

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

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

- 5
- 6 1. Brookhaven National Laboratory, Upton, NY, USA
 - 7 2. Met Office, FitzRoy Road, Exeter, EX13PB, UK
 - 8 3. Lawrence Livermore National Laboratory, Livermore, CA, USA
 - 9 4. Pacific Northwest National Laboratory, Richland, WA, USA
 - 10 5. Department of Geography, Ghent University, Belgium
 - 11 6. Royal Meteorological Institute of Belgium, Brussels, Belgium
 - 12 7. Department of Mathematics and Statistics, Exeter University, Exeter, UK
- 13 Correspondence to: Tao Zhang (taozhang.ccs@gmail.com)

14 **Abstract**

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

31

32 Plain Language

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

42 1. Introduction

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

54
55 In recent years, machine learning (ML) techniques have emerged as a promising approach to
56 improve parameterizations in ESMs. They are capable of learning complex patterns and
57 relationships directly from observational data or high-resolution simulations, enabling the
58 capture of nonlinearities and intricate interactions that may be challenging to represent with
59 traditional parameterizations. For example, Zhang et al. (2021) proposed a ML trigger function
60 for a deep convection parameterization by learning from field observations, demonstrating its
61 superior accuracy compared to traditional CAPE-based trigger functions. Chen et al. (2023)
62 developed a neural network-based cloud fraction parameterization, better predicting both spatial

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

69
70 However, the aforementioned studies primarily focus on offline ML of parameterizations that do
71 not directly interact with ESMs. Recently, there have been efforts to implement ML
72 parameterizations that can be directly coupled with ESMs. Several studies have developed ML
73 parameterizations in ESMs by hard coding custom neural network modules, such as O’Gorman
74 & Dwyer (2018), Rasp et al. (2018), Han et al. (2020) and Gettelman et al. (2021). They
75 incorporated a Fortran-based ML inference module to allow the loading of the pre-trained ML
76 weights to reconstruct the ML algorithm in ESMs. The hard-coding has limitations. Such hard-
77 coding approach restricts the ML algorithm’s ability to adapt to changes in the model dynamics
78 over time, as the ‘online’ updating requires a two-way coupling between the dominantly Fortran-
79 based ESMs and Python ML libraries.

80
81 Fortran-Keras Bridge (FKB; Ott et al. (2020)) and C Foreign Function Interface (CFFI;
82 <https://cffi.readthedocs.io>) are two packages that support two-way coupling between Fortran-based ESM
83 and Python based ML parameterizations. FKB enables tight integration of Keras deep learning models but
84 is specifically bound to the Keras library, limiting its compatibility with other frameworks like PyTorch
85 and Scikit-Learn. On the other hand, CFFI provides a more flexible solution that in principle supports
86 coupling various ML packages due to its language-agnostic design. Brenowitz & Bretherton (2018)
87 utilized it to enable the calling of Python ML algorithms within ESMs. However, the CFFI has several
88 limitations. When utilizing CFFI to interface Fortran and Python, it uses global data structures to pass
89 variables between the two languages. This approach results in additional memory overhead as variable
90 values need to be copied between languages, instead of being passed by reference. Additionally, CFFI
91 lacks automatic garbage collection for the unused memory within these data structures and copies.
92 Consequently, the memory usage of the program gradually increases over its lifetime. In addition, when
93 using CFFI to call Python functions from a Fortran program, the process involves several steps such as
94 registering variables into a global data structure, calling the Python function, and retrieving the calculated

95 result. These multiple steps can introduce computational overhead due to the additional operations
96 required.

97
98 Additionally, Wang et al. (2022) developed a coupler to facilitate two-way communication between ML
99 parameterizations and host ESMs. The coupler gathers state variables from the ESM using the Message
100 Passing Interface (MPI) and transfers them to a Python-based ML module. It then receives the output
101 from the Python code and returns them to the ESM. While this approach effectively bridges Fortran and
102 Python, its use of file-based data passing to exchange information between modules carries some
103 performance overhead relative to tighter coupling techniques. Optimizing the data transfer, such as via
104 shared memory, remains an area for improvement to fully leverage this coupler's ability to integrate
105 online-adaptive ML parameterizations within large-scale ESM simulations, which is the main goal for this
106 study.

107
108 In this study, we investigate the integration of ML parameterizations into Fortran-based ESM
109 models by establishing a flexible interface that enables the invocation of ML algorithms in
110 Python from Fortran. This integration offers access to any Python codes from Fortran, including
111 a diverse range of ML frameworks, such as PyTorch, TensorFlow, and Scikit-learn, which can
112 effectively be utilized for parameterizing intricate atmospheric and other climate system
113 processes. The coupling of the Fortran model and the Python ML code needs to be performed for
114 thousands of model columns and over thousands of timesteps for a typical model simulation.
115 Therefore, it is crucial for the coupling interface to be both robust and efficient. We showcase the
116 feasibility and benefits of this approach through case studies that involve the parameterization of
117 deep convection and wildfire processes in ESMs. The two cases demonstrate the robustness and
118 efficiency of the coupling interface. The focus of this paper is on documenting the coupling
119 between the Fortran ESM and the ML algorithms and systematically evaluating the
120 computational efficiency and memory usage of different ML frameworks (such as Pytorch and
121 TensorFlow), different ML algorithms, and different configuration of a climate model. The
122 assessment of the scientific performance of the ML emulators will be addressed in follow-on
123 papers. The showcase examples emphasize the potential for high modularity and reusability by
124 separating the ML components into Python modules. This modular design facilitates independent
125 development and testing of ML-based parameterizations by researchers. It enables easier code
126 maintenance, updates, and the adoption of state-of-the-art ML techniques with only minimal

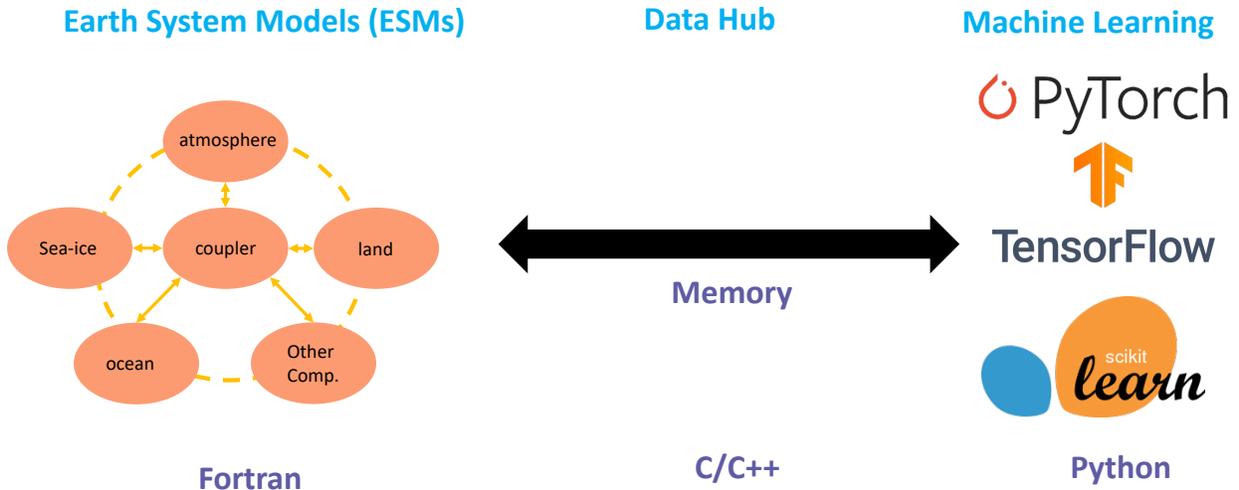
127 disrupting the existing Fortran infrastructure. Ultimately, this advancement will contribute to
128 enhanced predictions and a deeper comprehension of the evolving climate of our planet. It is
129 important to note that the current interface only supports executing deep learning algorithms on CPUs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

131
132 The rest of this manuscript is organized as follows: Section 2 presents the detailed interface that
133 integrates ML into Fortran-based ESM models. Section 3 discusses the performance of the
134 interface and presents its application in two case studies. Finally, Section 4 provides a summary
135 of the findings and a discussion of their implications.

136 2. General design of the ML interface

137 2.1 Architecture of the ML interface

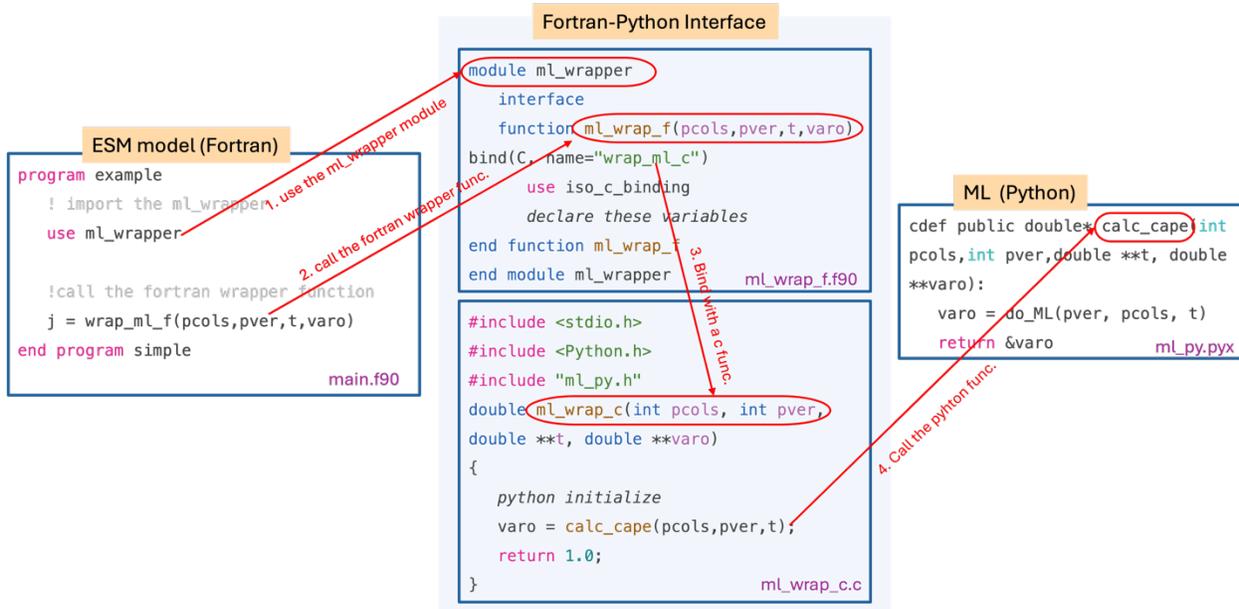
138 We developed an interface using shared memory to enable two-way coupling between Fortran and Python
139 (Figure 1). The ESM used in the demonstration in Figure 1 is the U.S. Department of Energy (DOE)
140 Energy Exascale Earth System Model (E3SM; Golaz et al., 2019, 2022). Because Fortran cannot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



149
 150 **Figure 1.** The interface of the ML bridge for two-way communication via memory between Fortran ESM
 151 and Python ML module.

152 **2.2 Code structure**

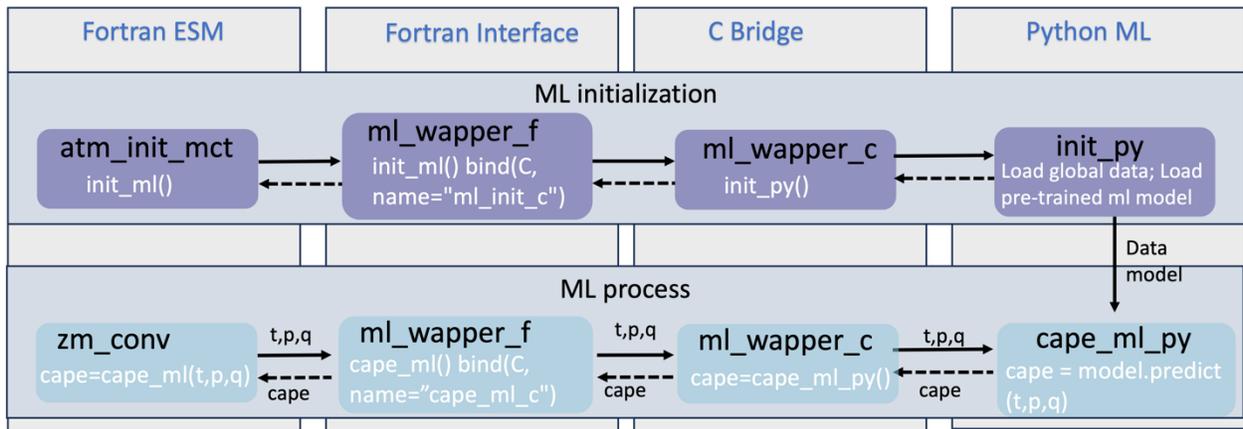
153 Figure 2 illustrates how the framework operates using toy code example. The Fortran-Python interface
 154 comprises a Fortran wrapper and C wrapper files, which are bound together. The Fortran-based ESM first
 155 imports the Fortran wrapper, allowing it to call wrapper functions with input and output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



158
 159 **Figure 2.** Toy code illustrating the Fortran-Python interface.

160
 161
 162
 163
 164
 165
 166
 167
 168
 169
 170
 171
 172
 173
 174

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

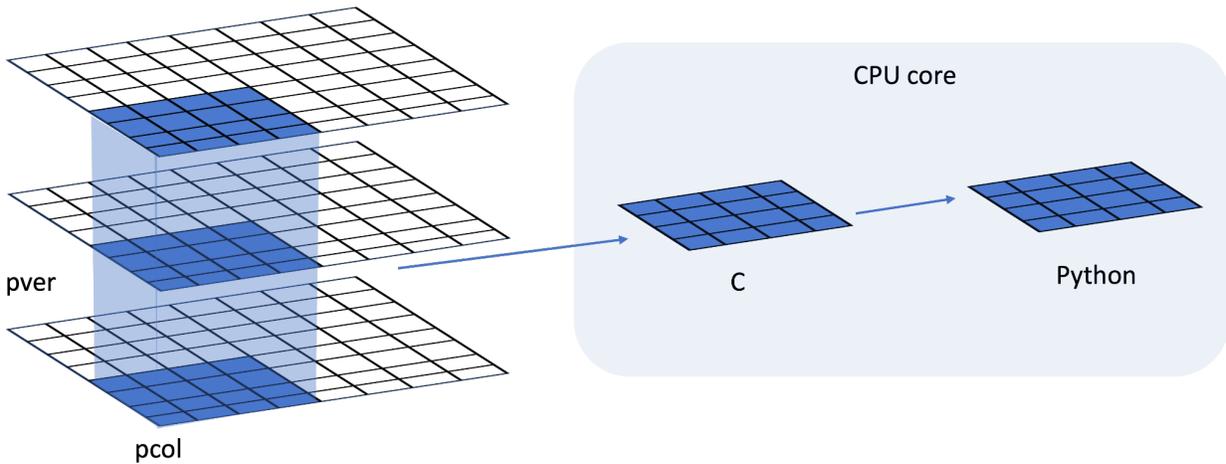


175
 176
 177
 178

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

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

186 neural network computations. If the ML were called separately for every column, parallel efficiencies
187 would not be realized. By aggregating inputs over the chunk-scale prior to interfacing with Python,
188 performance is improved through better utilization of multi-core and GPU-based ML capabilities during
189 parameterization calculations.
190



191
192 **Figure 4.** Data and system structure. The model domain is decomposed into chunks of columns. pver
193 refers to number of pressure vertical levels. A chunk contains multiple columns (up to pcol). Multiple
194 chunks can be assigned to each CPU core.

195
196

197 3. Results

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

205

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

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

214
215 To comprehensively assess performance, we conducted a systematic evaluation of various ML
216 frameworks, ML algorithms, and physical models. This evaluation is built upon the foundations
217 established for evaluating the ML trigger function in the deep convection parameterization.

218 3.1 Application cases

219 3.1.1 ML trigger function in deep convection parameterization

220 In General Circulation Models, uncertainties in convection parameterizations are recognized to be closely
221 linked to the convection trigger function used in these schemes (Bechtold et al., 2004; Xie et al., 2004,
222 2019; Xie & Zhang, 2000; Lee et al., 2007). The convective trigger in a convective parameterization
223 determines when and where model convection should be triggered as the simulation advances. In many
224 convection parameterizations, the trigger function consists of a simple, arbitrary threshold for a physical
225 quantity, such as convective available potential energy (CAPE). Convection will be triggered if the CAPE
226 value exceeds a threshold value.

227
228 In this work, we use this interface to test a newly developed ML trigger function in E3SM. The ML
229 trigger function was developed with the training data originating from simulations performed using the
230 kilometer-resolution (1.5 km grid spacing). Met Office Unified Model Regional Atmosphere 1.0
231 configuration (Bush et al., 2020). Each simulation consists of a limited area model (LAM) nested within a
232 global forecast model providing boundary conditions (Walters et al., 2017; Webster et al., 2008). In total
233 80 LAM simulations were run located so as to sample different geographical regions worldwide. Each
234 LAM was run for 1 month, with 2-hourly output, using a grid-length of 1.5 km, a 512 x 512 domain, and
235 a model physics package used for operational weather forecasting. The 1.5 km data is coarse-grained to
236 several scales from 15 to 144 km.

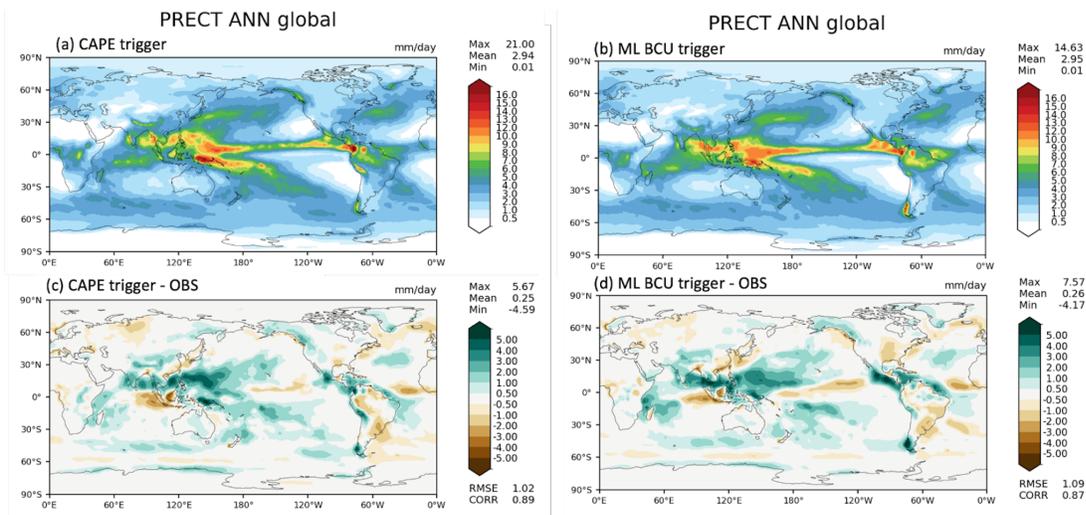
237
238 A two-stream neural network architecture is used for the ML model. The first stream takes profiles of
239 temperature, specific humidity and pressure across 72 levels at each scale as inputs and passes them
240 through a 4-layer convolutional neural network (CNN) with kernel sizes of 3, to extract large scale

241 features. The second stream takes mean orographic height, standard deviation of orographic height, land
 242 fraction and the size of the grid-box as inputs. The outputs of the two streams are then combined and fed
 243 into a 2-layer fully connected network to allow the ML model to leverage both atmospheric and surface
 244 features when making its predictions. The output is a binary variable indicating whether the convection
 245 happens, based on the condition of buoyant cloudy updrafts (BCU, e.g. Hartmann et al., 2019; Swann,
 246 2001). If there are 3 contiguous levels where the predicted BCU is larger than 0.05, the convection
 247 scheme is triggered. Once trained, the CNN is coupled to E3SM and thermodynamic information from
 248 E3SM is passed to it to predict the trigger condition. Then, the predicted result is returned to E3SM.

249

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

255



256

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

259 3.1.2 ML learning fire model

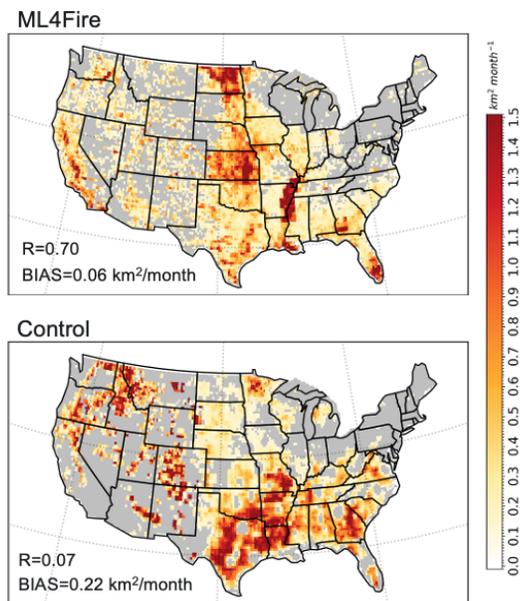
260 Predicting wildfire burned area is challenging due to the complex interrelationships between fires,
 261 climate, weather, vegetation, topography, and human activities (Huang et al., 2020). Traditionally,
 262 statistical methods like multiple linear regression have been applied, but are limited in the number and
 263 diversity of predictors considered (Yue et al., 2013). In this study, we develop a coupled fire-land-

264 atmosphere framework that uses machine learning to predict wildfire area, enhancing long-term burned
265 area projections and assessing fire impacts by enabling simulations of interactions among fire,
266 atmosphere, land cover, and vegetation.

267
268 The ML algorithm is trained using a monthly dataset, which includes the target variable of burned area, as
269 well as various predictor variables. These predictors encompass local meteorological data (e.g., surface
270 temperature, precipitation), land surface properties (e.g., monthly mean evapotranspiration and surface
271 soil moisture), and socioeconomic variables (e.g., gross domestic product, population density), as
272 described by Wang et al. (2022). In the coupled fire-land-atmosphere framework, meteorology variables
273 and land surface properties are provided by the E3SM. We use the eXtreme Gradient Boosting algorithm
274 implemented in Scikit-Learn to train the ML fire model. Figure 6 demonstrates that the ML4Fire model
275 exhibits superior performance in terms of spatial distribution compared to process-based fire models,
276 particularly in the Southern US region. Detailed analysis will be presented in a separate paper. The
277 ML4Fire model has proven to be a valuable tool for studying vegetation-fire interactions, enabling
278 seamless exploration of climate-fire feedbacks.

279

280

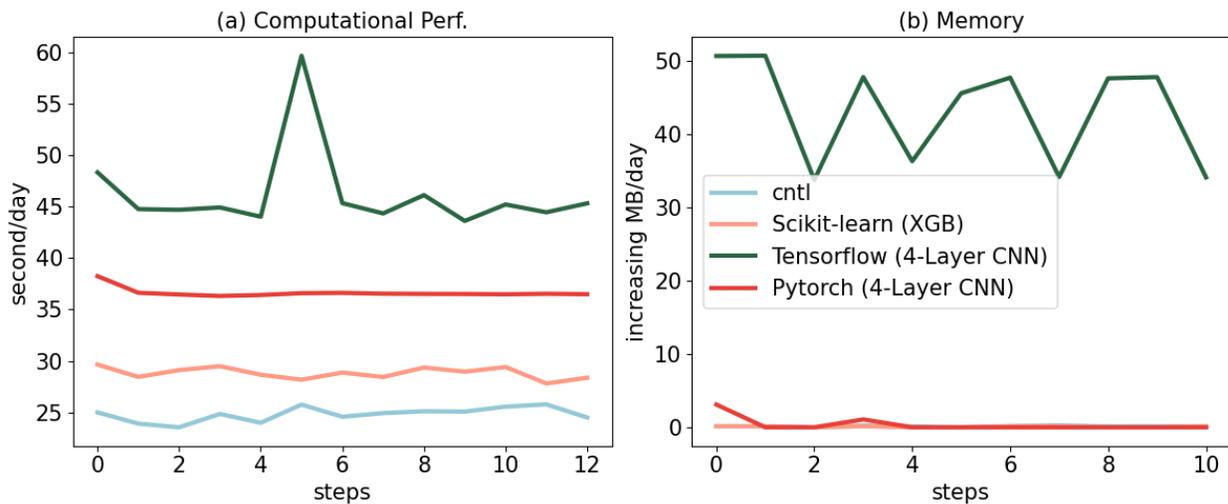


281

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

285 3.2 Performance of different ML frameworks

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



301
302 **Figure 7.** Computational and memory overhead as the simulation progresses for coupling the ML trigger
303 function with the E3SM model. The x-axis represents the simulated time step. The y-axis of (a) represents
304 the simulation speed measured in seconds per day (indicating the number of seconds required to simulate
305 one day). The y-axis of (b) represents the relative increase in memory usage for Scikit-learn, TensorFlow,
306 and PyTorch compared with CNTL. CNTL represents the original simulation without using the ML
307 framework.
308

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

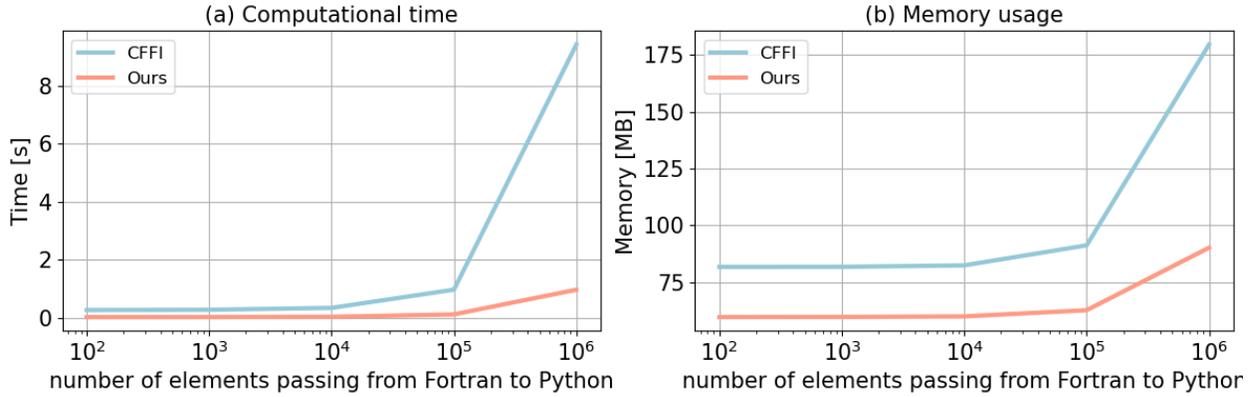
323

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

333

334 In Figure 8, we present a comparison between the two frameworks by testing the different number of
335 elements passed from Fortran to Python. The evaluation is based on a demo example that focuses solely
336 on declaring arrays and transferring them from Fortran to Python, rather than a real E3SM simulation.
337 Figure 8a illustrates the impact of the number of passing elements on the overhead of the two interfaces.
338 As the number of elements exceeds 10^4 , the overhead of CFFI becomes significant. When the number
339 surpasses 10^6 , the overhead of CFFI is nearly ten times greater than that of our interface. Regarding
340 memory usage, our interface maintains a stable memory footprint of approximately 60MB. Even as the
341 number of elements increases, the memory usage only shows minimal growth. However, for CFFI, the

342 memory usage starts at 80MB, which is 33% higher than our interface. As the number of elements
343 reaches 10^6 , the memory overhead for CFFI dramatically rises to 180MB, twice as much as our interface.
344



345
346 Figure 8. Comparison of our framework and the CFFI framework in terms of computational time
347 and memory usage. The x-axis represents the number of elements transferred from Fortran to
348 Python, while the y-axis displays the total time (a) and total memory usage (b) for a
349 demonstration example. The evaluations presented are based on the average results obtained
350 from 5 separate tests.

351

352 3.3 Performance of ML algorithms of different complexities

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

364

365

366

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

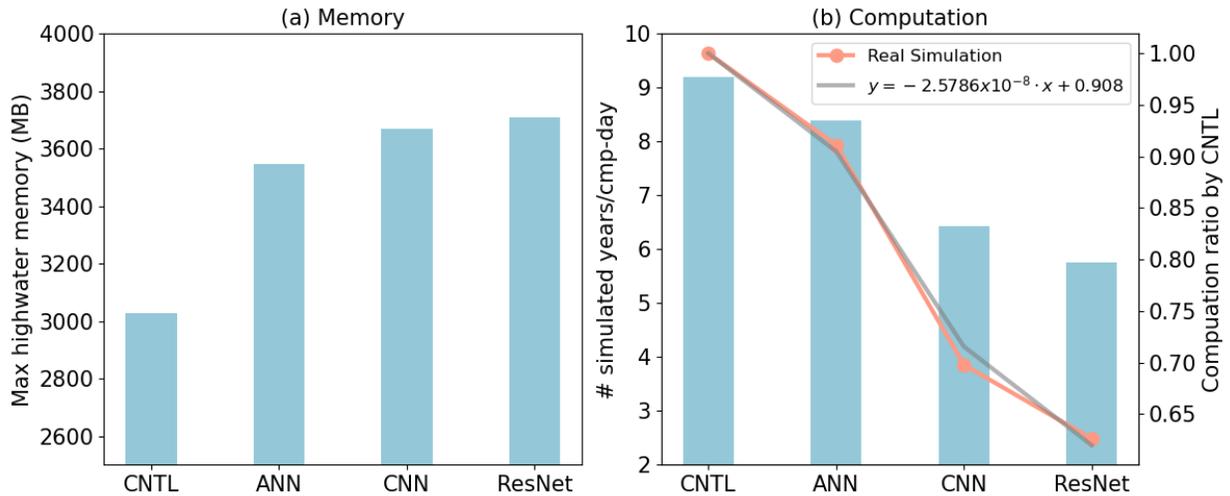
| Algorithms | Structure | # of parameters |
|------------|--------------------------|-----------------|
| ANN | 3 x Linear | 121,601 |
| CNN | 4 x Conv2d + 2 x Linear | 7,466,753 |
| ResNet | 17 x Conv2d + 1 x Linear | 11,177,025 |

368
 369 Figure 9 presents a comparison of the memory and computational costs between the CNTL run without
 370 deep learning parameterization and the hybrid run with various deep learning algorithms. The same
 371 specific process-element layout (placement of ESM component models on distributed CPU cores) is used
 372 for all the simulations. Deep learning algorithms incur a significant yet affordable increase in memory
 373 overhead, with at least a 20% increase compared to the CNTL run (Figure 9a). This is primarily due to the
 374 integration of ML algorithms into the ESM, which persist throughout the simulations. Although there is a
 375 notable increase in complexity among the deep learning algorithms, their memory usage only shows a
 376 slight rise. This is because the memory increment resulting from the ML parameters is relatively small.
 377 Specifically, ANN requires 1MB of memory, CNN requires 60MB, and the ResNet algorithms requires
 378 85MB, which are calculated based on the number of parameters in each algorithm. When comparing these
 379 values to the memory consumption of the CNTL run, which is approximately 3000MB, the additional
 380 parameters' incremental memory consumption is not substantial. However, when we use 128 MPI
 381 processes per node, it could bring the total memory requirement to approximately 460 GB per node. If the
 382 available hardware memory is less than this, the process layout must be adjusted accordingly.

383
 384 In terms of computational performance, the Python-based ML calls inevitably introduce some overhead.
 385 However, as shown in Figure 9b, the performance decrease is not substantial. The simple ANN model
 386 reduces performance by only about 10% compared to the CNTL run, while even the more complex
 387 ResNet model results in a 35% decrease. In contrast, Wang et al. (2022) reported a 100% overhead in
 388 their interface, which transfers parameters via files. It is worth noting that in this study, the deep learning
 389 algorithms are executed on CPUs. To enhance computational performance, future work could consider
 390 utilizing GPUs for acceleration.

391
 392 In addition, we develop a performance model to estimate computational performance for the hybrid
 393 model using different ML model sizes and complexities. This performance model, based on linear
 394 regression, predicts the computational ratio relative to the CNTL run by taking the number of ML

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

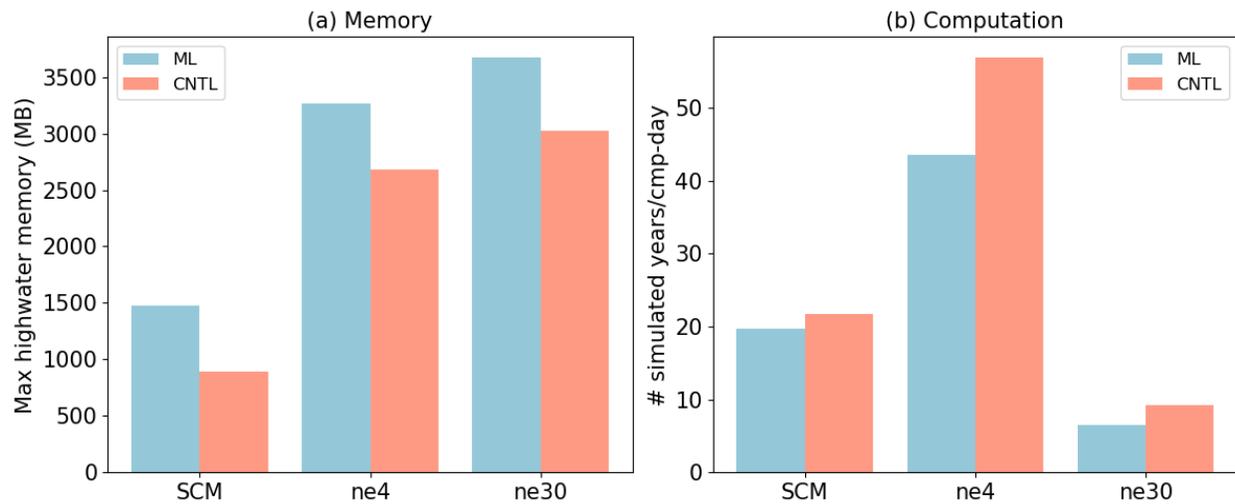


398
 399 **Figure 9.** Comparison of CNTL and the hybrid model using various ML algorithms in terms of memory
 400 and computation. CNTL is the default run without ML parameterizations. In (b), the left y-axis represents
 401 the actual number of simulated years per day, while the right y-axis shows the relative performance
 402 compared to the CNTL run (orange line). The gray line illustrates the regression between the number of
 403 ML parameters (x) and the relative performance of the hybrid system (y).
 404

405 3.4 Performance for physical models of different complexities

406 ML parameterization can be applied to various ESM configurations, for example, with the E3SM
 407 Atmosphere Model (EAM), we experiment with Single Column Model (SCM), the ultra low-resolution
 408 model of EAM (ne4), and the nominal low resolution model of EAM (ne30) configurations. The SCM
 409 consists of one single atmosphere column of a global EAM (Bogenschutz et al., 2020; Gettelman et al.,
 410 2019). ne4 has 384 columns, with each column representing the horizontal resolution of 7.5° . ne30 is the
 411 default resolution for EAM and comprises 21,600 columns, with each column representing the horizontal
 412 resolution of 1° . In the case of the ML trigger function, the memory overhead is approximately 500MB
 413 for all configurations due to the loading of the ML algorithm, which does not vary with the configuration
 414 of the ESM.

415



416
 417 **Figure 10.** Comparison of CNTL and ML for various ESMs in terms of memory and computation. The
 418 ESM configuration include SCM, ultra-low resolution model (ne4) and nominal low-resolution model
 419 (ne30).

420
 421 Regarding computational performance, SCM utilizes 1 process, ne4 employs 1 node with 64 processes,
 422 and ne30 utilizes 10 nodes with each node using 128 processes. In the case of SCM, the overhead
 423 attributed to the ML parameterization is approximately 9% due to the utilization of only 1 process.
 424 However, for ne4 and ne30, the overhead is 23% and 28% respectively (Figure 10). The increasing
 425 computational overhead is primarily due to resource competition when multiple processes are used within
 426 a single node. It is noted that although there is a significant computational gap between ML and CNTL
 427 for ne4, the relative performance between ML and CNTL for ne4 is approximately 76.7%, which is close
 428 to ne30 at 71.4%.

429
 430 **4. Discussion and Conclusion**

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

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

444
445 In this study, we develop a novel Fortran-Python interface for developing ML parameterizations. This
446 interface demonstrates feasibility in supporting various ML frameworks, such as PyTorch, TensorFlow,
447 and Scikit-learn and enables the effective development of new ML-based parameterizations to explore
448 ML-based applications in ESMs. Through two cases - a ML trigger function in convection
449 parameterization and a ML wildfire model - we highlight high modularity and reusability of the
450 framework. We conduct a systematic evaluation of memory usage and computational overhead from the
451 integrated Python codes.

452
453 Based on our performance evaluation, we observe that coupling ML algorithms using TensorFlow into
454 ESMs can lead to memory leaks. As a recommendation, we suggest using PyTorch for complex deep
455 learning algorithms and Scikit-learn for simple ML algorithms for the Fortran-Python ML interface.

456
457 The memory overhead primarily arises from loading ML algorithms into ESMs. If the ML algorithms are
458 implemented using PyTorch or Scikit-learn, the memory usage will not increase significantly. The
459 computational overhead is influenced by the complexity of the neural network and the number of
460 processes running on a single node. As the complexity of the neural network increases, more parameters
461 in the neural network require forward computation. Similarly, when there are more processes running
462 on a single node, the integrated Python codes introduce more resource competition.

463
464 Although this interface provides a flexible tool for ML parameterizations, it does not currently utilize
465 GPUs for ML algorithms. In Figure 3, it is shown that each chunk is assigned to a CPU core. However, to
466 effectively leverage GPUs, it is necessary to gather the variables from multiple chunks and pass them to
467 the GPUs. Additionally, if an ESM calls the Python ML module multiple times in each time step, the
468 computational overhead becomes significant. It is crucial to gather the variables and minimize the number
469 of calls. In the future, we will enhance the framework to support this mechanism, enabling GPU
470 utilization and overall performance improvement.

471 Acknowledge

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

480

481 Author contribution

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

487 Conflict of Interest

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

489

490 Data Availability Statement

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

497 References

498 Bechtold, P., Chaboureau, J.-P., Beljaars, A., Betts, A. K., Köhler, M., Miller, M., & Redelsperger, J.-L.
499 (2004). The simulation of the diurnal cycle of convective precipitation over land in a global

500 model. *Quarterly Journal of the Royal Meteorological Society*, 130(604), 3119–3137.
501 <https://doi.org/10.1256/qj.03.103>

502 Bogenschutz, P. A., Tang, S., Caldwell, P. M., Xie, S., Lin, W., & Chen, Y.-S. (2020). The E3SM version
503 1 single-column model. *Geoscientific Model Development*, 13(9), 4443–4458.
504 <https://doi.org/10.5194/gmd-13-4443-2020>

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

508 Brenowitz, N. D., & Bretherton, C. S. (2019). Spatially extended tests of a neural network
509 parametrization trained by coarse-graining. *Journal of Advances in Modeling Earth Systems*,
510 11(8), 2728–2744. <https://doi.org/10.1029/2019ms001711>

511 Bush, M., Allen, T., Bain, C., Boutle, I., Edwards, J., Finnenkoetter, A., Franklin, C., Hanley, K., Lean,
512 H., Lock, A., Manners, J., Mittermaier, M., Morcrette, C., North, R., Petch, J., Short, C., Vosper,
513 S., Walters, D., Webster, S., ... Zerroukat, M. (2020). The first Met Office Unified Model–
514 JULES Regional Atmosphere and Land configuration, RAL1. *Geoscientific Model Development*,
515 13(4), 1999–2029. <https://doi.org/10.5194/gmd-13-1999-2020>

516 Chen, G., Wang, W., Yang, S., Wang, Y., Zhang, F., & Wu, K. (2023). A Neural Network-Based Scale-
517 Adaptive Cloud-Fraction Scheme for GCMs. *Journal of Advances in Modeling Earth Systems*,
518 15(6), e2022MS003415. <https://doi.org/10.1029/2022MS003415>

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

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

524 Gettelman, A., Gagne, D. J., Chen, C.-C., Christensen, M. W., Lebo, Z. J., Morrison, H., & Gantos, G.
525 (2021). Machine Learning the Warm Rain Process. *Journal of Advances in Modeling Earth*
526 *Systems*, 13(2), e2020MS002268. <https://doi.org/10.1029/2020MS002268>

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

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

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

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

546 Hartmann, D. L., Blossey, P. N., & Dygert, B. D. (2019). Convection and Climate: What Have We
547 Learned from Simple Models and Simplified Settings? *Current Climate Change Reports*, 5(3),
548 196–206. <https://doi.org/10.1007/s40641-019-00136-9>

549 Hourdin, F., Mauritsen, T., Gettelman, A., Golaz, J.-C., Balaji, V., Duan, Q., Folini, D., Ji, D., Klocke,
550 D., Qian, Y., Rauser, F., Rio, C., Tomassini, L., Watanabe, M., & Williamson, D. (2017). The Art
551 and Science of Climate Model Tuning. *Bulletin of the American Meteorological Society*, 98(3),
552 589–602. <https://doi.org/10.1175/BAMS-D-15-00135.1>

553 Huang, H., Xue, Y., Li, F., & Liu, Y. (2020). Modeling long-term fire impact on ecosystem
554 characteristics and surface energy using a process-based vegetation–fire model SSiB4/TRIFFID-
555 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)
556 [6029-2020](https://doi.org/10.5194/gmd-13-6029-2020)

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

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

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

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

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

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

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

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

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

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

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

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

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

603 Zhang, T., Morcrette, C., Zhang, M., Lin, W., Xie, S., Liu, Y., Weverberg, K. V., & Rodrigues, J. (2024).
604 *tzhang-ccs/ML4ESM: ML4ESM_v1* (Version v1) [Computer software]. [object Object].
605 <https://doi.org/10.5281/ZENODO.11005103>
606