

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

Tao Zhang¹, Cyril Julien Morcrette², Meng Zhang³, Wuyin Lin⁴, Shaocheng Xie³, Ye Liu⁵, Kwinten Van Weverberg⁶, and Joana Rodrigues²

¹Brookhaven National Laboratory

²Met Office

³Lawrence Livermore National Laboratory

⁴Brookhaven National Laboratory (DOE)

⁵Pacific Northwest National Laboratory (DOE)

⁶Department of Geography

April 16, 2024

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.

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

Tao Zhang¹, Cyril Morcrette^{2,7}, Meng Zhang³, Wuyin Lin¹, Shaocheng Xie³, Ye Liu⁴, Kwinten Van Weverberg^{5,6}, Joana Rodrigues²

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

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.

33 Plain Language

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

43 1. Introduction

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

55
56 In recent years, machine learning (ML) techniques have emerged as a promising approach to
57 improve parameterizations in ESMs. They are capable of learning complex patterns and
58 relationships directly from observational data or high-resolution simulations, enabling the
59 capture of nonlinearities and intricate interactions that may be challenging to represent with
60 traditional parameterizations. For example, Zhang et al. (2021) proposed a ML trigger function
61 for a deep convection parameterization by learning from field observations, demonstrating its
62 superior accuracy compared to traditional CAPE-based trigger functions. Chen et al. (2023)
63 developed a neural network-based cloud fraction parameterization, better predicting both spatial

64 distribution and vertical structure of cloud fraction when compared to the traditional Xu-Randall
65 scheme (Xu & Randall, 1996). Krasnopolsky et al. (2013) prototyped a system using a neural
66 network to learn the convective temperature and moisture tendencies from cloud-resolving
67 model (CRM) simulations. These tendencies refer to the rates of change of various atmospheric
68 variables over one time step, diagnosed from particular parameterization schemes. These studies
69 lay the groundwork for integrating ML-based parameterization into ESMs.

70
71 However, the aforementioned studies primarily focus on offline ML of parameterizations that do
72 not directly interact with ESMs. Recently, there have been efforts to implement ML
73 parameterizations that can be directly coupled with ESMs. Several studies have developed ML
74 parameterizations in ESMs by hard coding custom neural network modules, such as O’Gorman
75 & Dwyer (2018), Rasp et al. (2018), Han et al. (2020) and Gettelman et al. (2021). They
76 incorporated a Fortran-based ML inference module to allow the loading of the pre-trained ML
77 weights to reconstruct the ML algorithm in ESMs. The hard-coding has limitations. Kochkov et
78 al. (2023) presented an innovative ML parameterization that feeds back from the dynamics, in
79 order to improve stability and reduce bias. However, such hard-coding approach restricts the ML
80 algorithm’s ability to adapt to changes in the model dynamics over time, as the ‘online’ updating
81 requires a two-way coupling between the dominantly Fortran-based ESMs and Python ML
82 libraries.

83
84 Fortran-Keras Bridge (FKB; Ott et al. (2020)) and C Foreign Function Interface (CFFI;
85 <https://cffi.readthedocs.io>) are two packages that support two-way coupling between Fortran-based ESM
86 and Python based ML parameterizations. FKB enables tight integration of Keras deep learning models but
87 is specifically bound to the Keras library, limiting its compatibility with other frameworks like PyTorch
88 and Scikit-Learn. On the other hand, CFFI provides a more flexible solution that in principle supports
89 coupling various ML packages due to its language-agnostic design. Brenowitz & Bretherton (2018)
90 utilized it to enable the calling of Python ML algorithms within ESMs. However, the CFFI has several
91 limitations. When utilizing CFFI to interface Fortran and Python, it uses global data structures to pass
92 variables between the two languages. This approach results in additional memory overhead as variable
93 values need to be copied between languages, instead of being passed by reference. Additionally, CFFI
94 lacks automatic garbage collection for the unused memory within these data structures and copies.
95 Consequently, the memory usage of the program gradually increases over its lifetime. In addition, when

96 using CFFI to call Python functions from a Fortran program, the process involves several steps such as
97 registering variables into a global data structure, calling the Python function, and retrieving the calculated
98 result. These multiple steps can introduce computational overhead due to the additional operations
99 required.

100

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

110

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

128 ML-based parameterizations by researchers. It enables easier code maintenance, updates, and the
129 adoption of state-of-the-art ML techniques without disrupting the existing Fortran infrastructure.
130 Ultimately, this advancement will contribute to enhanced predictions and a deeper
131 comprehension of the evolving climate of our planet.

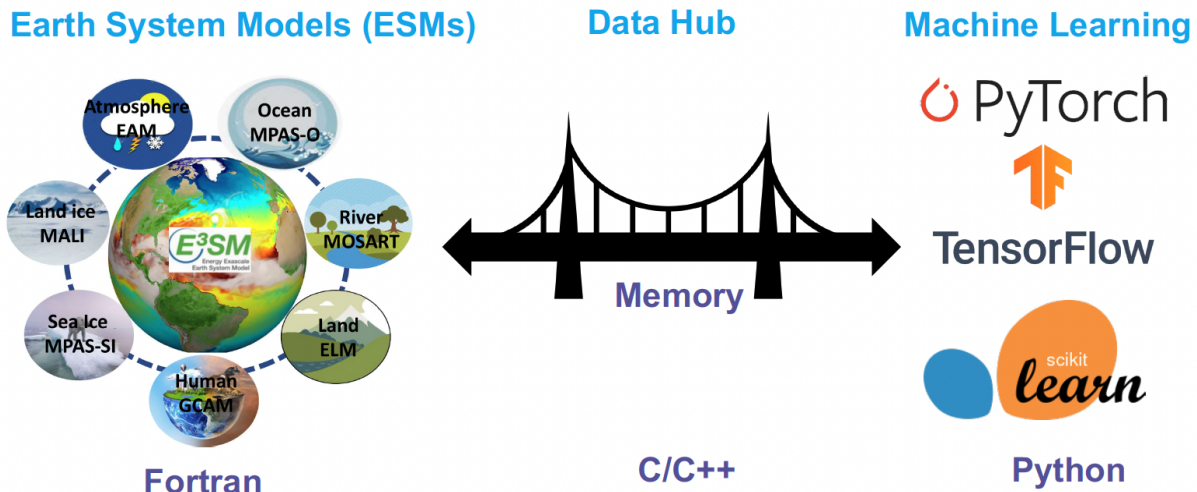
132

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

137 2. General design of the ML interface

138 2.1 Architecture of the ML interface

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



150
 151 **Figure 1.** The interface of the ML bridge for two-way communication via memory between Fortran ESM
 152 and Python ML module. The diagram for the ESMs uses E3SM as an illustration. Note that MALI and
 153 GCAM are yet active components of officially released E3SM.

154 2.2 Code structure

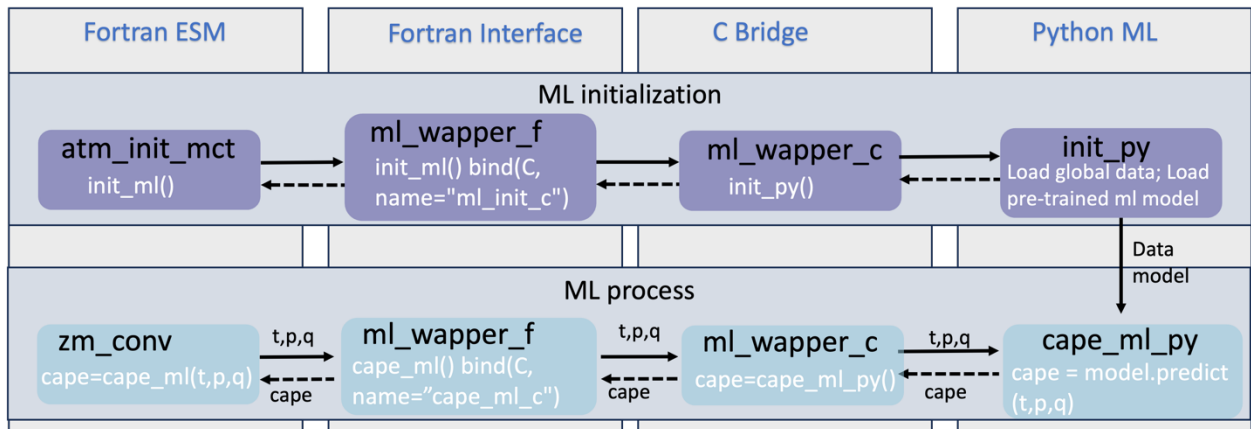
155 Figure 2 illustrates the structure of the ML bridge interface as applied to E3SM. The interface consists of
 156 four main components: the Fortran ESM, Fortran Interface, C Bridge, and Python ML. The ML functions
 157 are invoked within the original Fortran ESM parameterization components, such as the atmospheric
 158 convection and microphysics modules. This process involves transferring the required input variables to
 159 Python and defining the expected output variables to be returned to the Fortran component. The Fortran
 160 Interface and C Bridge play a crucial role in establishing the interface between Fortran and Python. They
 161 facilitate the transfer of variables between Fortran and Python by utilizing memory references. The ML
 162 function called within the Fortran ESM is defined in the Fortran interface, which is then bound to a
 163 corresponding C function. This seamless integration enables efficient communication and data exchange
 164 between the Fortran and Python components. The Python ML component is responsible for handling ML-
 165 related tasks, such as loading the trained ML algorithm and using it to make predictions. Cython is used
 166 to simplify the usage and facilitate the transfer and return of arrays. It allows for efficient integration of
 167 Python code with C libraries, enhancing performance and enabling seamless array operations within the
 168 ML component.

169
 170 The interface consists of two stages. The first stage involves initializing the ML environment, which
 171 persists throughout the model simulations. On the Fortran ESM side, the `init_ml()` function is called in the
 172 `atm_init_mct` module. Through the Fortran Interface and C Bridge, the corresponding function in the

173 Python ML component is invoked. This function loads the ML-related global data and the trained ML
 174 algorithm. This initialization process is performed only once to enhance efficiency and avoid unnecessary
 175 repetition during the simulations. The second stage involves the actual invocation of the ML process. The
 176 example here is an ML-based closure for the deep convection parameterization. We aim to utilize ML to
 177 calculate Convective Available Potential Energy (CAPE) by utilizing an ML emulator based on high-
 178 resolution cloud-resolving model simulations. We call the `cape_ml` function in the Fortran module
 179 `zm_conv`, providing temperature, pressure, and humidity as input variables, and defining the returned
 180 CAPE from the ML side. Through the Fortran Interface and C Bridge, these three variables are passed to
 181 the Python ML component. In the Python ML component, the received variables, along with other pre-
 182 loaded global data and the trained ML algorithm, are used to calculate the ML-based CAPE. The
 183 calculated result is then returned to the Fortran ESM. The Fortran ESM utilizes this ML-derived CAPE to
 184 determine how convection will evolve.

185

186



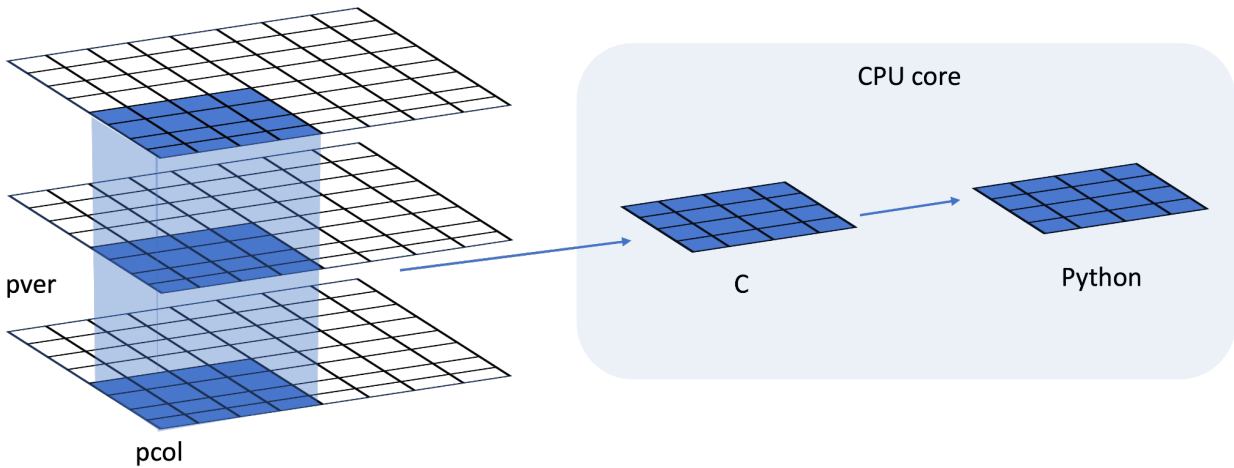
187

188 **Figure 2.** The code structure of the ML bridge interface using the ML closure in deep convection as an
 189 example.

190

191 In traditional ESMs, sub-grid scale parameterization routines such as convection parameterizations are
 192 often calculated separately for each vertical column of the model domain. Meanwhile, the domain is
 193 typically decomposed horizontally into 2D chunks that can be solved in parallel using MPI processes.
 194 Each CPU core/MPI process is assigned a number of chunks of model columns to update asynchronously
 195 (Figure 3). Our interface takes advantage of this existing parallel decomposition by designing the ML
 196 calls to operate over all columns simultaneously within each chunk, rather than invoking the ML scheme
 197 individually for each column. This allows the coupled model-ML system to leverage parallelism in the
 198 neural network computations. If the ML were called separately for every column, parallel efficiencies

199 would not be realized. By aggregating inputs over the chunk-scale prior to interfacing with Python,
 200 performance is improved through better utilization of multi-core and GPU-based ML capabilities during
 201 parameterization calculations. The Python, C, Cython and Fortran code components are compiled
 202 together into a unified executable file. Table 1 shows the detailed steps to enable the ML bridge interface
 203 in E3SM.
 204



205
 206 **Figure 3.** Data and system structure. The model domain is decomposed into chunks of columns. pver
 207 refers to number of pressure vertical levels. A chunk contains multiple columns (up to pcol). Multiple
 208 chunks can be assigned to each CPU core.
 209

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

Step	Description
1.	Create the Python environment using Conda <ul style="list-style-type: none"> ● conda create ML4ESM ● conda activate ML4ESM
2.	Add the Python ML environment in the compile CMake file
3.	Incorporate the ML bridge framework codes (including the Fortran Interface and C Bridge) into the ESM codebase.

4. Initialize of ML environment by loading necessary global data and the pre-trained ML algorithm.
5. Implement the ML prediction and the transmission of the resulting values to the ESM parameterization module.
6. Cythonize the Python code
7. Build and compile the ESM
8. Submit the job for model simulation

211

212 3. Results

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

220

221 The framework's performance is influenced by two primary factors: increasing memory usage and
222 increasing computational overhead. Firstly, maintaining the Python environment fully persistent in
223 memory throughout model simulations can impact memory usage, especially for large ML algorithms.
224 This elevated memory footprint increases the risk of leaks or crashes as simulations progress. Secondly,
225 executing ML components within the Python interpreter inevitably introduces some overhead compared
226 to the original ESMs. The increased memory requirements and decreased computational efficiency
227 associated with these considerations can impact the framework's usability, flexibility, and scalability for
228 different applications.

229

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

233 3.1 Application cases

234 3.1.1 ML trigger function in deep convection parameterization

235 Convection plays a vital role in atmospheric processes, such as precipitation formation, heat and moisture
236 transport, and energy redistribution (Arakawa, 2004; Arakawa & Schubert, 1974). However, the
237 deficiencies in convection parameterizations constitute one of the principal sources of uncertainties in
238 General Circulation Models (D. A. Randall, 2013). Some uncertainties in convection parameterizations
239 are recognized to be closely linked to the convection trigger function used in these schemes (Bechtold et
240 al., 2004; Xie et al., 2004, 2019; Xie & Zhang, 2000; Lee et al., 2007). The convective trigger in a
241 convective parameterization determines when and where model convection should be triggered as the
242 simulation advances. In many convection parameterizations, the trigger function consists of a simple,
243 arbitrary threshold for a physical quantity, such as convective available potential energy (CAPE).
244 Figure 4a illustrates how the CAPE-based trigger function works. Convection will be triggered if the
245 CAPE value exceeds a threshold value, such as 70 J/kg used in E3SM version 1.

246

247 In this work, we develop a ML trigger function and apply it to E3SM (Golaz et al., 2019, 2022). A brief
248 overview of this ML trigger function is given here, while further details will be elaborated upon in a
249 subsequent paper. The training data originates from simulations performed using the Met Office Unified
250 Model Regional Atmosphere 1.0 configuration (Bush et al., 2020). Each simulation consists of a limited
251 area model (LAM) nested within a global forecast model providing boundary conditions (Walters et al.,
252 2017; Webster et al., 2008). In total 80 LAM simulations were run located so as to sample different
253 geographical regions worldwide. Each LAM was run for 1 month, with 2-hourly output, using a grid-
254 length of 1.5 km, a 512 x 512 domain, and a model physics package used for operational weather
255 forecasting. This physics package does not include a convective parameterization scheme, but does
256 include a representation of fractional cloudiness (Bush et al., 2020). The 1.5 km data is coarse-grained to
257 several scales from 15 to 144 km, comparable to the scale a global model might be run at. At each scale,
258 we assess whether individual pixels can be considered to be buoyant cloudy updrafts (BCU, e.g.
259 Hartmann et al., 2019; Swann, 2001). Here, the threshold for buoyant is local virtual temperature more
260 than 0.1 K warmer than the average at that scale and height. Cloudy is defined whenever the fractional
261 cloud cover is greater than 0.0 and updraft is defined as vertical ascent larger than 0.2 m/s. In each

262 averaging region, the number of grid points that meet all three criteria are counted and saved as a profile
263 of BCU fraction.

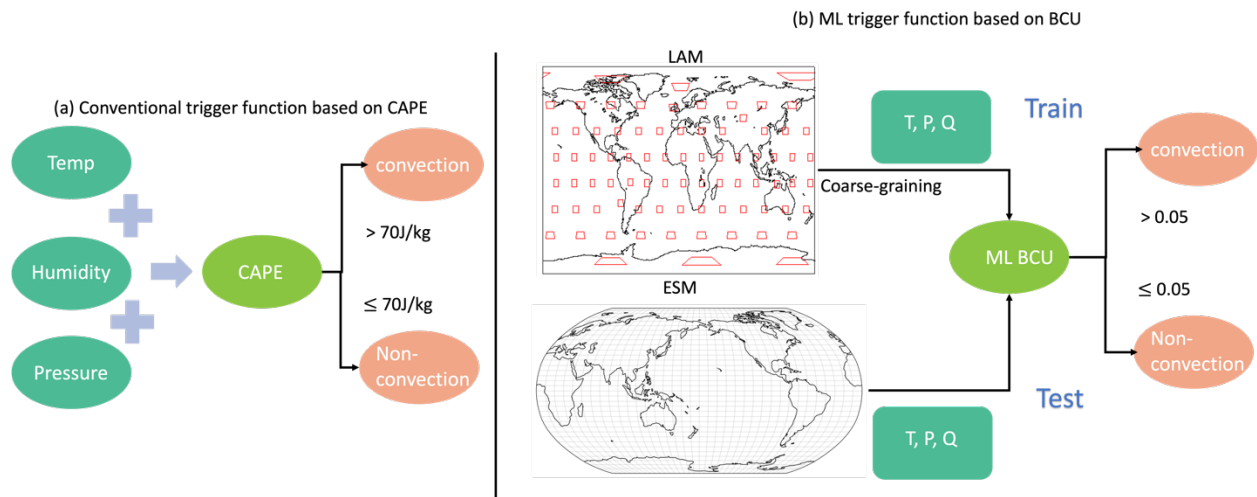
264

265 A two-stream neural network architecture is used for the ML model. The first stream takes profiles of
266 temperature, specific humidity and pressure as inputs and passes them through a 4-layer convolutional
267 neural network (CNN) with kernel sizes of 3, to extract large scale features. The second stream takes
268 mean orographic height, standard deviation of orographic height, land fraction and the size of the grid-
269 box as inputs. The outputs of the two streams are then combined and fed into a 2-layer fully connected
270 network to allow the ML model to leverage both atmospheric and surface features when making its
271 predictions. The output of the ML model is a profile of BCU.

272

273 Once trained, the CNN is coupled to E3SM and thermodynamic information from E3SM is passed to it to
274 predict the profile of BCU. If there are 3 contiguous levels where the predicted BCU is larger than 0.05,
275 the convection scheme is triggered.

276



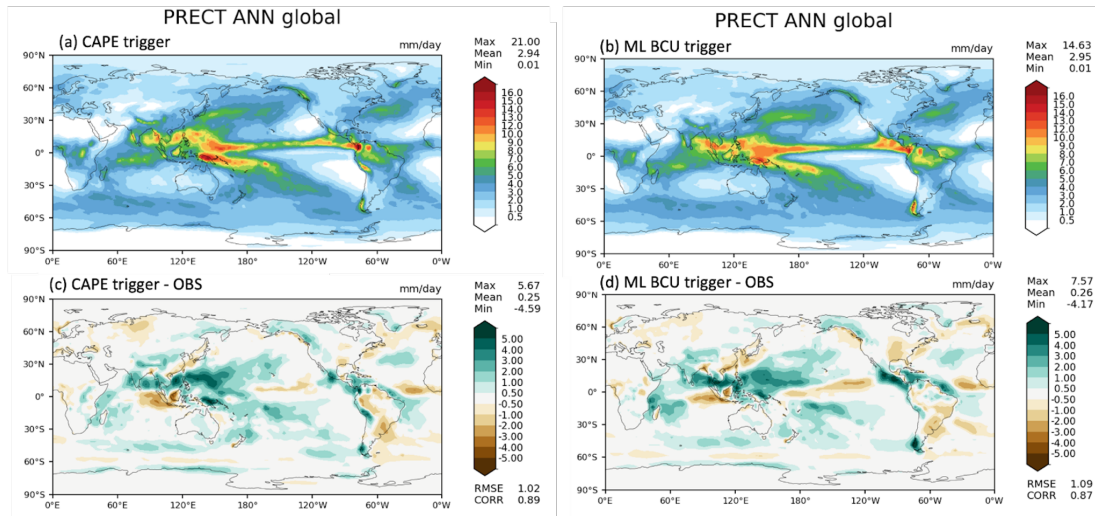
277

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

280

281 The ML trigger function is implemented using this two-stream architecture and coupled with the E3SM
282 model using the framework described in Section 2. Figure 5 shows the comparison of annual mean
283 precipitation between the control run using the CAPE-based trigger function and the run using the ML
284 BCU trigger function. The ML BCU scheme demonstrates reasonable spatial patterns of precipitation,
285 similar to the control run, with comparable root-mean-square error and spatial correlation. Additional

286 experiments exploring the definition of BCU and varying the thresholds along with an in-depth analysis
287 will be presented in a follow-up paper.
288



289
290 **Figure 5.** Comparison of annual mean precipitation between the control run using the CAPE-based
291 trigger function (a, c) and the run using the ML BCU trigger function (b, d).
292

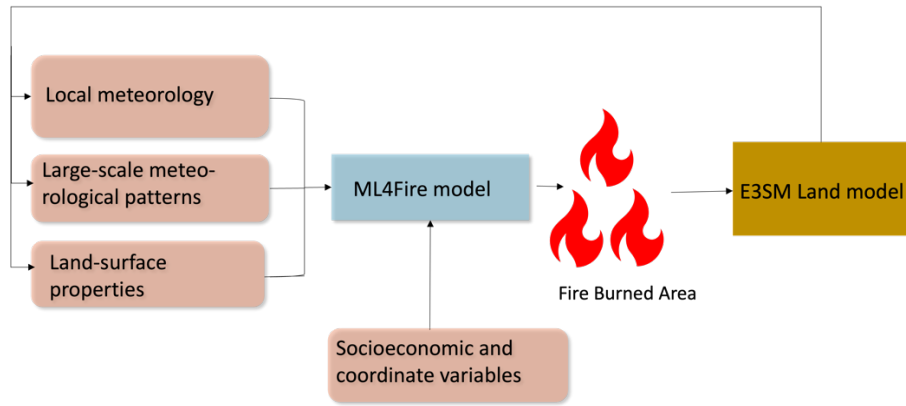
293 3.1.2 ML learning fire model

294 Wildfires in the United States have significantly increased in frequency and intensity in recent decades,
295 resulting in substantial direct and indirect losses (Iglesias et al., 2022). Predicting wildfire burned area is
296 challenging due to the complex interrelationships between fires, climate, weather, vegetation, topography,
297 and human activities (Huang et al., 2020). Traditionally, statistical methods like multiple linear regression
298 have been applied, but are limited in the number and diversity of predictors considered (Yue et al., 2013).
299 Alternatively, ML algorithms that capture statistical relationships between the burned area and
300 environmental factors have shown promising burned area prediction (Kondylatos et al., 2022; Li et al.,
301 2023; Wang et al., 2022, 2023). However, improving long-term burned area projections and evaluating
302 fire impacts requires the coupling of the fire model to an earth system model, which allows simulations of
303 the interactions between the fire, atmosphere, land cover and vegetation (Huang et al., 2021). To achieve
304 this, we develop a coupled fire-land-atmosphere framework using ML.

305

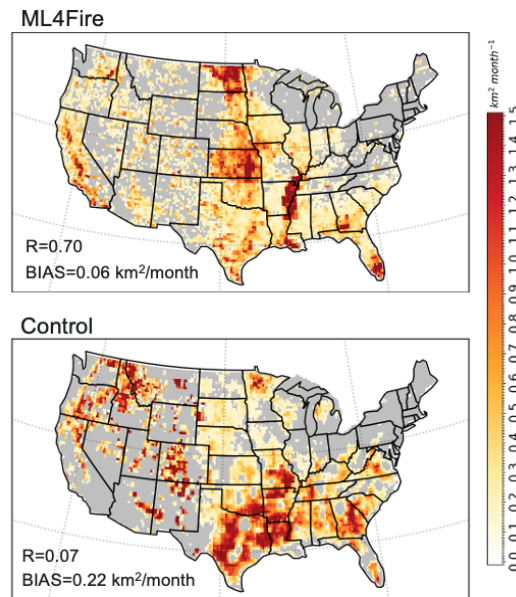
306 The ML algorithm is trained using a monthly dataset, which includes the target variable of burned area, as
307 well as various predictor variables. These predictors encompass local meteorological data (e.g., surface
308 temperature, precipitation), land surface properties (e.g., monthly mean evapotranspiration and surface

309 soil moisture), and socioeconomic variables (e.g., gross domestic product, population density), as
 310 described by Wang et al. (2022). In the coupled fire-land-atmosphere framework, meteorology variables
 311 and land surface properties are provided by the E3SM, as illustrated in Figure 6. We use the eXtreme
 312 Gradient Boosting algorithm implemented in Scikit-Learn to train the ML fire model. Figure 7
 313 demonstrates that the ML4Fire model exhibits superior performance in terms of spatial distribution
 314 compared to process-based fire models, particularly in the Southern US region. Detailed analysis will be
 315 presented in a separate paper. The ML4Fire model has proven to be a valuable tool for studying
 316 vegetation-fire interactions, enabling seamless exploration of climate-fire feedbacks.



317
 318
 319

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

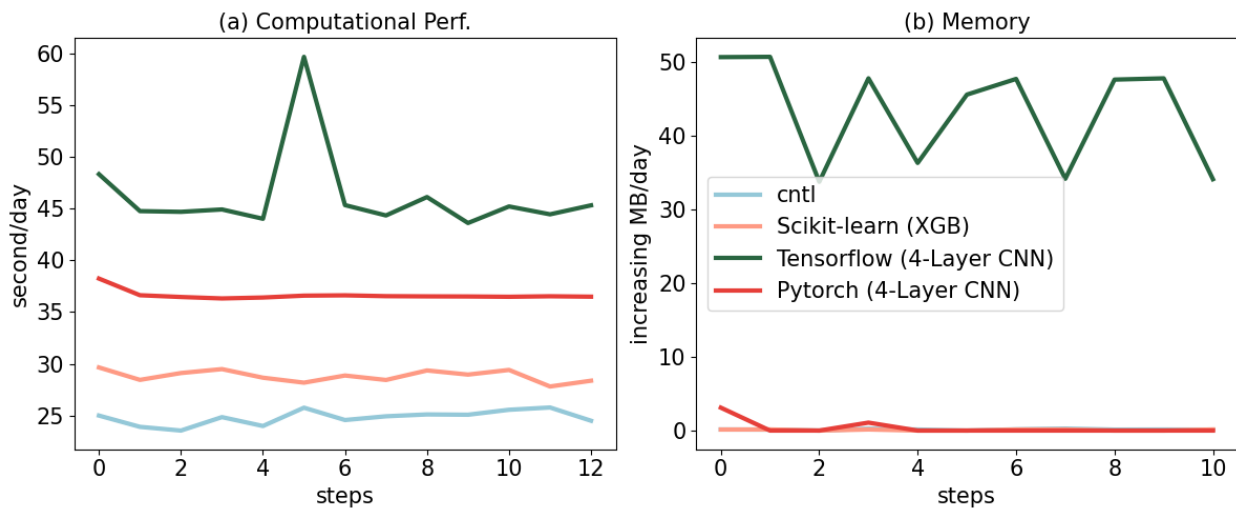


320

321 **Figure 7.** Comparison between ML4Fire model and process-based fire model against the historical
 322 burned area from Global Fire Emissions Database 5 from 2001-2020. R and BIAS are the spatial
 323 pattern correlation and difference against the observation, respectively.

324 3.2 Performance of different ML frameworks

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



338 **Figure 8.** Computational and memory overhead as the simulation progresses for coupling the ML trigger
339 function with the E3SM model. The x-axis represents the simulated time step. The y-axis of (a) represents
340 the simulation speed measured in seconds per day (indicating the number of seconds required to simulate
341 one day). The y-axis of (b) represents the relative increase in memory usage for Scikit-learn, TensorFlow,
342 and PyTorch compared with CNTL. CNTL represents the original simulation without using the ML
343 framework.
344

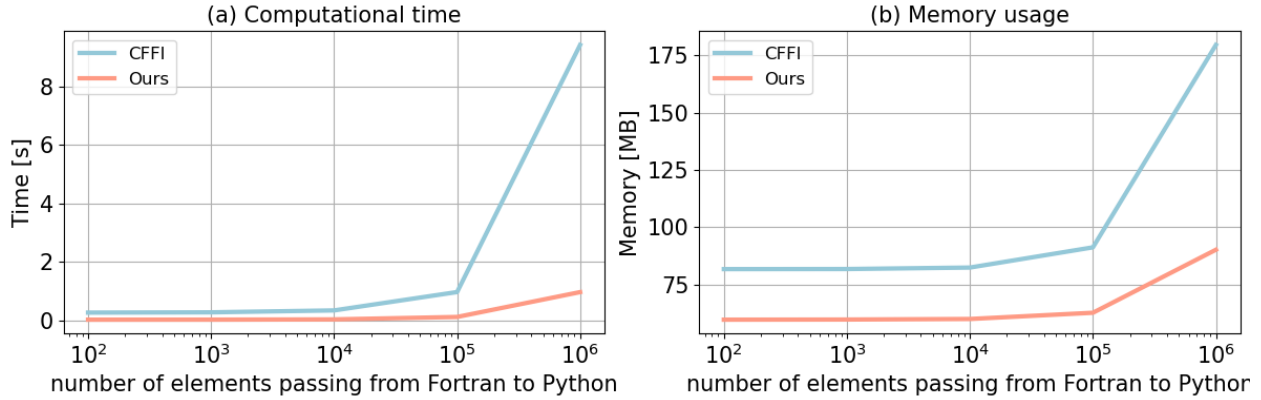
346 Figure 8 illustrates the computational and memory overhead associated with the ML parameterization
347 using different ML frameworks. It shows that XGBoost only exhibits a 20% increase in the simulation
348 time required for simulating one day due to its simpler algorithm. For more complex neural networks,

349 PyTorch incurs a 52% overhead, while TensorFlow's overhead is almost 100% – about two times as much
350 as the overhead by PyTorch. In terms of memory usage, we use the highwater memory metric (Gerber &
351 Wasserman, 2013), which represents the total memory footprint of a process. Scikit-learn and PyTorch do
352 not show any significant increase in memory usage. However, TensorFlow shows a considerable increase
353 up to 50MB per simulation day per MPI process element. This is significant because for a node with 48
354 cores, it would equate to an increase of around 2GB per simulated day on that node. This rapid memory
355 growth could quickly lead to a simulation crash due to insufficient memory during continuous
356 integrations, preventing the use in practical simulations. Our findings show that the TensorFlow
357 prediction function does not release memory after each call. Therefore, we recommend using PyTorch for
358 complex deep learning algorithms and Scikit-learn for simpler ML algorithms to avoid these potential
359 memory-related issues when using TensorFlow.

360
361 Previous work, such as Brenowitz & Bretherton (2018, 2019) has utilized the CFFI package to establish
362 communication between Fortran ESM and ML Python. As described in the Introduction, while CFFI
363 offers flexibility in supporting various ML packages, it does have certain limitations. To pass variables
364 from Fortran to Python, the approach relies on global data structures to store all variables, including both
365 the input from Fortran to Python and the output returning to Fortran. Consequently, this package results in
366 additional memory copy operations and increasing overall memory usage. In contrast, our interface takes
367 a different approach by utilizing memory references to transfer data between Fortran and Python,
368 avoiding the need for global data structures and the associated overhead. This allows for a more efficient
369 data transfer process.

370
371 In Figure 9, we present a comparison between the two frameworks by testing the different number of
372 elements passed from Fortran to Python. The evaluation is based on a demo example that focuses solely
373 on declaring arrays and transferring them from Fortran to Python, rather than a real E3SM simulation.
374 Figure 9a illustrates the impact of the number of passing elements on the overhead of the two interfaces.
375 As the number of elements exceeds 10^4 , the overhead of CFFI becomes significant. When the number
376 surpasses 10^6 , the overhead of CFFI is nearly ten times greater than that of our interface. Regarding
377 memory usage, our interface maintains a stable memory footprint of approximately 60MB. Even as the
378 number of elements increases, the memory usage only shows minimal growth. However, for CFFI, the
379 memory usage starts at 80MB, which is 33% higher than our interface. As the number of elements
380 reaches 10^6 , the memory overhead for CFFI dramatically rises to 180MB, twice as much as our interface.

381



382
 383 Figure 9. Comparison of our framework and the CFFI framework in terms of computational time
 384 and memory usage. The x-axis represents the number of elements transferred from Fortran to
 385 Python, while the y-axis displays the total time (a) and total memory usage (b) for a
 386 demonstration example. The evaluations presented are based on the average results obtained
 387 from 5 separate tests.

388

389 3.3 Performance of ML algorithms of different complexities

390 ML parameterizations can be implemented using various deep learning algorithms with different levels of
 391 complexity. The computational performance and memory usage can be influenced by the complexity of
 392 these algorithms. In the case of the ML trigger function, a two-stream four-layer CNN structure is
 393 employed. We compare this structure with other ML algorithms such as Artificial Neural Network (ANN)
 394 and Residual Network (ResNet), whose structures are detailed in Table 2. These algorithms are
 395 implemented in PyTorch. The algorithm’s complexity is measured by the number of parameters, with the
 396 CNN having approximately 60 times more parameters than ANN, and ResNet having roughly 1.5 times
 397 more parameters than CNN.

398

399 **Table 2.** The structure and number of parameters of each ML algorithms.

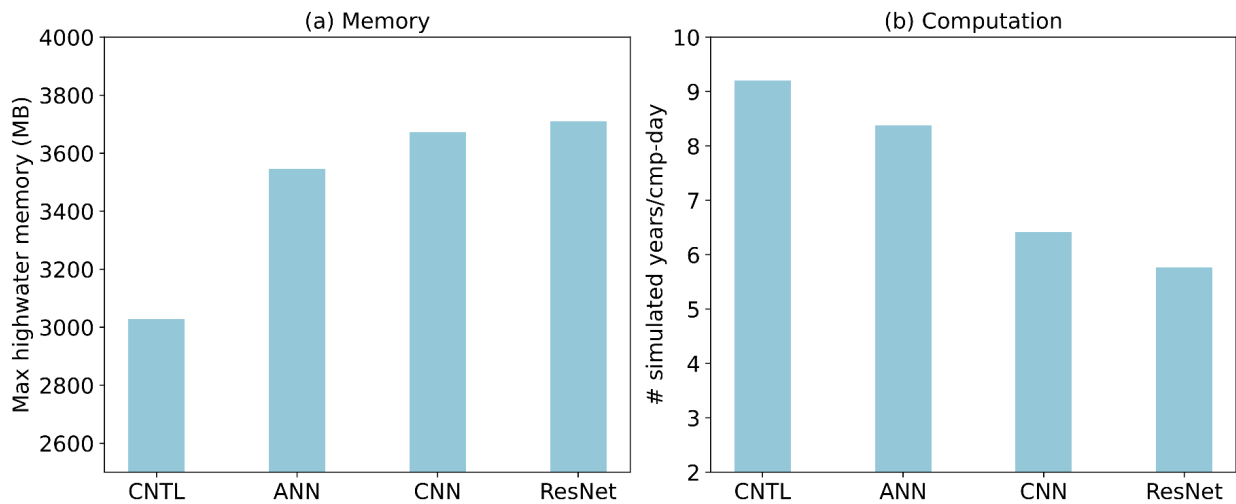
Algorithms	Structure	# of parameters
ANN	3 x Linear	121,601
CNN	4 x Conv2d + 2 x Linear	7,466,753
ResNet	17 x Conv2d + 1 x Linear	11,177,025

400

401 Figure 10 presents a comparison of the memory and computational costs between the CNTL run without
402 deep learning parameterization and various deep learning algorithms. A same specific process-element
403 layout (placement of ESM component models on distributed CPU cores) is used for all the simulations.
404 Deep learning algorithms incur a significant yet affordable increase in memory overhead, with at least a
405 20% increase compared to the CNTL run (Figure 10a). This is primarily due to the integration of ML
406 algorithms into the ESM, which persist throughout the simulations. Although there is a notable increase in
407 complexity among the deep learning algorithms, their memory usage only shows a slight rise. This is
408 because the memory increment resulting from the ML parameters is relatively small. Specifically, ANN
409 requires 1MB of memory, CNN requires 60MB, and the ResNet algorithms requires 85MB, which are
410 calculated based on the number of parameters in each algorithm. When comparing these values to the
411 memory consumption of the CNTL run, which is approximately 3000MB, the additional parameters'
412 incremental memory consumption is not substantial.

413

414 However, there is a significant decrease in computational performance as the complexity of the deep
415 learning algorithms increases (Figure 10b). This is primarily due to the larger number of parameters in
416 neural networks, which require more forward computations. It is worth noting that in this study, the deep
417 learning algorithms are executed on CPUs. To enhance computational performance, future work could
418 consider utilizing GPUs for acceleration.



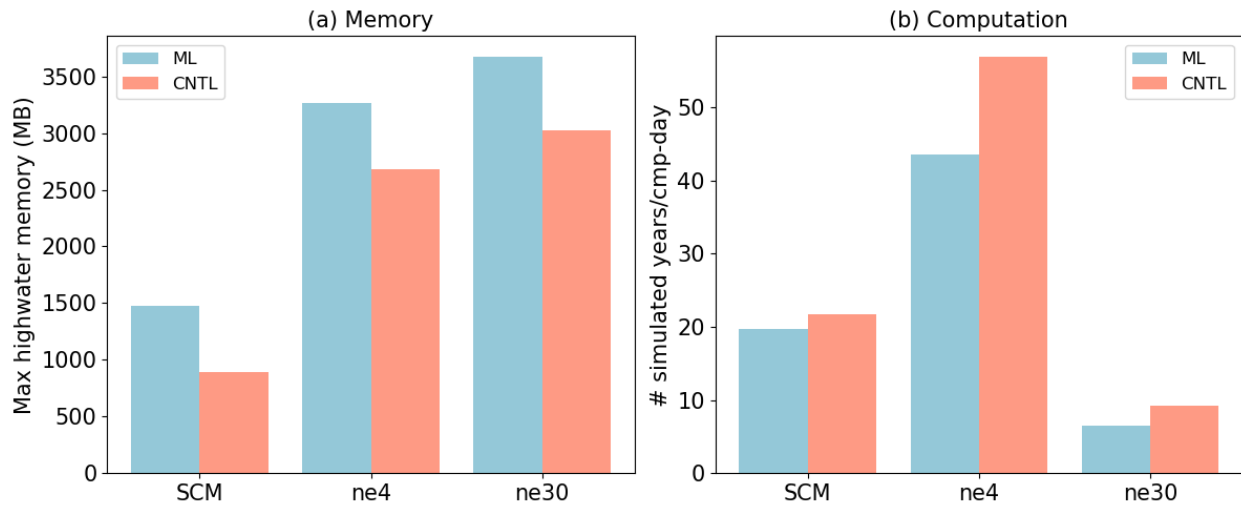
419

420 **Figure 10.** Comparison of CNTL and various ML algorithms in terms of memory and computation.

421 CNTL is the default run without ML parameterizations.

422

423 3.4 Performance for physical models of different complexities



424
 425 **Figure 11.** Compassion CNTL and ML for various ESMs in terms of memory and computation. The
 426 ESM configuration include SCM, ultra-low resolution model (ne4) and nominal low-resolution model
 427 (ne30).

428
 429 ML parameterization can be applied to various ESM configurations, for example, with the E3SM
 430 Atmosphere Model (EAM), we experiment with Single Column Model (SCM), the ultra low-resolution
 431 model of EAM (ne4), and the nominal low resolution model of EAM (ne30) configurations. The SCM
 432 consists of one single atmosphere column of a global EAM (Bogenschutz et al., 2020; Gettelman et al.,
 433 2019). ne4 has 384 columns, with each column representing the horizontal resolution of 7.5°. ne30 is the
 434 default resolution for EAM and comprises 21,600 columns, with each column representing the horizontal
 435 resolution of 1°. In the case of the ML trigger function, the memory overhead is approximately 500MB
 436 for all configurations due to the loading of the ML algorithm, which does not vary with the configuration
 437 of the ESM.

438
 439 Regarding computational performance, SCM utilizes 1 process, ne4 employs 1 node with 64 processes,
 440 and ne30 utilizes 10 nodes with each node using 128 processes. In the case of SCM, the overhead
 441 attributed to the ML parameterization is approximately 9% due to the utilization of only 1 process.
 442 However, for ne4 and ne30, the overhead is 23% and 28% respectively (Figure 11). The increasing
 443 computational overhead is primarily due to resource competition when multiple processes are used within
 444 a single node.

445

446 4. Discussion and Conclusion

447 In this study, we develop a novel Fortran-Python interface for developing ML parameterizations. ML
448 algorithm can learn detailed information about cloud processes and atmospheric dynamics from
449 kilometer-scale models and observations and serves as an approximate surrogate for the kilometer-scale
450 model. Instead of explicitly simulating kilometer-scale processes, the ML algorithms can be designed to
451 capture the essential features and relationships between atmospheric variables by training on available
452 kilometer-scale data. The trained algorithms can then be used to develop parameterizations for use in
453 models at coarser resolutions, reducing the computational and memory costs. By using ML
454 parameterizations, scientists can effectively incorporate the insights gained from kilometer-scale models
455 for coarser-resolution simulations. Through learning the complex relationships and patterns present in the
456 high-resolution data, the ML-based parameterizations have the potentials to more accurately represent
457 cloud processes and atmospheric dynamics in the ESMs. This approach strikes a balance between
458 computational efficiency and capturing critical processes, enabling more realistic simulations and
459 predictions while minimizing computational resources. All these potential benefits in turn promote
460 innovative developments to facilitate increasing and more efficient use of ML parameterizations.

461
462 In this study, we develop a novel Fortran-Python interface for developing ML parameterizations. This
463 interface demonstrates feasibility in supporting various ML frameworks, such as PyTorch, TensorFlow,
464 and Scikit-learn and enables the effective development of new ML-based parameterizations to explore
465 ML-based applications in ESMs. Through two cases - a ML trigger function in convection
466 parameterization and a ML wildfire model - we highlight high modularity and reusability of the
467 framework. We conduct a systematic evaluation of memory usage and computational overhead from the
468 integrated Python codes.

469
470 Based on our performance evaluation, we observe that coupling ML algorithms using TensorFlow into
471 ESMs can lead to memory leaks. As a recommendation, we suggest using PyTorch for complex deep
472 learning algorithms and Scikit-learn for simple ML algorithms for the Fortran-Python ML interface.

473
474 The memory overhead primarily arises from loading ML algorithms into ESMs. If the ML algorithms are
475 implemented using PyTorch or Scikit-learn, the memory usage will not increase significantly. The
476 computational overhead is influenced by the complexity of the neural network and the number of
477 processes running on a single node. As the complexity of the neural network increases, more parameters

478 in the neural network require gradient computation. Similarly, when there are more processes running on
479 a single node, the integrated Python codes introduces more resource competition.

480

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

488 Acknowledge

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

497

498 Conflict of Interest

499 The authors declare that they have no conflict of interest.

500

501 Data Availability Statement

502 The Fortran-Python interface for developing ML parameterizations can be archived at
503 <https://github.com/tzhang-ccs/ML4ESM>. The E3SM model can be accessed at [https://github.com/E3SM-
504 Project/E3SM](https://github.com/E3SM-Project/E3SM).

505 References

- 506 Bechtold, P., Chaboureaud, J.-P., Beljaars, A., Betts, A. K., Köhler, M., Miller, M., & Redelsperger, J.-L.
507 (2004). The simulation of the diurnal cycle of convective precipitation over land in a global
508 model. *Quarterly Journal of the Royal Meteorological Society*, *130*(604), 3119–3137.
509 <https://doi.org/10.1256/qj.03.103>
- 510 Bogenschutz, P. A., Tang, S., Caldwell, P. M., Xie, S., Lin, W., & Chen, Y.-S. (2020). The E3SM version
511 1 single-column model. *Geoscientific Model Development*, *13*(9), 4443–4458.
512 <https://doi.org/10.5194/gmd-13-4443-2020>
- 513 Brenowitz, N. D., & Bretherton, C. S. (2018). Prognostic validation of a neural network unified physics
514 parameterization. *Geophysical Research Letters*, *45*(12), 6289–6298.
515 <https://doi.org/10.1029/2018gl078510>
- 516 Brenowitz, N. D., & Bretherton, C. S. (2019). Spatially extended tests of a neural network
517 parametrization trained by coarse-graining. *Journal of Advances in Modeling Earth Systems*,
518 *11*(8), 2728–2744. <https://doi.org/10.1029/2019ms001711>
- 519 Bush, M., Allen, T., Bain, C., Boutle, I., Edwards, J., Finnenkoetter, A., Franklin, C., Hanley, K., Lean,
520 H., Lock, A., Manners, J., Mittermaier, M., Morcrette, C., North, R., Petch, J., Short, C., Vosper,
521 S., Walters, D., Webster, S., ... Zerroukat, M. (2020). The first Met Office Unified Model–
522 JULES Regional Atmosphere and Land configuration, RAL1. *Geoscientific Model Development*,
523 *13*(4), 1999–2029. <https://doi.org/10.5194/gmd-13-1999-2020>
- 524 Chen, G., Wang, W., Yang, S., Wang, Y., Zhang, F., & Wu, K. (2023). A Neural Network-Based Scale-
525 Adaptive Cloud-Fraction Scheme for GCMs. *Journal of Advances in Modeling Earth Systems*,
526 *15*(6), e2022MS003415. <https://doi.org/10.1029/2022MS003415>
- 527 Covey, C., Gleckler, P. J., Doutriaux, C., Williams, D. N., Dai, A., Fasullo, J., Trenberth, K., & Berg, A.
528 (2016). Metrics for the Diurnal Cycle of Precipitation: Toward Routine Benchmarks for Climate
529 Models. *Journal of Climate*, *29*(12), 4461–4471. <https://doi.org/10.1175/JCLI-D-15-0664.1>

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

533 Gettelman, A., Gagne, D. J., Chen, C.-C., Christensen, M. W., Lebo, Z. J., Morrison, H., & Gantos, G.
534 (2021). Machine Learning the Warm Rain Process. *Journal of Advances in Modeling Earth*
535 *Systems*, 13(2), e2020MS002268. <https://doi.org/10.1029/2020MS002268>

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

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

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

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

555 Hartmann, D. L., Blossey, P. N., & Dygert, B. D. (2019). Convection and Climate: What Have We
556 Learned from Simple Models and Simplified Settings? *Current Climate Change Reports*, 5(3),
557 196–206. <https://doi.org/10.1007/s40641-019-00136-9>

558 Hourdin, F., Mauritsen, T., Gettelman, A., Golaz, J.-C., Balaji, V., Duan, Q., Folini, D., Ji, D., Klocke,
559 D., Qian, Y., Rauser, F., Rio, C., Tomassini, L., Watanabe, M., & Williamson, D. (2017). The Art
560 and Science of Climate Model Tuning. *Bulletin of the American Meteorological Society*, 98(3),
561 589–602. <https://doi.org/10.1175/BAMS-D-15-00135.1>

562 Huang, H., Xue, Y., Li, F., & Liu, Y. (2020). Modeling long-term fire impact on ecosystem
563 characteristics and surface energy using a process-based vegetation–fire model SSiB4/TRIFFID-
564 Fire v1.0. *Geoscientific Model Development*, 13(12), 6029–6050. [https://doi.org/10.5194/gmd-13-](https://doi.org/10.5194/gmd-13-6029-2020)
565 [6029-2020](https://doi.org/10.5194/gmd-13-6029-2020)

566 Huang, H., Xue, Y., Liu, Y., Li, F., & Okin, G. S. (2021). Modeling the short-term fire effects on
567 vegetation dynamics and surface energy in southern Africa using the improved SSiB4/TRIFFID-
568 Fire model. *Geoscientific Model Development*, 14(12), 7639–7657. [https://doi.org/10.5194/gmd-](https://doi.org/10.5194/gmd-14-7639-2021)
569 [14-7639-2021](https://doi.org/10.5194/gmd-14-7639-2021)

570 Kochkov, D., Yuval, J., Langmore, I., Norgaard, P., Smith, J., Mooers, G., Lottes, J., Rasp, S., Düben, P.,
571 Klöwer, M., Hatfield, S., Battaglia, P., Sanchez-Gonzalez, A., Willson, M., Brenner, M. P., &
572 Hoyer, S. (2023). *Neural General Circulation Models* (arXiv:2311.07222). arXiv.
573 <http://arxiv.org/abs/2311.07222>

574 Kondylatos, S., Prapas, I., Ronco, M., Papoutsis, I., Camps-Valls, G., Piles, M., Fernández-Torres, M.-Á.,
575 & Carvalhais, N. (2022). Wildfire Danger Prediction and Understanding With Deep Learning.
576 *Geophysical Research Letters*, 49(17), e2022GL099368. <https://doi.org/10.1029/2022GL099368>

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

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

585 Li, F., Zhu, Q., Riley, W. J., Zhao, L., Xu, L., Yuan, K., Chen, M., Wu, H., Gui, Z., Gong, J., &
586 Randerson, J. T. (2023). AttentionFire_v1.0: Interpretable machine learning fire model for
587 burned-area predictions over tropics. *Geoscientific Model Development*, 16(3), 869–884.
588 <https://doi.org/10.5194/gmd-16-869-2023>

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

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

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

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

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

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

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

613 Wang, S. S.-C., Leung, L. R., & Qian, Y. (2023). Projection of Future Fire Emissions Over the
614 Contiguous US Using Explainable Artificial Intelligence and CMIP6 Models. *Journal of*
615 *Geophysical Research: Atmospheres*, 128(14), e2023JD039154.
616 <https://doi.org/10.1029/2023JD039154>

617 Wang, S. S.-C., Qian, Y., Leung, L. R., & Zhang, Y. (2022). Interpreting machine learning prediction of
618 fire emissions and comparison with FireMIP process-based models. *Atmospheric Chemistry and*
619 *Physics*, 22(5), 3445–3468. <https://doi.org/10.5194/acp-22-3445-2022>

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

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

627 Xu, K.-M., & Randall, D. A. (1996). A Semiempirical Cloudiness Parameterization for Use in Climate
628 Models. *Journal of the Atmospheric Sciences*, 53(21), 3084–3102. [https://doi.org/10.1175/1520-](https://doi.org/10.1175/1520-0469(1996)053<3084:ASCPFU>2.0.CO;2)
629 [0469\(1996\)053<3084:ASCPFU>2.0.CO;2](https://doi.org/10.1175/1520-0469(1996)053<3084:ASCPFU>2.0.CO;2)

630 Zhang, T., Lin, W., Vogelmann, A. M., Zhang, M., Xie, S., Qin, Y., & Golaz, J.-C. (2021). Improving
631 Convection Trigger Functions in Deep Convective Parameterization Schemes Using Machine
632 Learning. *Journal of Advances in Modeling Earth Systems*, 13(5), e2020MS002365.
633 <https://doi.org/10.1029/2020MS002365>
634

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

Tao Zhang¹, Cyril Morcrette^{2,7}, Meng Zhang³, Wuyin Lin¹, Shaocheng Xie³, Ye Liu⁴, Kwinten Van Weverberg^{5,6}, Joana Rodrigues²

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

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.

33 Plain Language

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

43 1. Introduction

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

55
56 In recent years, machine learning (ML) techniques have emerged as a promising approach to
57 improve parameterizations in ESMs. They are capable of learning complex patterns and
58 relationships directly from observational data or high-resolution simulations, enabling the
59 capture of nonlinearities and intricate interactions that may be challenging to represent with
60 traditional parameterizations. For example, Zhang et al. (2021) proposed a ML trigger function
61 for a deep convection parameterization by learning from field observations, demonstrating its
62 superior accuracy compared to traditional CAPE-based trigger functions. Chen et al. (2023)
63 developed a neural network-based cloud fraction parameterization, better predicting both spatial

64 distribution and vertical structure of cloud fraction when compared to the traditional Xu-Randall
65 scheme (Xu & Randall, 1996). Krasnopolsky et al. (2013) prototyped a system using a neural
66 network to learn the convective temperature and moisture tendencies from cloud-resolving
67 model (CRM) simulations. These tendencies refer to the rates of change of various atmospheric
68 variables over one time step, diagnosed from particular parameterization schemes. These studies
69 lay the groundwork for integrating ML-based parameterization into ESMs.

70
71 However, the aforementioned studies primarily focus on offline ML of parameterizations that do
72 not directly interact with ESMs. Recently, there have been efforts to implement ML
73 parameterizations that can be directly coupled with ESMs. Several studies have developed ML
74 parameterizations in ESMs by hard coding custom neural network modules, such as O’Gorman
75 & Dwyer (2018), Rasp et al. (2018), Han et al. (2020) and Gettelman et al. (2021). They
76 incorporated a Fortran-based ML inference module to allow the loading of the pre-trained ML
77 weights to reconstruct the ML algorithm in ESMs. The hard-coding has limitations. Kochkov et
78 al. (2023) presented an innovative ML parameterization that feeds back from the dynamics, in
79 order to improve stability and reduce bias. However, such hard-coding approach restricts the ML
80 algorithm’s ability to adapt to changes in the model dynamics over time, as the ‘online’ updating
81 requires a two-way coupling between the dominantly Fortran-based ESMs and Python ML
82 libraries.

83
84 Fortran-Keras Bridge (FKB; Ott et al. (2020)) and C Foreign Function Interface (CFFI;
85 <https://cffi.readthedocs.io>) are two packages that support two-way coupling between Fortran-based ESM
86 and Python based ML parameterizations. FKB enables tight integration of Keras deep learning models but
87 is specifically bound to the Keras library, limiting its compatibility with other frameworks like PyTorch
88 and Scikit-Learn. On the other hand, CFFI provides a more flexible solution that in principle supports
89 coupling various ML packages due to its language-agnostic design. Brenowitz & Bretherton (2018)
90 utilized it to enable the calling of Python ML algorithms within ESMs. However, the CFFI has several
91 limitations. When utilizing CFFI to interface Fortran and Python, it uses global data structures to pass
92 variables between the two languages. This approach results in additional memory overhead as variable
93 values need to be copied between languages, instead of being passed by reference. Additionally, CFFI
94 lacks automatic garbage collection for the unused memory within these data structures and copies.
95 Consequently, the memory usage of the program gradually increases over its lifetime. In addition, when

96 using CFFI to call Python functions from a Fortran program, the process involves several steps such as
97 registering variables into a global data structure, calling the Python function, and retrieving the calculated
98 result. These multiple steps can introduce computational overhead due to the additional operations
99 required.

100

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

110

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

128 ML-based parameterizations by researchers. It enables easier code maintenance, updates, and the
129 adoption of state-of-the-art ML techniques without disrupting the existing Fortran infrastructure.
130 Ultimately, this advancement will contribute to enhanced predictions and a deeper
131 comprehension of the evolving climate of our planet.

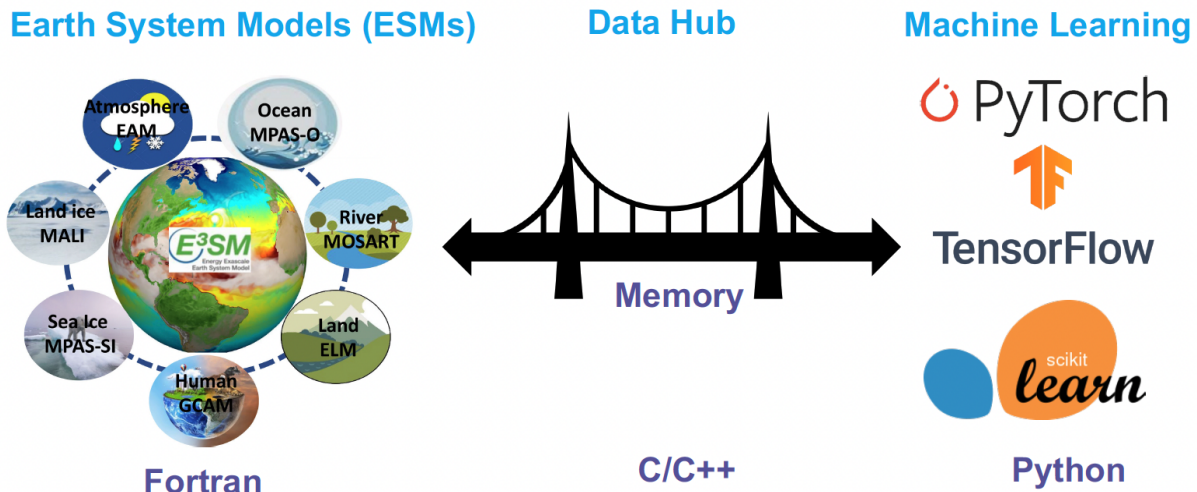
132

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

137 2. General design of the ML interface

138 2.1 Architecture of the ML interface

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



150
 151 **Figure 1.** The interface of the ML bridge for two-way communication via memory between Fortran ESM
 152 and Python ML module. The diagram for the ESMs uses E3SM as an illustration. Note that MALI and
 153 GCAM are yet active components of officially released E3SM.

154 2.2 Code structure

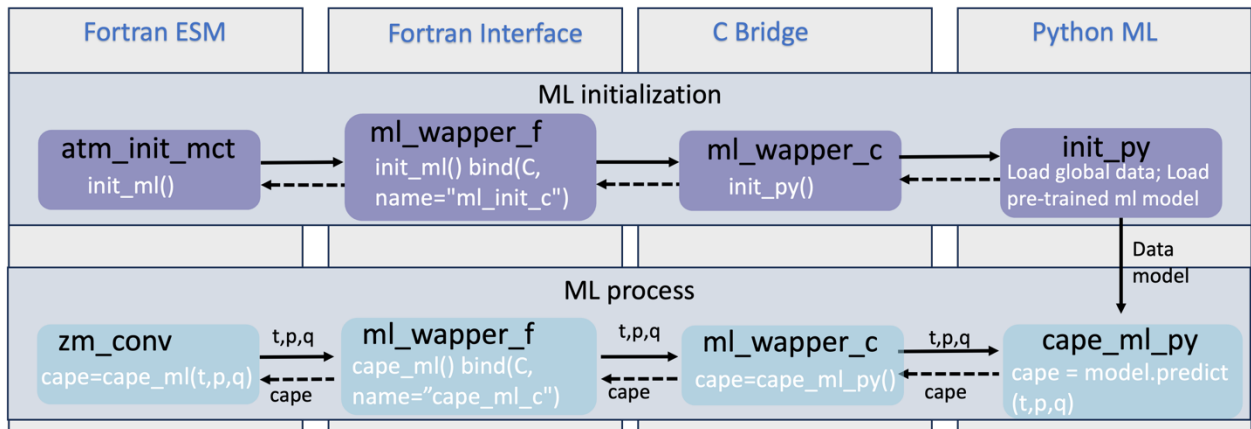
155 Figure 2 illustrates the structure of the ML bridge interface as applied to E3SM. The interface consists of
 156 four main components: the Fortran ESM, Fortran Interface, C Bridge, and Python ML. The ML functions
 157 are invoked within the original Fortran ESM parameterization components, such as the atmospheric
 158 convection and microphysics modules. This process involves transferring the required input variables to
 159 Python and defining the expected output variables to be returned to the Fortran component. The Fortran
 160 Interface and C Bridge play a crucial role in establishing the interface between Fortran and Python. They
 161 facilitate the transfer of variables between Fortran and Python by utilizing memory references. The ML
 162 function called within the Fortran ESM is defined in the Fortran interface, which is then bound to a
 163 corresponding C function. This seamless integration enables efficient communication and data exchange
 164 between the Fortran and Python components. The Python ML component is responsible for handling ML-
 165 related tasks, such as loading the trained ML algorithm and using it to make predictions. Cython is used
 166 to simplify the usage and facilitate the transfer and return of arrays. It allows for efficient integration of
 167 Python code with C libraries, enhancing performance and enabling seamless array operations within the
 168 ML component.

169
 170 The interface consists of two stages. The first stage involves initializing the ML environment, which
 171 persists throughout the model simulations. On the Fortran ESM side, the `init_ml()` function is called in the
 172 `atm_init_mct` module. Through the Fortran Interface and C Bridge, the corresponding function in the

173 Python ML component is invoked. This function loads the ML-related global data and the trained ML
 174 algorithm. This initialization process is performed only once to enhance efficiency and avoid unnecessary
 175 repetition during the simulations. The second stage involves the actual invocation of the ML process. The
 176 example here is an ML-based closure for the deep convection parameterization. We aim to utilize ML to
 177 calculate Convective Available Potential Energy (CAPE) by utilizing an ML emulator based on high-
 178 resolution cloud-resolving model simulations. We call the `cape_ml` function in the Fortran module
 179 `zm_conv`, providing temperature, pressure, and humidity as input variables, and defining the returned
 180 CAPE from the ML side. Through the Fortran Interface and C Bridge, these three variables are passed to
 181 the Python ML component. In the Python ML component, the received variables, along with other pre-
 182 loaded global data and the trained ML algorithm, are used to calculate the ML-based CAPE. The
 183 calculated result is then returned to the Fortran ESM. The Fortran ESM utilizes this ML-derived CAPE to
 184 determine how convection will evolve.

185

186



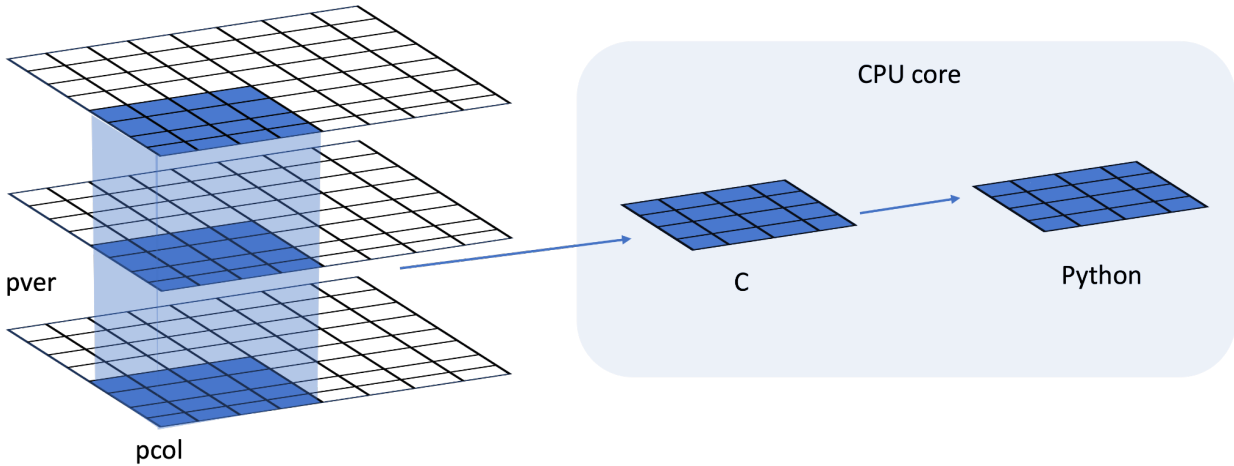
187

188 **Figure 2.** The code structure of the ML bridge interface using the ML closure in deep convection as an
 189 example.

190

191 In traditional ESMs, sub-grid scale parameterization routines such as convection parameterizations are
 192 often calculated separately for each vertical column of the model domain. Meanwhile, the domain is
 193 typically decomposed horizontally into 2D chunks that can be solved in parallel using MPI processes.
 194 Each CPU core/MPI process is assigned a number of chunks of model columns to update asynchronously
 195 (Figure 3). Our interface takes advantage of this existing parallel decomposition by designing the ML
 196 calls to operate over all columns simultaneously within each chunk, rather than invoking the ML scheme
 197 individually for each column. This allows the coupled model-ML system to leverage parallelism in the
 198 neural network computations. If the ML were called separately for every column, parallel efficiencies

199 would not be realized. By aggregating inputs over the chunk-scale prior to interfacing with Python,
 200 performance is improved through better utilization of multi-core and GPU-based ML capabilities during
 201 parameterization calculations. The Python, C, Cython and Fortran code components are compiled
 202 together into a unified executable file. Table 1 shows the detailed steps to enable the ML bridge interface
 203 in E3SM.
 204



205
 206 **Figure 3.** Data and system structure. The model domain is decomposed into chunks of columns. pver
 207 refers to number of pressure vertical levels. A chunk contains multiple columns (up to pcol). Multiple
 208 chunks can be assigned to each CPU core.
 209

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

Step	Description
1.	Create the Python environment using Conda <ul style="list-style-type: none"> ● conda create ML4ESM ● conda activate ML4ESM
2.	Add the Python ML environment in the compile CMake file
3.	Incorporate the ML bridge framework codes (including the Fortran Interface and C Bridge) into the ESM codebase.

4. Initialize of ML environment by loading necessary global data and the pre-trained ML algorithm.
5. Implement the ML prediction and the transmission of the resulting values to the ESM parameterization module.
6. Cythonize the Python code
7. Build and compile the ESM
8. Submit the job for model simulation

211

212 3. Results

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

220

221 The framework's performance is influenced by two primary factors: increasing memory usage and
222 increasing computational overhead. Firstly, maintaining the Python environment fully persistent in
223 memory throughout model simulations can impact memory usage, especially for large ML algorithms.
224 This elevated memory footprint increases the risk of leaks or crashes as simulations progress. Secondly,
225 executing ML components within the Python interpreter inevitably introduces some overhead compared
226 to the original ESMs. The increased memory requirements and decreased computational efficiency
227 associated with these considerations can impact the framework's usability, flexibility, and scalability for
228 different applications.

229

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

233 3.1 Application cases

234 3.1.1 ML trigger function in deep convection parameterization

235 Convection plays a vital role in atmospheric processes, such as precipitation formation, heat and moisture
236 transport, and energy redistribution (Arakawa, 2004; Arakawa & Schubert, 1974). However, the
237 deficiencies in convection parameterizations constitute one of the principal sources of uncertainties in
238 General Circulation Models (D. A. Randall, 2013). Some uncertainties in convection parameterizations
239 are recognized to be closely linked to the convection trigger function used in these schemes (Bechtold et
240 al., 2004; Xie et al., 2004, 2019; Xie & Zhang, 2000; Lee et al., 2007). The convective trigger in a
241 convective parameterization determines when and where model convection should be triggered as the
242 simulation advances. In many convection parameterizations, the trigger function consists of a simple,
243 arbitrary threshold for a physical quantity, such as convective available potential energy (CAPE).
244 Figure 4a illustrates how the CAPE-based trigger function works. Convection will be triggered if the
245 CAPE value exceeds a threshold value, such as 70 J/kg used in E3SM version 1.

246

247 In this work, we develop a ML trigger function and apply it to E3SM (Golaz et al., 2019, 2022). A brief
248 overview of this ML trigger function is given here, while further details will be elaborated upon in a
249 subsequent paper. The training data originates from simulations performed using the Met Office Unified
250 Model Regional Atmosphere 1.0 configuration (Bush et al., 2020). Each simulation consists of a limited
251 area model (LAM) nested within a global forecast model providing boundary conditions (Walters et al.,
252 2017; Webster et al., 2008). In total 80 LAM simulations were run located so as to sample different
253 geographical regions worldwide. Each LAM was run for 1 month, with 2-hourly output, using a grid-
254 length of 1.5 km, a 512 x 512 domain, and a model physics package used for operational weather
255 forecasting. This physics package does not include a convective parameterization scheme, but does
256 include a representation of fractional cloudiness (Bush et al., 2020). The 1.5 km data is coarse-grained to
257 several scales from 15 to 144 km, comparable to the scale a global model might be run at. At each scale,
258 we assess whether individual pixels can be considered to be buoyant cloudy updrafts (BCU, e.g.
259 Hartmann et al., 2019; Swann, 2001). Here, the threshold for buoyant is local virtual temperature more
260 than 0.1 K warmer than the average at that scale and height. Cloudy is defined whenever the fractional
261 cloud cover is greater than 0.0 and updraft is defined as vertical ascent larger than 0.2 m/s. In each

262 averaging region, the number of grid points that meet all three criteria are counted and saved as a profile
263 of BCU fraction.

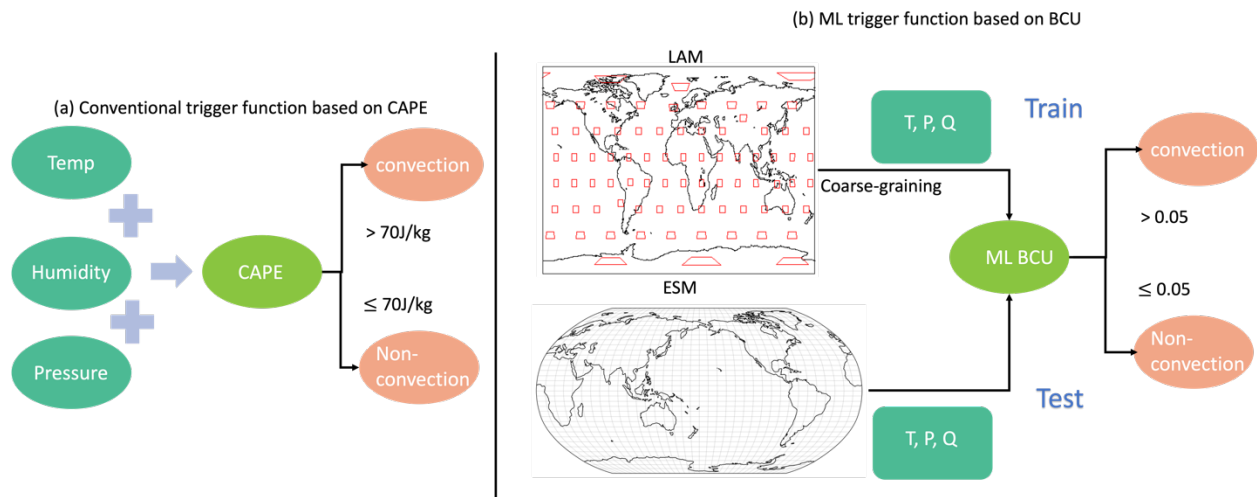
264

265 A two-stream neural network architecture is used for the ML model. The first stream takes profiles of
266 temperature, specific humidity and pressure as inputs and passes them through a 4-layer convolutional
267 neural network (CNN) with kernel sizes of 3, to extract large scale features. The second stream takes
268 mean orographic height, standard deviation of orographic height, land fraction and the size of the grid-
269 box as inputs. The outputs of the two streams are then combined and fed into a 2-layer fully connected
270 network to allow the ML model to leverage both atmospheric and surface features when making its
271 predictions. The output of the ML model is a profile of BCU.

272

273 Once trained, the CNN is coupled to E3SM and thermodynamic information from E3SM is passed to it to
274 predict the profile of BCU. If there are 3 contiguous levels where the predicted BCU is larger than 0.05,
275 the convection scheme is triggered.

276



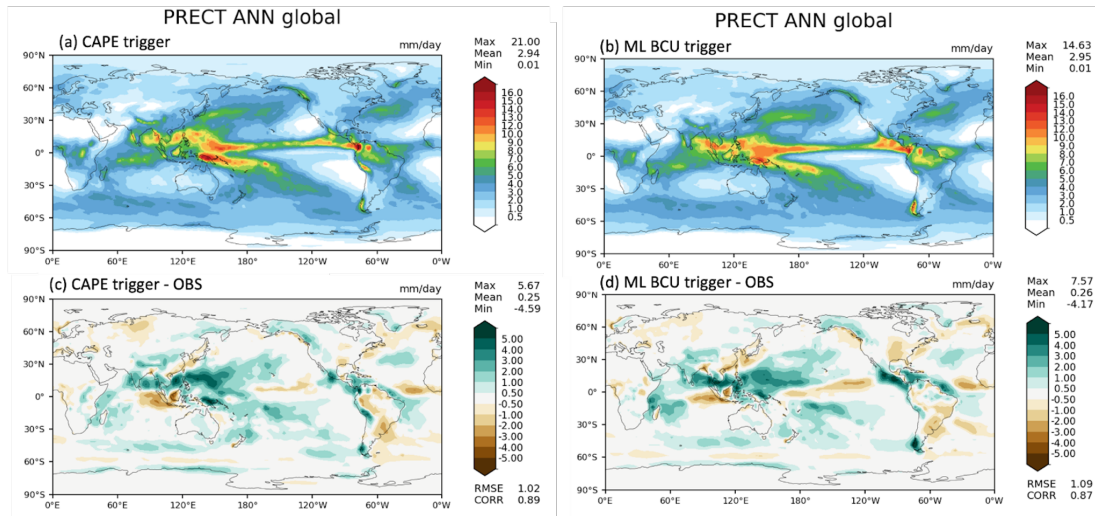
277

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

280

281 The ML trigger function is implemented using this two-stream architecture and coupled with the E3SM
282 model using the framework described in Section 2. Figure 5 shows the comparison of annual mean
283 precipitation between the control run using the CAPE-based trigger function and the run using the ML
284 BCU trigger function. The ML BCU scheme demonstrates reasonable spatial patterns of precipitation,
285 similar to the control run, with comparable root-mean-square error and spatial correlation. Additional

286 experiments exploring the definition of BCU and varying the thresholds along with an in-depth analysis
287 will be presented in a follow-up paper.
288



289
290 **Figure 5.** Comparison of annual mean precipitation between the control run using the CAPE-based
291 trigger function (a, c) and the run using the ML BCU trigger function (b, d).
292

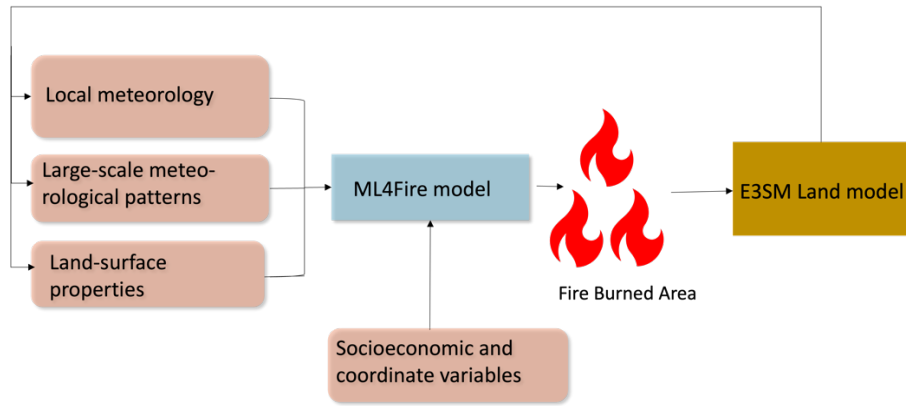
293 3.1.2 ML learning fire model

294 Wildfires in the United States have significantly increased in frequency and intensity in recent decades,
295 resulting in substantial direct and indirect losses (Iglesias et al., 2022). Predicting wildfire burned area is
296 challenging due to the complex interrelationships between fires, climate, weather, vegetation, topography,
297 and human activities (Huang et al., 2020). Traditionally, statistical methods like multiple linear regression
298 have been applied, but are limited in the number and diversity of predictors considered (Yue et al., 2013).
299 Alternatively, ML algorithms that capture statistical relationships between the burned area and
300 environmental factors have shown promising burned area prediction (Kondylatos et al., 2022; Li et al.,
301 2023; Wang et al., 2022, 2023). However, improving long-term burned area projections and evaluating
302 fire impacts requires the coupling of the fire model to an earth system model, which allows simulations of
303 the interactions between the fire, atmosphere, land cover and vegetation (Huang et al., 2021). To achieve
304 this, we develop a coupled fire-land-atmosphere framework using ML.

305

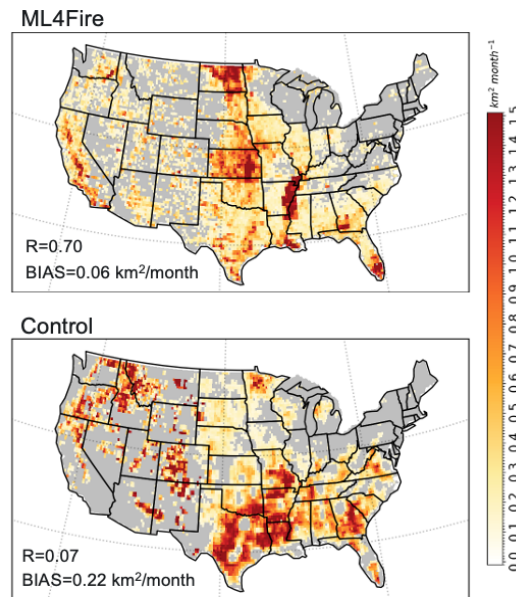
306 The ML algorithm is trained using a monthly dataset, which includes the target variable of burned area, as
307 well as various predictor variables. These predictors encompass local meteorological data (e.g., surface
308 temperature, precipitation), land surface properties (e.g., monthly mean evapotranspiration and surface

309 soil moisture), and socioeconomic variables (e.g., gross domestic product, population density), as
 310 described by Wang et al. (2022). In the coupled fire-land-atmosphere framework, meteorology variables
 311 and land surface properties are provided by the E3SM, as illustrated in Figure 6. We use the eXtreme
 312 Gradient Boosting algorithm implemented in Scikit-Learn to train the ML fire model. Figure 7
 313 demonstrates that the ML4Fire model exhibits superior performance in terms of spatial distribution
 314 compared to process-based fire models, particularly in the Southern US region. Detailed analysis will be
 315 presented in a separate paper. The ML4Fire model has proven to be a valuable tool for studying
 316 vegetation-fire interactions, enabling seamless exploration of climate-fire feedbacks.



317
 318
 319

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

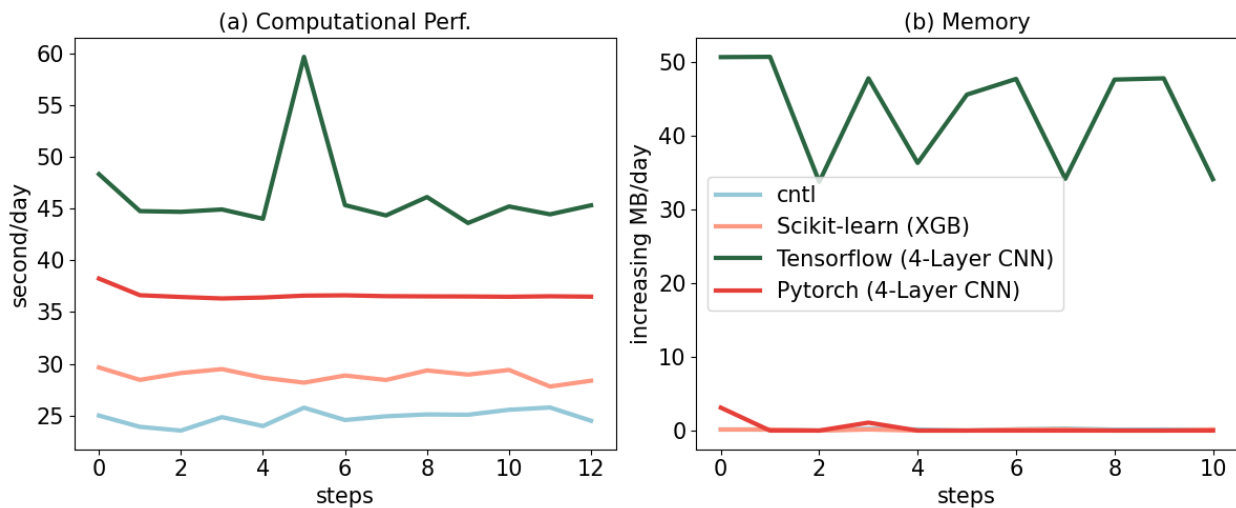


320

321 **Figure 7.** Comparison between ML4Fire model and process-based fire model against the historical
 322 burned area from Global Fire Emissions Database 5 from 2001-2020. R and BIAS are the spatial
 323 pattern correlation and difference against the observation, respectively.

324 3.2 Performance of different ML frameworks

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



338 **Figure 8.** Computational and memory overhead as the simulation progresses for coupling the ML trigger
339 function with the E3SM model. The x-axis represents the simulated time step. The y-axis of (a) represents
340 the simulation speed measured in seconds per day (indicating the number of seconds required to simulate
341 one day). The y-axis of (b) represents the relative increase in memory usage for Scikit-learn, TensorFlow,
342 and PyTorch compared with CNTL. CNTL represents the original simulation without using the ML
343 framework.
344

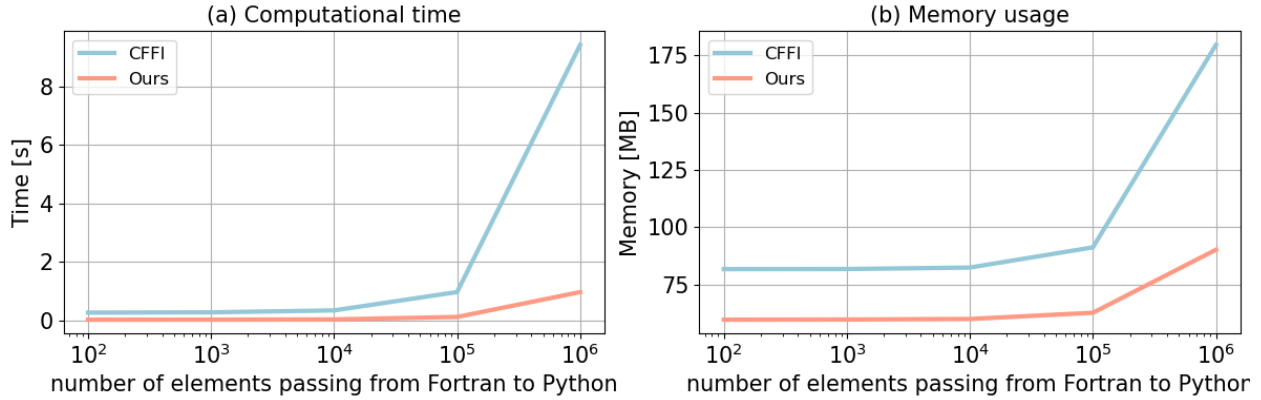
345
346 Figure 8 illustrates the computational and memory overhead associated with the ML parameterization
347 using different ML frameworks. It shows that XGBoost only exhibits a 20% increase in the simulation
348 time required for simulating one day due to its simpler algorithm. For more complex neural networks,

349 PyTorch incurs a 52% overhead, while TensorFlow's overhead is almost 100% – about two times as much
350 as the overhead by PyTorch. In terms of memory usage, we use the highwater memory metric (Gerber &
351 Wasserman, 2013), which represents the total memory footprint of a process. Scikit-learn and PyTorch do
352 not show any significant increase in memory usage. However, TensorFlow shows a considerable increase
353 up to 50MB per simulation day per MPI process element. This is significant because for a node with 48
354 cores, it would equate to an increase of around 2GB per simulated day on that node. This rapid memory
355 growth could quickly lead to a simulation crash due to insufficient memory during continuous
356 integrations, preventing the use in practical simulations. Our findings show that the TensorFlow
357 prediction function does not release memory after each call. Therefore, we recommend using PyTorch for
358 complex deep learning algorithms and Scikit-learn for simpler ML algorithms to avoid these potential
359 memory-related issues when using TensorFlow.

360
361 Previous work, such as Brenowitz & Bretherton (2018, 2019) has utilized the CFFI package to establish
362 communication between Fortran ESM and ML Python. As described in the Introduction, while CFFI
363 offers flexibility in supporting various ML packages, it does have certain limitations. To pass variables
364 from Fortran to Python, the approach relies on global data structures to store all variables, including both
365 the input from Fortran to Python and the output returning to Fortran. Consequently, this package results in
366 additional memory copy operations and increasing overall memory usage. In contrast, our interface takes
367 a different approach by utilizing memory references to transfer data between Fortran and Python,
368 avoiding the need for global data structures and the associated overhead. This allows for a more efficient
369 data transfer process.

370
371 In Figure 9, we present a comparison between the two frameworks by testing the different number of
372 elements passed from Fortran to Python. The evaluation is based on a demo example that focuses solely
373 on declaring arrays and transferring them from Fortran to Python, rather than a real E3SM simulation.
374 Figure 9a illustrates the impact of the number of passing elements on the overhead of the two interfaces.
375 As the number of elements exceeds 10^4 , the overhead of CFFI becomes significant. When the number
376 surpasses 10^6 , the overhead of CFFI is nearly ten times greater than that of our interface. Regarding
377 memory usage, our interface maintains a stable memory footprint of approximately 60MB. Even as the
378 number of elements increases, the memory usage only shows minimal growth. However, for CFFI, the
379 memory usage starts at 80MB, which is 33% higher than our interface. As the number of elements
380 reaches 10^6 , the memory overhead for CFFI dramatically rises to 180MB, twice as much as our interface.

381



382
 383 Figure 9. Comparison of our framework and the CFFI framework in terms of computational time
 384 and memory usage. The x-axis represents the number of elements transferred from Fortran to
 385 Python, while the y-axis displays the total time (a) and total memory usage (b) for a
 386 demonstration example. The evaluations presented are based on the average results obtained
 387 from 5 separate tests.

388

389 3.3 Performance of ML algorithms of different complexities

390 ML parameterizations can be implemented using various deep learning algorithms with different levels of
 391 complexity. The computational performance and memory usage can be influenced by the complexity of
 392 these algorithms. In the case of the ML trigger function, a two-stream four-layer CNN structure is
 393 employed. We compare this structure with other ML algorithms such as Artificial Neural Network (ANN)
 394 and Residual Network (ResNet), whose structures are detailed in Table 2. These algorithms are
 395 implemented in PyTorch. The algorithm’s complexity is measured by the number of parameters, with the
 396 CNN having approximately 60 times more parameters than ANN, and ResNet having roughly 1.5 times
 397 more parameters than CNN.

398

399 **Table 2.** The structure and number of parameters of each ML algorithms.

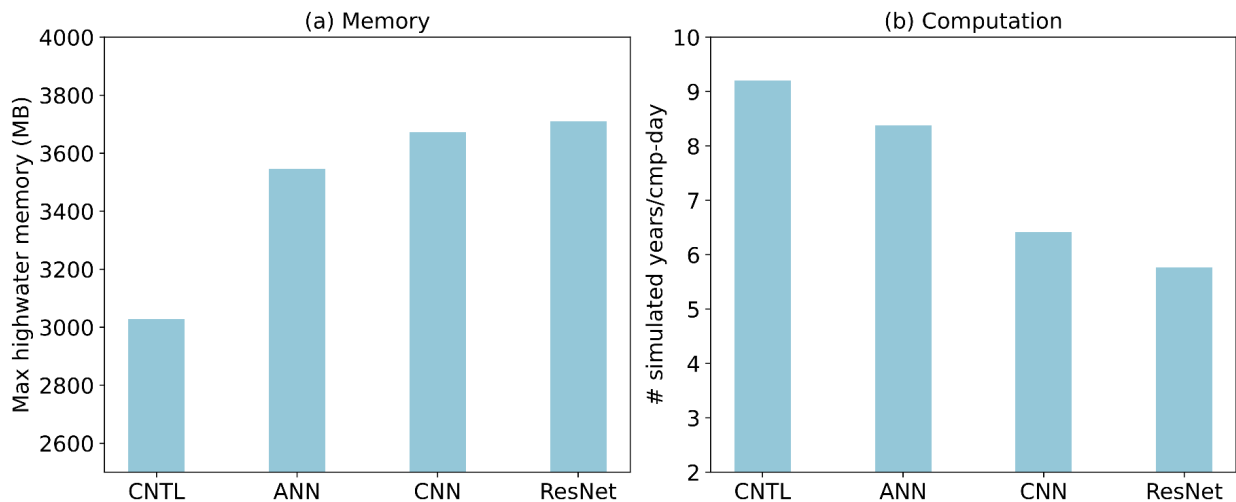
Algorithms	Structure	# of parameters
ANN	3 x Linear	121,601
CNN	4 x Conv2d + 2 x Linear	7,466,753
ResNet	17 x Conv2d + 1 x Linear	11,177,025

400

401 Figure 10 presents a comparison of the memory and computational costs between the CNTL run without
402 deep learning parameterization and various deep learning algorithms. A same specific process-element
403 layout (placement of ESM component models on distributed CPU cores) is used for all the simulations.
404 Deep learning algorithms incur a significant yet affordable increase in memory overhead, with at least a
405 20% increase compared to the CNTL run (Figure 10a). This is primarily due to the integration of ML
406 algorithms into the ESM, which persist throughout the simulations. Although there is a notable increase in
407 complexity among the deep learning algorithms, their memory usage only shows a slight rise. This is
408 because the memory increment resulting from the ML parameters is relatively small. Specifically, ANN
409 requires 1MB of memory, CNN requires 60MB, and the ResNet algorithms requires 85MB, which are
410 calculated based on the number of parameters in each algorithm. When comparing these values to the
411 memory consumption of the CNTL run, which is approximately 3000MB, the additional parameters'
412 incremental memory consumption is not substantial.

413

414 However, there is a significant decrease in computational performance as the complexity of the deep
415 learning algorithms increases (Figure 10b). This is primarily due to the larger number of parameters in
416 neural networks, which require more forward computations. It is worth noting that in this study, the deep
417 learning algorithms are executed on CPUs. To enhance computational performance, future work could
418 consider utilizing GPUs for acceleration.



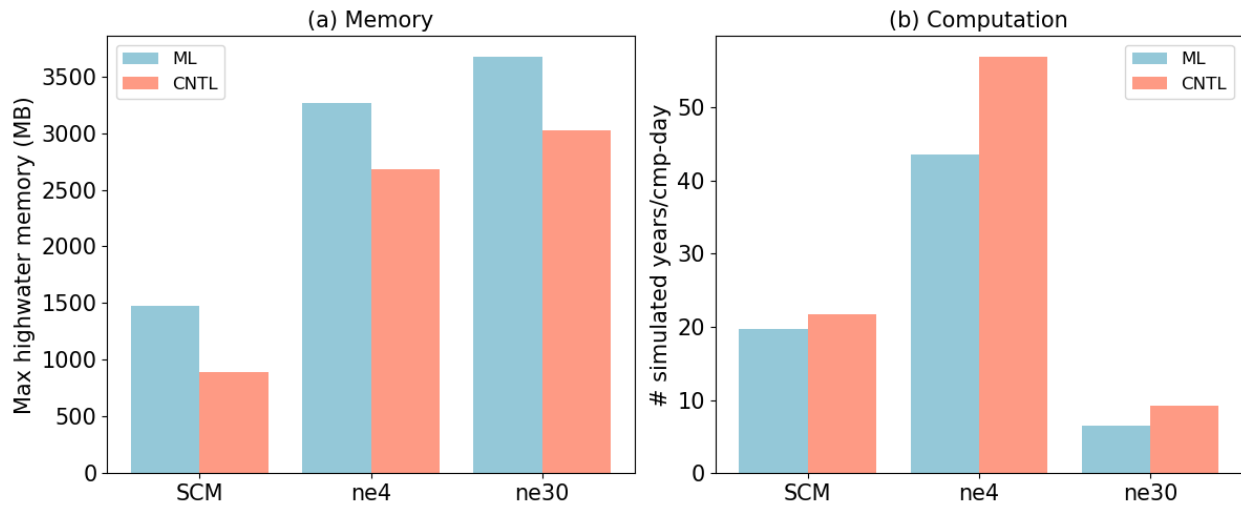
419

420 **Figure 10.** Comparison of CNTL and various ML algorithms in terms of memory and computation.

421 CNTL is the default run without ML parameterizations.

422

423 3.4 Performance for physical models of different complexities



424
 425 **Figure 11.** Compassion CNTL and ML for various ESMs in terms of memory and computation. The
 426 ESM configuration include SCM, ultra-low resolution model (ne4) and nominal low-resolution model
 427 (ne30).

428
 429 ML parameterization can be applied to various ESM configurations, for example, with the E3SM
 430 Atmosphere Model (EAM), we experiment with Single Column Model (SCM), the ultra low-resolution
 431 model of EAM (ne4), and the nominal low resolution model of EAM (ne30) configurations. The SCM
 432 consists of one single atmosphere column of a global EAM (Bogenschutz et al., 2020; Gettelman et al.,
 433 2019). ne4 has 384 columns, with each column representing the horizontal resolution of 7.5°. ne30 is the
 434 default resolution for EAM and comprises 21,600 columns, with each column representing the horizontal
 435 resolution of 1°. In the case of the ML trigger function, the memory overhead is approximately 500MB
 436 for all configurations due to the loading of the ML algorithm, which does not vary with the configuration
 437 of the ESM.

438
 439 Regarding computational performance, SCM utilizes 1 process, ne4 employs 1 node with 64 processes,
 440 and ne30 utilizes 10 nodes with each node using 128 processes. In the case of SCM, the overhead
 441 attributed to the ML parameterization is approximately 9% due to the utilization of only 1 process.
 442 However, for ne4 and ne30, the overhead is 23% and 28% respectively (Figure 11). The increasing
 443 computational overhead is primarily due to resource competition when multiple processes are used within
 444 a single node.

445

446 4. Discussion and Conclusion

447 In this study, we develop a novel Fortran-Python interface for developing ML parameterizations. ML
448 algorithm can learn detailed information about cloud processes and atmospheric dynamics from
449 kilometer-scale models and observations and serves as an approximate surrogate for the kilometer-scale
450 model. Instead of explicitly simulating kilometer-scale processes, the ML algorithms can be designed to
451 capture the essential features and relationships between atmospheric variables by training on available
452 kilometer-scale data. The trained algorithms can then be used to develop parameterizations for use in
453 models at coarser resolutions, reducing the computational and memory costs. By using ML
454 parameterizations, scientists can effectively incorporate the insights gained from kilometer-scale models
455 for coarser-resolution simulations. Through learning the complex relationships and patterns present in the
456 high-resolution data, the ML-based parameterizations have the potentials to more accurately represent
457 cloud processes and atmospheric dynamics in the ESMs. This approach strikes a balance between
458 computational efficiency and capturing critical processes, enabling more realistic simulations and
459 predictions while minimizing computational resources. All these potential benefits in turn promote
460 innovative developments to facilitate increasing and more efficient use of ML parameterizations.

461
462 In this study, we develop a novel Fortran-Python interface for developing ML parameterizations. This
463 interface demonstrates feasibility in supporting various ML frameworks, such as PyTorch, TensorFlow,
464 and Scikit-learn and enables the effective development of new ML-based parameterizations to explore
465 ML-based applications in ESMs. Through two cases - a ML trigger function in convection
466 parameterization and a ML wildfire model - we highlight high modularity and reusability of the
467 framework. We conduct a systematic evaluation of memory usage and computational overhead from the
468 integrated Python codes.

469
470 Based on our performance evaluation, we observe that coupling ML algorithms using TensorFlow into
471 ESMs can lead to memory leaks. As a recommendation, we suggest using PyTorch for complex deep
472 learning algorithms and Scikit-learn for simple ML algorithms for the Fortran-Python ML interface.

473
474 The memory overhead primarily arises from loading ML algorithms into ESMs. If the ML algorithms are
475 implemented using PyTorch or Scikit-learn, the memory usage will not increase significantly. The
476 computational overhead is influenced by the complexity of the neural network and the number of
477 processes running on a single node. As the complexity of the neural network increases, more parameters

478 in the neural network require gradient computation. Similarly, when there are more processes running on
479 a single node, the integrated Python codes introduces more resource competition.

480

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

488 Acknowledge

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

497

498 Conflict of Interest

499 The authors declare that they have no conflict of interest.

500

501 Data Availability Statement

502 The Fortran-Python interface for developing ML parameterizations can be archived at
503 <https://github.com/tzhang-ccs/ML4ESM>. The E3SM model can be accessed at [https://github.com/E3SM-
504 Project/E3SM](https://github.com/E3SM-Project/E3SM).

505 References

- 506 Bechtold, P., Chaboureaud, J.-P., Beljaars, A., Betts, A. K., Köhler, M., Miller, M., & Redelsperger, J.-L.
507 (2004). The simulation of the diurnal cycle of convective precipitation over land in a global
508 model. *Quarterly Journal of the Royal Meteorological Society*, *130*(604), 3119–3137.
509 <https://doi.org/10.1256/qj.03.103>
- 510 Bogenschutz, P. A., Tang, S., Caldwell, P. M., Xie, S., Lin, W., & Chen, Y.-S. (2020). The E3SM version
511 1 single-column model. *Geoscientific Model Development*, *13*(9), 4443–4458.
512 <https://doi.org/10.5194/gmd-13-4443-2020>
- 513 Brenowitz, N. D., & Bretherton, C. S. (2018). Prognostic validation of a neural network unified physics
514 parameterization. *Geophysical Research Letters*, *45*(12), 6289–6298.
515 <https://doi.org/10.1029/2018gl078510>
- 516 Brenowitz, N. D., & Bretherton, C. S. (2019). Spatially extended tests of a neural network
517 parametrization trained by coarse-graining. *Journal of Advances in Modeling Earth Systems*,
518 *11*(8), 2728–2744. <https://doi.org/10.1029/2019ms001711>
- 519 Bush, M., Allen, T., Bain, C., Boutle, I., Edwards, J., Finkenkoetter, A., Franklin, C., Hanley, K., Lean,
520 H., Lock, A., Manners, J., Mittermaier, M., Morcrette, C., North, R., Petch, J., Short, C., Vosper,
521 S., Walters, D., Webster, S., ... Zerroukat, M. (2020). The first Met Office Unified Model–
522 JULES Regional Atmosphere and Land configuration, RAL1. *Geoscientific Model Development*,
523 *13*(4), 1999–2029. <https://doi.org/10.5194/gmd-13-1999-2020>
- 524 Chen, G., Wang, W., Yang, S., Wang, Y., Zhang, F., & Wu, K. (2023). A Neural Network-Based Scale-
525 Adaptive Cloud-Fraction Scheme for GCMs. *Journal of Advances in Modeling Earth Systems*,
526 *15*(6), e2022MS003415. <https://doi.org/10.1029/2022MS003415>
- 527 Covey, C., Gleckler, P. J., Doutriaux, C., Williams, D. N., Dai, A., Fasullo, J., Trenberth, K., & Berg, A.
528 (2016). Metrics for the Diurnal Cycle of Precipitation: Toward Routine Benchmarks for Climate
529 Models. *Journal of Climate*, *29*(12), 4461–4471. <https://doi.org/10.1175/JCLI-D-15-0664.1>

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

533 Gettelman, A., Gagne, D. J., Chen, C.-C., Christensen, M. W., Lebo, Z. J., Morrison, H., & Gantos, G.
534 (2021). Machine Learning the Warm Rain Process. *Journal of Advances in Modeling Earth*
535 *Systems*, 13(2), e2020MS002268. <https://doi.org/10.1029/2020MS002268>

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

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

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

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

555 Hartmann, D. L., Blossey, P. N., & Dygert, B. D. (2019). Convection and Climate: What Have We
556 Learned from Simple Models and Simplified Settings? *Current Climate Change Reports*, 5(3),
557 196–206. <https://doi.org/10.1007/s40641-019-00136-9>

558 Hourdin, F., Mauritsen, T., Gettelman, A., Golaz, J.-C., Balaji, V., Duan, Q., Folini, D., Ji, D., Klocke,
559 D., Qian, Y., Rauser, F., Rio, C., Tomassini, L., Watanabe, M., & Williamson, D. (2017). The Art
560 and Science of Climate Model Tuning. *Bulletin of the American Meteorological Society*, 98(3),
561 589–602. <https://doi.org/10.1175/BAMS-D-15-00135.1>

562 Huang, H., Xue, Y., Li, F., & Liu, Y. (2020). Modeling long-term fire impact on ecosystem
563 characteristics and surface energy using a process-based vegetation–fire model SSiB4/TRIFFID-
564 Fire v1.0. *Geoscientific Model Development*, 13(12), 6029–6050. [https://doi.org/10.5194/gmd-13-](https://doi.org/10.5194/gmd-13-6029-2020)
565 [6029-2020](https://doi.org/10.5194/gmd-13-6029-2020)

566 Huang, H., Xue, Y., Liu, Y., Li, F., & Okin, G. S. (2021). Modeling the short-term fire effects on
567 vegetation dynamics and surface energy in southern Africa using the improved SSiB4/TRIFFID-
568 Fire model. *Geoscientific Model Development*, 14(12), 7639–7657. [https://doi.org/10.5194/gmd-](https://doi.org/10.5194/gmd-14-7639-2021)
569 [14-7639-2021](https://doi.org/10.5194/gmd-14-7639-2021)

570 Kochkov, D., Yuval, J., Langmore, I., Norgaard, P., Smith, J., Mooers, G., Lottes, J., Rasp, S., Düben, P.,
571 Klöwer, M., Hatfield, S., Battaglia, P., Sanchez-Gonzalez, A., Willson, M., Brenner, M. P., &
572 Hoyer, S. (2023). *Neural General Circulation Models* (arXiv:2311.07222). arXiv.
573 <http://arxiv.org/abs/2311.07222>

574 Kondylatos, S., Prapas, I., Ronco, M., Papoutsis, I., Camps-Valls, G., Piles, M., Fernández-Torres, M.-Á.,
575 & Carvalhais, N. (2022). Wildfire Danger Prediction and Understanding With Deep Learning.
576 *Geophysical Research Letters*, 49(17), e2022GL099368. <https://doi.org/10.1029/2022GL099368>

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

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

585 Li, F., Zhu, Q., Riley, W. J., Zhao, L., Xu, L., Yuan, K., Chen, M., Wu, H., Gui, Z., Gong, J., &
586 Randerson, J. T. (2023). AttentionFire_v1.0: Interpretable machine learning fire model for
587 burned-area predictions over tropics. *Geoscientific Model Development*, 16(3), 869–884.
588 <https://doi.org/10.5194/gmd-16-869-2023>

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

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

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

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

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

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

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

613 Wang, S. S.-C., Leung, L. R., & Qian, Y. (2023). Projection of Future Fire Emissions Over the
614 Contiguous US Using Explainable Artificial Intelligence and CMIP6 Models. *Journal of*
615 *Geophysical Research: Atmospheres*, 128(14), e2023JD039154.
616 <https://doi.org/10.1029/2023JD039154>

617 Wang, S. S.-C., Qian, Y., Leung, L. R., & Zhang, Y. (2022). Interpreting machine learning prediction of
618 fire emissions and comparison with FireMIP process-based models. *Atmospheric Chemistry and*
619 *Physics*, 22(5), 3445–3468. <https://doi.org/10.5194/acp-22-3445-2022>

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

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

627 Xu, K.-M., & Randall, D. A. (1996). A Semiempirical Cloudiness Parameterization for Use in Climate
628 Models. *Journal of the Atmospheric Sciences*, 53(21), 3084–3102. [https://doi.org/10.1175/1520-](https://doi.org/10.1175/1520-0469(1996)053<3084:ASCPFU>2.0.CO;2)
629 [0469\(1996\)053<3084:ASCPFU>2.0.CO;2](https://doi.org/10.1175/1520-0469(1996)053<3084:ASCPFU>2.0.CO;2)

630 Zhang, T., Lin, W., Vogelmann, A. M., Zhang, M., Xie, S., Qin, Y., & Golaz, J.-C. (2021). Improving
631 Convection Trigger Functions in Deep Convective Parameterization Schemes Using Machine
632 Learning. *Journal of Advances in Modeling Earth Systems*, 13(5), e2020MS002365.
633 <https://doi.org/10.1029/2020MS002365>
634