machine Learning the Warm Rain Process. *Journal of Advances in Modeling Earth Systems*, *13*(2), e2020MS002268. https://doi.org/10.1029/2020MS002268

Gettelman, A., Truesdale, J. E., Bacmeister, J. T., Caldwell, P. M., Neale, R. B., Bogenschutz, P. A., & Simpson, I. R. (2019). The Single Column Atmosphere Model Version 6 (SCAM6): Not a Scam but a Tool for Model Evaluation and Development. *Journal of Advances in Modeling Earth Systems*, *11*(5), 1381–1401. https://doi.org/10.1029/2018MS001578

Golaz, J.-C., Caldwell, P. M., Van Roekel, L. P., Petersen, M. R., Tang, Q., Wolfe, J. D., Abeshu, G., Anantharaj, V., Asay-Davis, X. S., Bader, D. C., Baldwin, S. A., Bisht, G., Bogenschutz, P. A., Branstetter, M., Brunke, M. A., Brus, S. R., Burrows, S. M., Cameron-Smith, P. J., Donahue, A. S., … Zhu, Q. (2019). The DOE E3SM Coupled Model Version 1: Overview and Evaluation at Standard Resolution. *Journal of Advances in Modeling Earth Systems*, *11*(7), 2089–2129. https://doi.org/10.1029/2018MS001603

Golaz, J.-C., Van Roekel, L. P., Zheng, X., Roberts, A. F., Wolfe, J. D., Lin, W., Bradley, A. M., Tang, Q., Maltrud, M. E., Forsyth, R. M., Zhang, C., Zhou, T., Zhang, K., Zender, C. S., Wu, M., Wang, H., Turner, A. K., Singh, B., Richter, J. H., … Bader, D. C. (2022). The DOE E3SM Model Version 2: Overview of the Physical Model and Initial Model Evaluation. *Journal of Advances in Modeling Earth Systems*, *14*(12), e2022MS003156. https://doi.org/10.1029/2022MS003156

Han, Y., Zhang, G. J., Huang, X., & Wang, Y. (2020). A Moist Physics Parameterization Based on Deep Learning. *Journal of Advances in Modeling Earth Systems*, *12*(9), e2020MS002076. https://doi.org/10.1029/2020MS002076

Hartmann, D. L., Blossey, P. N., & Dygert, B. D. (2019). Convection and Climate: What Have We Learned from Simple Models and Simplified Settings? *Current Climate Change Reports*, *5*(3), 196–206. https://doi.org/10.1007/s40641-019-00136-9

Hourdin, F., Mauritsen, T., Gettelman, A., Golaz, J.-C., Balaji, V., Duan, Q., Folini, D., Ji, D., Klocke, D., Qian, Y., Rauser, F., Rio, C., Tomassini, L., Watanabe, M., & Williamson, D. (2017). The Art and Science of Climate Model Tuning. *Bulletin of the American Meteorological Society*, *98*(3), 589–602. https://doi.org/10.1175/BAMS-D-15-00135.1

Huang, H., Xue, Y., Li, F., & Liu, Y. (2020). Modeling long-term fire impact on ecosystem characteristics and surface energy using a process-based vegetation–fire model SSiB4/TRIFFID-Fire v1.0. *Geoscientific Model Development*, *13*(12), 6029–6050. https://doi.org/10.5194/gmd-13-6029-2020

Krasnopolsky, V. M., Fox-Rabinovitz, M. S., & Belochitski, A. A. (2013). Using ensemble of neural networks to learn stochastic convection parameterizations for climate and numerical weather prediction models from data simulated by a cloud resolving model. *Advances in Artificial Neural Systems*, *2013*, 5–5. https://doi.org/10.1155/2013/485913

Lee, M.-I., Schubert, S. D., Suarez, M. J., Held, I. M., Lau, N.-C., Ploshay, J. J., Kumar, A., Kim, H.-K., & Schemm, J.-K. E. (2007). An Analysis of the Warm-Season Diurnal Cycle over the Continental United States and Northern Mexico in General Circulation Models. *Journal of Hydrometeorology*, *8*(3), 344–366. https://doi.org/10.1175/JHM581.1

O'Gorman, P. A., & Dwyer, J. G. (2018). Using machine learning to parameterize moist convection: Potential for modeling of climate, climate change, and extreme events. *Journal of Advances in Modeling Earth Systems*, *10*(10), 2548–2563. https://doi.org/10.1029/2018ms001351

Randall, D. A. (2013). Beyond deadlock. *Geophysical Research Letters*, *40*(22), 5970–5976. https://doi.org/10.1002/2013GL057998

Randall, D., Khairoutdinov, M., Arakawa, A., & Grabowski, W. (2003). Breaking the Cloud Parameterization Deadlock. *Bulletin of the American Meteorological Society*, *84*(11), 1547–1564. https://doi.org/10.1175/BAMS-84-11-1547

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

Schär, C., Fuhrer, O., Arteaga, A., Ban, N., Charpilloz, C., Girolamo, S. D., Hentgen, L., Hoefler, T., Lapillonne, X., Leutwyler, D., Osterried, K., Panosetti, D., Rüdisühli, S., Schlemmer, L., Schulthess, T. C., Sprenger, M., Ubbiali, S., & Wernli, H. (2020). Kilometer-Scale Climate Models: Prospects and Challenges. *Bulletin of the American Meteorological Society*, *101*(5), E567–E587. https://doi.org/10.1175/BAMS-D-18-0167.1

Swann, H. (2001). Evaluation of the mass-flux approach to parametrizing deep convection. *Quarterly Journal of the Royal Meteorological Society*, *127*(574), 1239–1260. https://doi.org/10.1002/qj.49712757406

Walters, D., Boutle, I., Brooks, M., Melvin, T., Stratton, R., Vosper, S., Wells, H., Williams, K., Wood, N., Allen, T., Bushell, A., Copsey, D., Earnshaw, P., Edwards, J., Gross, M., Hardiman, S., Harris, C., Heming, J., Klingaman, N., … Xavier, P. (2017). The Met Office Unified Model Global Atmosphere 6.0/6.1 and JULES Global Land 6.0/6.1 configurations. *Geoscientific Model Development*, *10*(4), 1487–1520. https://doi.org/10.5194/gmd-10-1487-2017

Wang, X., Han, Y., Xue, W., Yang, G., & Zhang, G. J. (2022). Stable climate simulations using a realistic general circulation model with neural network parameterizations for atmospheric moist physics and radiation processes. *Geoscientific Model Development*, *15*(9), 3923–3940. https://doi.org/10.5194/gmd-15-3923-2022

Webster, S., Uddstrom, M., Oliver, H., & Vosper, S. (2008). A high-resolution modelling case study of a severe weather event over New Zealand. *Atmospheric Science Letters*, *9*(3), 119–128. https://doi.org/10.1002/asl.172

Xu, K.-M., & Randall, D. A. (1996). A Semiempirical Cloudiness Parameterization for Use in Climate Models. *Journal of the Atmospheric Sciences*, *53*(21), 3084–3102. https://doi.org/10.1175/1520-0469(1996)053<3084:ASCPFU>2.0.CO;2

Zhang, T., Lin, W., Vogelmann, A. M., Zhang, M., Xie, S., Qin, Y., & Golaz, J.-C. (2021). Improving Convection Trigger Functions in Deep Convective Parameterization Schemes Using Machine Learning. *Journal of Advances in Modeling Earth Systems*, *13*(5), e2020MS002365. https://doi.org/10.1029/2020MS002365

Zhang, T., Morcrette, C., Zhang, M., Lin, W., Xie, S., Liu, Y., Weverberg, K. V., & Rodrigues, J. (2024). *tzhang-ccs/ML4ESM: ML4ESM_v1* (Version v1) [Computer software]. [object Object]. https://doi.org/10.5281/ZENODO.11005103