Hindawi
Scientific Programming
Volume 2020, Article ID 8888811, 13 pages
https://doi.org/10.1155/2020/8888811

Research Article

A Fortran-Keras Deep Learning Bridge for Scientific Computing

Jordan Ott,[1] Mike Pritchard,[2] Natalie Best,[3] Erik Linstead,[3] Milan Curcic,[4] and Pierre Baldi[1]

[1]*Department of Computer Science, University of California, Irvine, CA, USA*
[2]*Department of Earth System Science, University of California, Irvine, CA, USA*
[3]*Fowler School of Engineering, Chapman University, Orange, CA, USA*
[4]*Department of Ocean Sciences, University of Miami, Coral Gables, FL, USA*

Correspondence should be addressed to Pierre Baldi; pfbaldi@ics.uci.edu

Received 20 April 2020; Accepted 7 August 2020; Published 28 August 2020

Academic Editor: Manuel E. Acacio Sanchez

Copyright © 2020 Jordan Ott et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Implementing artificial neural networks is commonly achieved via high-level programming languages such as Python and easy-to-use deep learning libraries such as Keras. These software libraries come preloaded with a variety of network architectures, provide autodifferentiation, and support GPUs for fast and efficient computation. As a result, a deep learning practitioner will favor training a neural network model in Python, where these tools are readily available. However, many large-scale scientific computation projects are written in Fortran, making it difficult to integrate with modern deep learning methods. To alleviate this problem, we introduce a software library, the Fortran-Keras Bridge (FKB). This two-way bridge connects environments where deep learning resources are plentiful with those where they are scarce. The paper describes several unique features offered by FKB, such as customizable layers, loss functions, and network ensembles. The paper concludes with a case study that applies FKB to address open questions about the robustness of an experimental approach to global climate simulation, in which subgrid physics are outsourced to deep neural network emulators. In this context, FKB enables a hyperparameter search of one hundred plus candidate models of subgrid cloud and radiation physics, initially implemented in Keras, to be transferred and used in Fortran. Such a process allows the model's emergent behavior to be assessed, i.e., when fit imperfections are coupled to explicit planetary-scale fluid dynamics. The results reveal a previously unrecognized strong relationship between offline validation error and online performance, in which the choice of the optimizer proves unexpectedly critical. This in turn reveals many new neural network architectures that produce considerable improvements in climate model stability including some with reduced error, for an especially challenging training dataset.

1. Introduction

The Fortran programming language was originally developed in the 1950s and published in 1957. It was created to help programmers implement solutions for scientific and engineering problems on the IBM 704 computer, which at the time needed to be written in machine or assembly language. Fortran has been regarded as revolutionary and possibly one of the most influential software products in history [1]. Having evolved many times since its creation, with the most recent release in 2018, each version adds new features and capabilities. Fortran initially gained popularity and remains a widely used language due to its fast and efficient computational ability. Additionally, Fortran's strength is its backward compatibility, which allows modern compilers to build code written in the 60s and 70s.

Though not as popular as it once was, Fortran is still used in specialized fields, including oceanography, solid mechanics, computational physics, earthquake simulation, climate modeling, and aerospace. Because of Fortran's continued use, a great deal of legacy code and new code exists. Unfortunately, it is difficult to rewrite all existing code bases in more mainstream languages, due to their size and complexity. Therefore, when algorithms and extensive libraries are created in modern languages, backwards compatible methods must be developed to make them available in older legacy code, such as Fortran.

In recent years, the rise of machine learning and deep learning has led to successful applications in various domains. Substantial improvements in the size of the training sets and available computing power have led to a new wave of implementations [2, 3]. In turn, this success has increased the usage and dissemination of deep learning. These methods have been applied to a variety of domains, e.g., ranging from remote sensing [4, 5] to computer vision [6–10], and to games [11, 12]. Specifically, within scientific computing, many advancements have been achieved through the application of neural networks. Neural networks have been augmented with physically informed capabilities [13, 14], better suiting them for conservation restrictions. Learning partial differential equations [15, 16] has proved valuable in multiple scientific domains.

The success and popularity of deep learning have inspired the creation of powerful software libraries written in several modern programming languages. However, Fortran is not among the modern languages that benefit from these deep learning libraries. This absence leaves Fortran programmers with few options to implement deep neural networks.

The implementation of deep neural networks, in Fortran, may be achieved via two primary pathways. One solution is to rewrite all existing deep learning libraries in Fortran. The second solution is to leverage existing frameworks and bridge available functionalities to Fortran. The former is extremely arduous and time consuming, considering the size and scope of existing deep learning packages and the dizzying pace of their evolution [17–19]. The latter approach, which this paper describes, is to allow users to leverage the power of existing frameworks while providing a bridge between paradigms where deep learning resources are plentiful and those where they are scarce. In this way, we can leverage aspects of currently available deep learning software libraries, such as Keras [20], and bring them to large-scale scientific computing packages written in Fortran. To this end, we propose the Fortran-Keras Bridge (FKB), a two-way bridge connecting models in Keras with ones available in Fortran. The source code is publicly available and can be found in https://github.com/scientific-computing/FKB. We begin by reviewing existing Fortran projects that would benefit from the integration of FKB.

2. Fortran Projects

FKB can be integrated with many existing large-scale and computationally intensive projects written in Fortran. These projects will benefit from the easy integration of neural network models, which FKB makes possible.

For example, Fortran is used to do a great deal of work in climate and ocean modeling. For instance, the US-produced Community Earth System Model [21] is written in object-oriented Fortran-90; this is the most widely used climate model in the world, so are the other climate simulation codes used by the US Department of Energy [22] and the National Oceanographic and Atmospheric Administration's Geophysical Fluid Dynamics Laboratory [23]. Meanwhile, the Nucleus for European Modelling of the Ocean (NEMO)[26] engine is used for studying ocean circulation problems on regional and global scales and making future predictions and is also written in Fortran. The Hybrid Coordinate Ocean Model (HYCOM)[27], also used for ocean modeling, extends traditional ocean models to allow for a smooth transition from the deep ocean to coastal regimes. Researchers have also developed models for the modeling of waves and wind stress [26]. The Weather Research and Forecasting Model (WRF) is arguably the most widely used numerical weather prediction models for regional decision support [27]. Since its release in 2000, the number of WRF registrations has grown to over 36,000. WRF produces atmospheric simulations with support for special applications, including air chemistry, hydrology, wildland fires, hurricanes, and regional climate, and is again a Fortran-based model.

Fortran has found continued use in solid mechanics packages for implementing finite element methods. Popular packages such as ANSYS [28], ABAQUS [29], and LS-DYNA [30] are written in Fortran or accept Fortran subroutines. Similarly, in earthquake modeling, the SPECFEM3D [31] package leverages Fortran for simulations.

The list goes on. Code Saturne [32], developed by Électricité de France, and NEK5000 [33] are Fortran open-source computational fluid dynamics packages. Code_Saturne allows for user customization via Fortran subroutines, which is just one application domain for FKB. NEK5000 is actively used in the Center for Exascale Simulation of Advanced Reactors (CESAR) projects. Fortran has also been continually used for molecular modeling within chemistry and physics. The Chemistry at Harvard Macromolecular Mechanics (CHARMM) Development Project has produced a powerful molecular simulation program in Fortran [34]. This simulation program not only primarily targets biological systems but can also be used for inorganic materials. A similar tool, NWChem, has been developed by the Molecular Sciences Software Group at the Pacific Northwest National Laboratory [35]. NWChem is a computational chemistry software that includes quantum chemical and molecular dynamics functionalities. Within the molecular physics domain, Fluktuierende Kaskade (FLUKA) is a proprietary tool for calculations of particle transport and interactions with matter [36].

The models mentioned above and projects can leverage the FKB library to leverage neural networks within their codebases. For example, neural networks have proven useful in modeling sea surface temperature cooling for typhoon forecasting [37]. Therefore, the integration of FKB with tools such as NEMO, HYCOM, or WRF models is a possibility. In a recent study of computational fluid dynamics, Ling et al. solve the Reynolds-averaged Navier–Stokes equations, similar to Code_Saturne and NEK5000. By implementing deep neural networks, the authors report that the architecture improved prediction accuracy [38]. Finally, the Fluka tool contains a wide range of molecular physics applications, including dosimetry calculations. Vega-Carrillo et al. have shown neural networks aided in the calculation of neutron doses [39]. For global climate simulation, there is proof that deep neural networks can offer skillful alternatives to

assumption-prone approximations of subgrid cloud and turbulence physics in the atmosphere [40, 41]. We hope that the FKB library enables Fortran users to expand their research and projects to include neural networks.

Having reviewed several Fortran-based projects that can leverage FKB, we now introduce the two sides of this bridge. The following sections will develop the foundations on which to anchor each side of this two-way bridge. We start by introducing the deep learning anchor.

3. The Python Anchor (Deep Learning)

Many programming languages offer tools and libraries for implementing artificial neural networks. However, in recent years, Python has emerged as the clear favorite within this domain. Metrics in Figure 1 display Python's dominance. Python is used nearly 50% more than the second most popular language; R. Python's ubiquitous presence in machine learning makes it the obvious choice to leverage existing libraries for Fortran. The question then becomes, which available software library within Python is best suited to bridge to Fortran?

Of the available deep learning libraries, Keras [18] is the most popular among practitioners (Figure 1(b)). Keras is an Application Programming Interface (API) built on top of TensorFlow [17], which provides users the ability to implement quickly, train, and test networks. This convenience encapsulates much of the low-level complexity one must manage when implementing deep networks from scratch. Keras abstracts many of the complicated aspects of TensorFlow while still providing customizability and ease of use. This combination makes Keras the first choice of many for deep learning applications. As a result of its popularity and ease of use, Keras is the clear choice on which to build one end of the two-way bridge.

Figure 2 depicts the positioning of the Python anchor, FKB/P, within the deep learning ecosystem. The Keras API leverages Python to build deep neural networks. FKB/P resides on top of Keras to access models produced from Keras and transmit them to the Fortran anchor, FKB/F. This structure allows for integration with Fortran applications that wish to leverage deep neural network architectures. Having described the deep learning anchor within Python, Section 4 develops the foundation for anchoring the bridge with Fortran.

4. The Fortran Anchor (Scientific Computing)

Several attempts have been made to implement neural networks in Fortran, with some success [43–47]. However, many implementations resort to hacking a single-use neural network by hand, or binding code from other languages [47]. Along these lines, one may consider accessing Python functionality directly from Fortran, by running a Python instance within Fortran. While providing flexibility and ease of use, this is vulnerable to extreme deficiencies in speed and computational resources. As a result, this solution becomes untenable for large-scale computation projects such as the ones described in Section 2.

There are a small number of existing neural network libraries in Fortran [46–48]. The most recent and well developed library is Neural Fortran [46], a lightweight neural network library, written natively in Fortran. The Neural Fortran library provides the ability to implement artificial neural networks of arbitrary size with data-based parallelism. Additionally, in benchmark studies, Neural Fortran was shown to have comparable compute performance with Keras while maintaining a lower memory footprint. This library offers a foundation to anchor the Fortran side of the two-way bridge, FKB/F. By extending and building on top of-Neural Fortran, we can convert Keras models to ones readily available in Fortran and implement them in existing Fortran projects.

The positioning of FKB within the scientific computing ecosystem is shown in Figure 2. The Fortran anchor, FKB/F, can use models originally constructed and trained in Keras, which can then be transferred to Fortran via FKB/P. To use these models, the Fortran side of FKB implements a neural network library. This portion of FKB can be used within large-scale scientific computation software, such as the projects identified in Section 2.

By leveraging FKB, it becomes seamless to train networks in Python and transfer them to Fortran, to run inside large-scale simulations. Similarly, neural network models constructed in Fortran can be transferred to Python for additional analysis, expansion, and optimization-including hyperparameter searches using available tools in Python [20, 49, 50]. As both sides of the bridge have been properly introduced, the following section will describe the specific features and functionalities of FKB.

5. Features of FKB

Once a neural network is trained in high-level APIs such as Keras, the practitioner has few practical avenues for using this model in Fortran-based projects. One approach maybe to hard code network operations inside Fortran while manually moving parameters from the Keras model. Several examples of this can been seen in climate modeling [41, 51–53].

To provide one specific example, in [41], the authors trained a deep neural network (DNN) to represent subgrid cloud and convective energy transport processes, in Keras. To assess its credibility, they needed to test the DNN's two-way interactions when thousands of replicates of it were embedded within a coarse-resolution global atmospheric model, written in Fortran neural network-emulated clouds interacting with deterministic physical calculations of planetary geophysical fluid dynamics. As the global atmospheric simulator does not offer native neural network support, the authors hardcoded their DNN model into the global simulation software framework. This approach has obvious disadvantages. Every minor change made to the model in Keras requires rewriting the Fortran code. If one wishes to test a suite of models in Fortran, this approach becomes untenable.

As each network may require different hyperparameters and, as a result, necessitates rewriting and compiling the

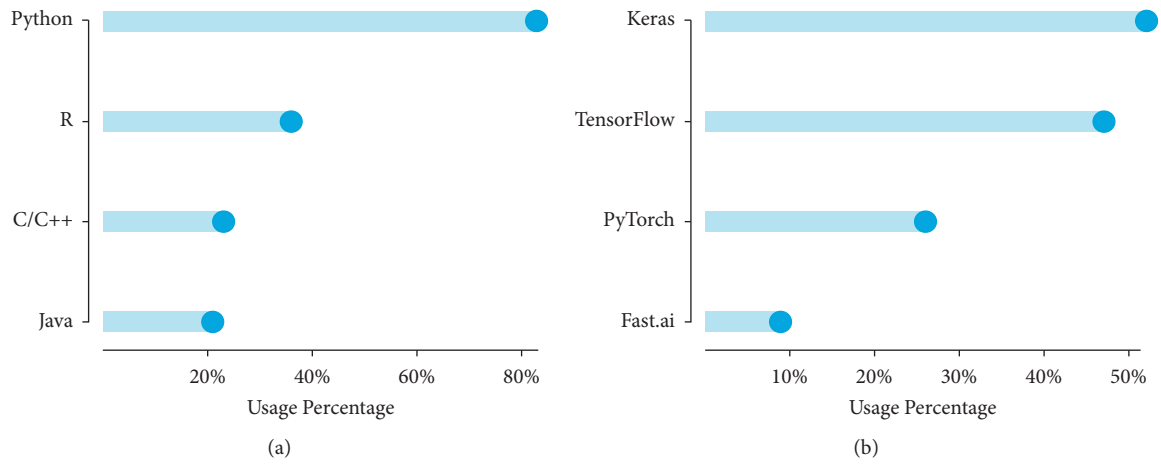

Figure 1: (a) Usage of programming languages for machine learning and data science. Statistics are from the 2018 Kaggle ML and DS survey [40]. (b) Usage metrics of deep learning frameworks. Statistics are from the 2019 Kaggle State of Data Science and Machine Learning Report [42].

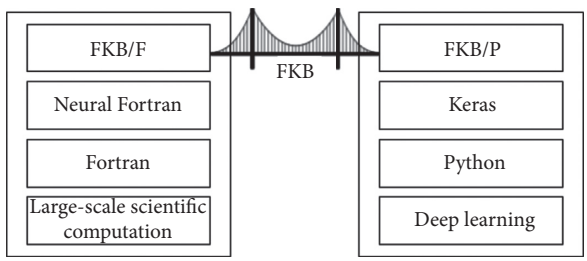

Figure 2: Positioning of FKB within Fortran and Python ecosystems.

Fortran code for every new model. This process drastically limits the breadth of available models to be tested within the simulator. This bottleneck is currently a significant roadblock to ongoing debates in the climate simulation community, more broadly, about whether or not to use DNN representations of subgrid physics in next-generation climate modeling. Insufficient testing of diverse candidate neural networks (NN) means that little is known about how minor imperfections in the fit of one NN can amplify when the NN is coupled to fluid dynamics, which is just beginning to be explored [54].

These issues demand a solution, in the form of a bridge between Keras and Fortran. The FKB software solves these issues via two key elements. First, it provides a neural network library implemented in Fortran (FKB/F). Second, it offers the ability to parse existing Keras models into formats consistent with the Fortran neural network library (FKB/P). As a result, users can switch, seamlessly, back and forth between Python and Fortran. This context provides a way for iterative neural network tuning (Python) and testing (Fortran), with a simple way to translate between the two software environments. Additionally, FKB offers currently unavailable Fortran specific features for neural networks. It will be useful to highlight those new features while documenting the format to which FKB adheres. The following sections describe the Python and Fortran anchors' features, FKB/P and FKB/F, respectively.

5.1. FKB/P. Keras models—once built, trained, and saved—are stored in Hierarchical Data Format 5 (HDF5) files. These files contain the network architecture, weights, biases, and additional information-optimizers, learning rates, gradients, etc. From the HDF5 file, FKB/P parses the network architecture, extracting the number of layers, activation functions, nodes per layer, and all weights and biases. This information is converted to match the Fortran neural network configuration in FKB/F. This allows users to build an equivalent network in Fortran, which can easily be loaded and used within a Fortran environment. If any modifications to the model are made inside Fortran, FKB/P will parse this back into the equivalent HDF5 file to be used in Keras once again.

On the contrary, networks may be initially constructed in Fortran. After initial training and testing, a user can switch to Keras for further evaluation. From Keras, users can conduct additional testing or hyperparameter tuning where these tools are readily available [49].

The ability to seamlessly pass neural network architectures between Python and Fortran is essential for any practitioner working in this space. This bridge allows users to take advantage of the high-level Keras API—training on computationally efficient GPUs—then to insert their trained model into a Fortran codebase. The functionality provided bridges the chasm between Keras and Fortran.

5.2. FKB/F. The Fortran anchor of FKB leverages and extends the original Neural Fortran library. Below, we introduce newly implemented features to make Neural Fortran more flexible and able to communicate on the two-way bridge.

5.2.1. Custom Layers. To implement neural networks in Fortran, FKB leverages and extends the Neural Fortran library [46]. The prototype Neural Fortran library format that we build on was only capable of implementing a fully connected layer. Forward and backward operations occurred

outside this layer in the network module. An example of this is shown in Algorithm 1. From the algorithm, one can observe hardcoded matrix multiplication of layer weights, the addition of biases, and the activation functions inside the network module. This network-level subroutine accesses and modifies individual layer attributes. This rigid format is inconsistent with modern neural network implementation paradigms [17–19], but it makes it impossible to implement other layers or custom operations. To increase the library's flexibility, operations must be encapsulated inside the layer, consistent with current practice.

In FKB, we introduce an extendable layer type module (Algorithm 2). To implement a layer, one simply extends the layer type and specifies the construction of the forward and backward functions. Adhering to this format offers several advantages. By restructuring the format of the library, we offer the ability to implement arbitrary layers. Additionally, in the network module, all layers are stored in an array of pointers. This leads to the encapsulated version shown in Algorithm 2 wherein a forward pass, in the network module, calls the layer-specific forward function. In this way, all operations are confined to the layer module, and the output from one layer is passed as input to the next.

FKB supports fully connected or dense layers, dropout [55, 56], and batch normalization [57]. Algorithm 3 is an example of extending the layer_type to implement a batch normalization layer. This format translates to increased functionality and customizability to the user. As a result, more standard layers from Keras are available, while giving users the flexibility to implement their own custom operations.

5.2.2. Training in Fortran. It is necessary to distinguish between the terms *offline* versus *online* for the following section. These terms serve to distinguish two different settings in which a neural network can be used in a Fortran computing package. Both settings can make use of historical or simulated data to train an artificial network. The distinguishing feature is how the predictions of a model are used. In an online setting, predictions from the model are used to evolve a physical process. The predictions at one time step affect how the system acts at the following time step. As a result, inputs to the model will change based on how the model acted in the past. In offline settings, this is not the case. Predictions made in the past do not affect the input to the model in the future.

In many cases, offline training may be sufficient to learn a model, if enough prior data is available. However, in some cases, online training may be the method of choice. To this end, FKB is equipped to handle backpropagation for gradient descent optimization of a specified cost function.

The layer encapsulation mentioned above of forward and backward operations (Section 5.2.1) becomes extremely valuable in training. Instead of all computations occurring within the network module [46], they are contained in layer-specific functions. Much like the forward pass, backward operations occur in the layer. In this fashion, each layer is responsible for computing its gradients with respect to its parameters and returning the gradient with respect to the layer below it.

Online training can serve a variety of purposes. First, a neural network model may be learned entirely in Fortran, based on the evolving state variables during the integration of a physical dynamical system simulation, and then transferred to Keras after the fact. In this setting, the ground truth, from the simulator, is passed to the network for it to calculate its errors and update its parameters accordingly through backpropagation. Second, online training could serve to provide gentle corrections to an imperfect pretrained model, for instance, to hedge against the amplification of its imperfections that are only revealed once the NN is coupled to other physical calculations. Here, a model is trained offline in Keras and transferred to Fortran (Section 5.1). In some cases, for a variety of reasons, the offline training data may have a differing distribution than that of the online data. In such a setting, it proves beneficial to offer slight corrections to the network. Finally, a secondary model may be constructed to learn and compensate for the deficiencies in the primary model. In this way, the two networks work together to balance out any instability issues.

The ease of use and proper format directly results from the encapsulation of layer operations. Online training offers a solution to tackle a suite of potential problems. As a result, models may be updated with slight corrections or learned entirely online.

5.2.3. Custom Loss Functions. In many applications, practitioners may wish to optimize a unique quantity, a function other than a mean squared error or crossentropy. This is common when target variables interact or additional information is known about their relationship in a desired application. For example, in modeling any physical system, predictions from a neural network must not violate physical constraints, energy cannot be created or destroyed in the system. To satisfy this restriction, a loss function can be written to quantify the amount of violation of physical properties. This construction can then be minimized to alleviate constraint infractions [13].

The implementation of custom loss functions is standard for high-level APIs such as Keras, TensorFlow, and PyTorch to provide this ability in their codebase [17–19]. As FKB is designed for those working in the physical sciences where environmental, physical, or application-specific constraints are common, it provides the ability to implement custom loss functions. To take advantage of this functionality, users must implement their desired loss function, just as they would in Keras. As FKB does not provide automatic differentiation, the derivatives with respect to the input are also required for training. Once these functions have been specified, they can be dropped into the existing framework and run normally, much like Keras.

This capability is demonstrated through the implementation of the crossentropy loss function in Algorithm 4. To implement this previously unavailable loss function, we first declare two functions. First, the crossentropy scalar loss. Second, the loss with respect to the input logits is derived.

```
pure subroutine fwdprop(self, x)
   ! Performs the forward propagation and stores arguments to activation
   ! functions and activations themselves for use in backprop.
   class(network_type), intent(in out): self
   real(rk), intent(in): x()
   integer(ik): n
   associate(layers => self % layers)
     layers(1) % a = x
     do n = 2, size(layers)
       layers(n) % z = matmul(transpose(layers(n-1) % w), layers(n-1) % a) + layers(n) % b
       layers(n) % a = self % layers(n) % activation(layers(n) % z)
     end do
   end associate
 end subroutine fwdprop
```

Algorithm 1: Original code from [46]. Layer operations occur inside the network module, limiting flexibility.

```
function output(self, input) result(last_layer_output)
  ...
  ! iterate through layers passing activation forward
  do n = 1, size(layers)
    call layers(n) % p % forward(layers(n-1) % p % o)
   end do
  ! get output from last layer
  last_layer_output = layers(size(layers)) % p % o
end function output
```

Algorithm 2: Forward pass in the FKB network module. Each layer simply calls its own forward function. The technical operations occur within each layer.

```
! BatchNorm layer-extends from base layer_type
! Implements batch normalization
type, extends(layer_type): BatchNorm
  ! epsilon parameter
  real(rk): epsilon
contains
  procedure, public, pass(self): forward => batchnorm_forward
  procedure, public, pass(self): backward => batchnorm_backward
end type BatchNorm
```

Algorithm 3: Example of extending the layer_type to implement batch normalization.

These two functions are then referenced as the loss and d_loss, respectively. By providing this functionality, users may leverage a variety of loss functions that can be used to minimize application-specific quantities. Once described, they may be included with the existing framework and used during online training.

5.2.4. Ensembles. Ensembles consist of different models, each trained on the same, or bootstrapped, data. The output of the ensemble will be an average of all its member's predictions. In machine learning, ensembles of models typically perform better than any one of its members alone. The ensemble strategy exploits the fact that each model will make different errors. Therefore, when averaged together, these predictions become more accurate, as certain errors get smoothed out. A consensus from machine learning practitioners is ensembling and gives 1-2% improvement in performance [58].

As a result of this averaging, ensembles provide a boost in performance as well as additional robustness. In domains, where physical constraint violations yield stability issues,

```
real(rk) function crossentropy_loss(self, y_true, y_pred)
  ! Given predicted and expected output, returns the scalar loss
  class(network_type), intent(in out): self
  real(rk), intent(in): y_true(), y_pred()
  loss = - sum(y_true * log(y_pred))
end function loss
function d_crossentropy_loss(self, y_true, y_pred) result(loss)
  ! Given predicted and expected output
  ! returns the loss with respect to softmax input
  class(network_type), intent(in out): self
  real(rk), intent(in): y_true(), y_pred()
  real(rk), allocatable: loss()
  loss = y_pred - y_true
end function d_loss
```

Algorithm 4: Implementation of crossentropy loss function and the corresponding derivation with respect to the input logits.

ensembles may be applied to dampen these problems. By averaging across many networks, the instability of any one model will be drastically reduced in the presence of more sound predictions.

The functionality provided requires the user to specify a directory that contains the models of interest and a desired amount of noise. The ensemble type will read in each model and construct a network corresponding to each of them. To get a prediction from the ensemble, an input vector is passed to it. For nonzero amounts of noise, Gaussian noise is applied to the input vector each time it is passed to an ensemble member. This allows each member to see a slightly different variant of the input, increasing the robustness of prediction around that point. This operation runs in parallel using OpenMP, where each network can be given its thread to expedite computation; such an approach could easily be adapted via OpenACC for GPU-based threading of large ensemble network calculations. Following the computation, the predictions are averaged together, and the final output is given.

6. Case Study

The following section provides a case study demonstrating an application of FKB to experimental next-generation climate modeling. The Superparameterized Community Atmospheric Model version 3.0 (SPCAM3) is used for all simulations in this study. Superparameterization is an approach that confronts the decade-long problem of representing subgrid cloud physics in climate models by embedding thousands of limited-domain explicit submodels of moist convection within a conventional planetary-scale model of the large-scale atmosphere [59–62]. This approach tends to involve two orders of magnitude more computational intensity per unit area of the simulated earth, but recently Rasp et al. used a deep neural network to emulate all of the expensive subgrid cloud resolving models' (CRM) influence on the planetary host at drastically reduced computational expense [41]. This study, along with others in the emerging climate modeling literature [51] have demonstrated the potential advantages of a data-driven approach for addressing the critical unresolved effects of clouds and convection on planetary climate, as compared to previous, heuristic-based, approximations to subgrid physics. However, the idea of emulating turbulence in climate simulation is still an emerging one, with unclear trade-offs, including frequent instabilities when NN emulators are coupled with fluid dynamics, which the community is seeking to learn how to control [51]. It has even been questioned whether the offline skill of such emulators, during their training, is predictive of their online performance [63, 64], an important open question.

These questions are understudied primarily due to the lack of the simple software interface that FKB now enables for climate scientists to test diverse candidate neural networks and ensembles within planetary climate models.

To illustrate an advance on this front, we now apply FKB to shed new light on two related questions currently in debate:

(1) Does offline performance translate to online model performance [63, 64]?

(2) Which neural network hyperparameters most affect online performance?

Using FKB, the study can be broken into two stages. First, a suite of 108 candidate neural network models of convection are trained, via Keras, on simulated data from the SPCAM3. Second, the models are converted to Fortran and run online (i.e., coupled to planetary fluid dynamics) in the SPCAM3 simulator. The number of steps serves as a preliminary metric of performance until catastrophic failure. It is clear that, in the absence of the FKB library, running hundreds of candidate neural network submodels of convection within the Fortran-based model of the rest of the planet's atmosphere would be nearly impossible. As each network contains various hyperparameters, each with different weights and biases learned during training, including layer-specific properties such as optional use of dropout or batch normalization. To leverage the FKB library with SPCAM3, we simply compile the neural network library in advance and link it to the compilation of SPCAM3. Documentation steps for the implementation of this case study

are provided in https://github.com/scientific-computing/FKB/blob/master/SPCAM_Instructions.md.

The input to this neural network model is a 94-dimensional vector. Features include vertically resolved vectors representing the large-scale (host model) temperature, humidity, meridional wind vertical structure, surface pressure, incoming solar radiation, sensible heat flux, and latent heat flux scalars. The output of the network is a 65-dimensional vector composed of the embedded models' influence on their host, i.e., the sum of the CRM and radiative heating rates, the CRM moistening rate, the net radiative fluxes at the top of the atmosphere and surface of the Earth, and the precipitation.

The training data used here are challenging to fit, as they come from an enhanced version of the CRM training data that was originally studied by [41]. In superparameterized simulations, one can control the degrees of freedom of the interior resolved scale through the room available for interesting forms of subgrid storm organization to form. One can control the physical extent (i.e., number of columns used in) each embedded CRM array [65]. In [41], CRM arrays with only 8 columns (32-km extent, given the 4-km horizontal resolution) were used. Here, we quadruple the extent (from 32 km to 128 km, i.e., from 8 columns to 32 columns) to improve its physical realism. Despite several attempts, these data have never been fit successfully. NNs trained from the enriched data tend to produce crashes within just a few simulated weeks after they are embedded in the climate model (see discussion of "NN-unstable" by [54], for details).

Our working hypothesis is that historical failures in free-running tests when emulators are trained on higher quality CRM training data reflect a broader issue of insufficient hyperparameter tuning in climate model applications. To address this, we conducted neural network optimization via a random search using SHERPA [49], a Python library for hyperparameter tuning. We detail the hyperparameters of interest in Table 1, as well as the range of available options during the search. The hyperparameters of interest consisted of whether or not to use batch normalization, the amount of dropout, the leaky ReLU coefficient, learning rate, nodes per layer, the number of layers, and the optimizer. The random search algorithm has the advantage of making no assumptions about the structure of the hyperparameter search problem and is ideal for exploring a variety of settings.

We attained 108 candidate neural network model configurations, each trained for 25 epochs with early stopping monitoring the validation loss. Following the offline training stage, the neural network models were converted into their Fortran counterparts and ran inside SPCAM3. We underscore that this critical step would have been prohibitive using standard tools that have required manual translation of each candidate model. However, by leveraging the FKB library, each model was loaded independently into Fortran and run as the subgrid physics emulator inside SPCAM3's host planetary model, of the large-scale atmospheric state. Each model was coupled to fluid dynamics, to run a wide ensemble of prognostic tests across an unprecedented diversity of candidate neural network architectures. Each of the one hundred and eight candidate neural network models—with their various numbers of layers, layer-specific settings (batch normalization, relu magnitude, etc), nodes per layer, weights, and biases—were run online, all without rewriting any Fortran code.

Table 1: Hyperparameter space.

Name	Options	Parameter type
Batch normalization	[yes, no]	Choice
Dropout	[0, 0.25]	Continuous
Leaky ReLU coefficient	[0–0.4]	Continuous
Learning rate	[0.00001–0.01]	Continuous (log)
Nodes per layer	[125, 256, 512]	Discrete
Number of layers	[4–11]	Discrete
Optimizer	[Adam, RMSProp, SGD]	Choice

In order to address the first question and evaluate a neural network model's performance, we compare its validation MSE during training with the time-to-failure of the online tests in which 8,192 instances of the NN, spaced at regular intervals around the globe, are coupled interactively to their host global atmospheric model of large-scale geophysical fluid dynamics. This yields Figure 3, which sheds new light on the offline vs. online relationship.

The results in this figure demonstrate a relationship between offline validation error and online performance. There is a distinct, negative, relationship between offline MSE and online stability (Spearman correlation of −0.73; $p = 4.961e^{-19}$). Intriguingly, the mean-squared error loss of our multilayer perceptron is a reasonable predictor of stability once coupled to the climate model, insofar as the time-to-failure is concerned. This finding is interesting in the context of the recent speculation by [64] that such a relationship might not exist using similar NNs in a similar setting, as well as the comments by [63] about similar incongruities even in reduced-order dynamical systems when emulated with GANs.

Of course, stability alone is a necessary but not a sufficient condition of prognostic success, which also requires an in-depth analysis of biases in the simulated climate. Figure 4 shows the time evolution of the tropospheric temperature and humidity biases, colorized by the offline validation error. These metrics reveal that, although our search has uncovered many runs that are "stable,"can run without catastrophically crashing for several months, most of these runs would not be very useful in an operational setting. Almost all NNs exhibit major errors in the simulated climate, having drifted to erroneous attractors with root mean square errors in temperature frequently above 10 K. However, the NN that produced the best offline validation error stands out as having the combined desired qualities of stability and skill with temperature biases of less than 2 K, competitive with [41]. Interestingly, coupling instead to the ensemble mean of a few of the best-ranked models (magenta dashed lines) does not outperform coupling to the best fit

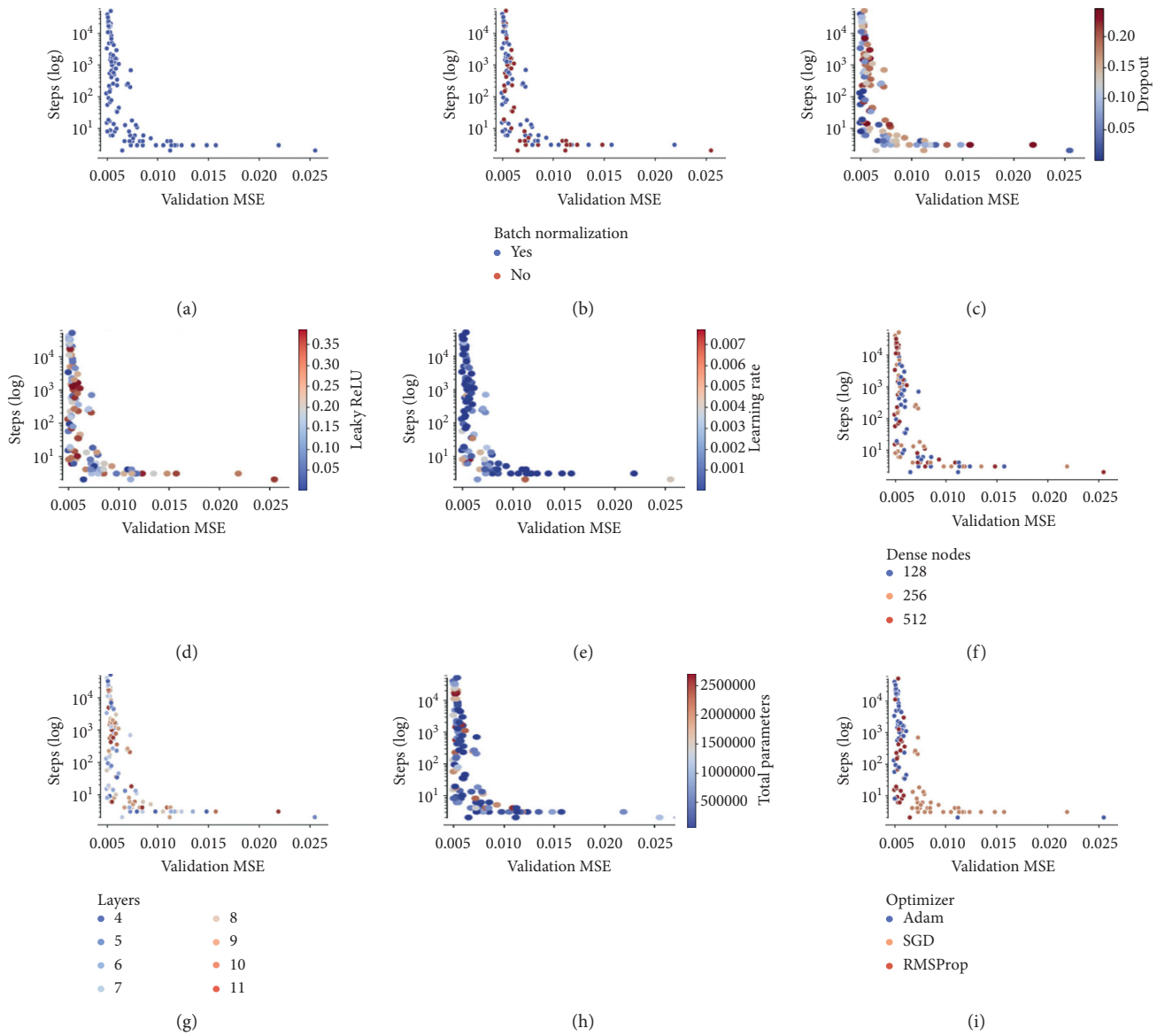

Figure 3: Offline performance-validation mean squared error (MSE) vs. online performance number of steps until crash. (a) All models. (b) By batch normalization usage. (c) By dropout amount. (d) By leaky ReLU coefficient. (e) By learning rate. (f) By number of dense nodes per layer. (g) By number of layers. (h) By total number of model parameters. (i) By optimizer type.

model, the value of having found it using SHERPA (Figure 4).

In short, we have produced a successful coupled simulation that was particularly challenging without formal hyperparameter tuning and FKB. This result suggests that sufficient hyperparameter tuning may be critical to solving chronic instability in climate model applications of DNNs for subgrid physics.

The second question naturally arises as to which of the hyperparameters are most impactful to the online performance. To assess this, Figures 4(b)–4(i) decompose the sensitivity of the baseline relationship to individual hyperparameter choices. The choice of the optimizer is shown to correlate most strongly with online performance (Figure 3). This finding is confirmed by Spearman values, as shown in Table 2. The optimizer hyperparameter has the largest absolute correlation value with online performance. No other hyperparameter shows as clear a distinction in correlation that is evident in the choice of the optimizer, including the network depth and total number of parameters, which are known to be important to offline fits for this problem [66], but are surprisingly not as predictive of coupled skill as the choice of the optimizer, whose impact has not previously been isolated (for this application).

Further investigation into the specific optimizer used reveals the SGD optimizer to perform poorly; NNs fit with SGD never run longer than 1,000 steps when coupled online (Figure 3). Again the visual intuition from Figure 3 is confirmed by Spearman correlation values. SGD, Adam, and RMSProp have Spearman values of -0.6670, 0.5936, 0.0586, respectively. These values demonstrate that the use of SGD is negatively correlated with online performance, whereas

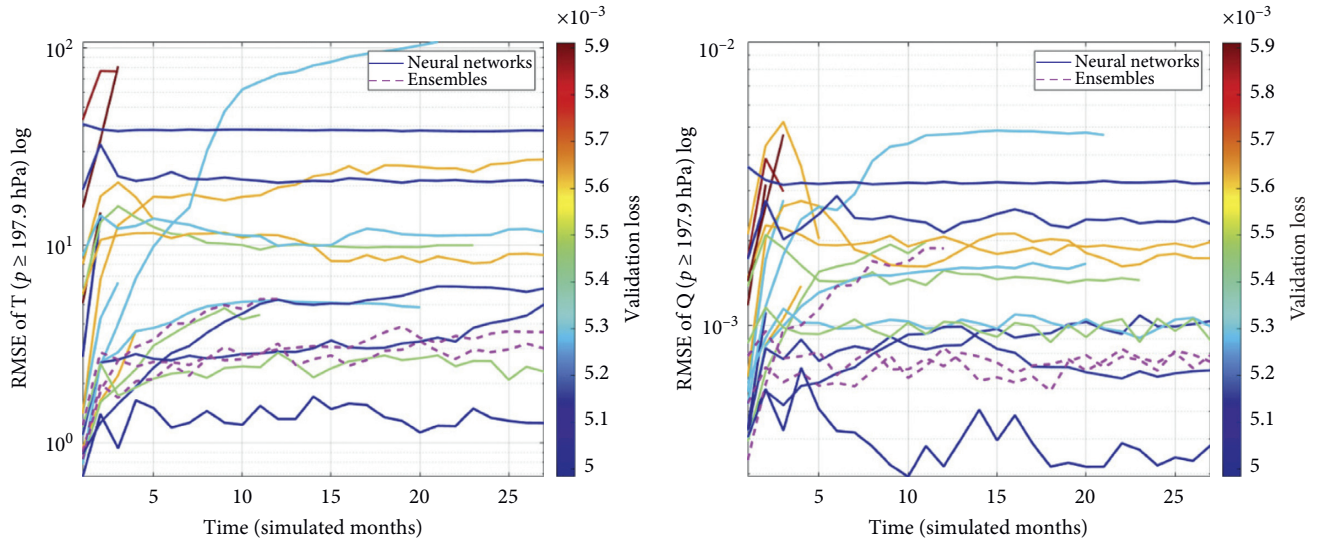

Figure 4: The time-evolution of the tropospheric (a) temperature and (b) humidity biases, colorized by the offline validation error.

Table 2: Spearman correlation of corresponding hyperparameter with online performance and associated p value.

	Correlation	p-value
BatchNorm	0.0859	$3.7896e{-01}$
Dropout	0.1919	$4.7591e{-02}$
Leaky ReLU	0.0055	$9.5465e{-01}$
Learning rate	−0.2087	$3.0923e{-02}$
Dense nodes	0.1427	$1.4249e{-01}$
Layers	0.0410	$6.7491e{-01}$
Optimizer	−0.6998	$5.0177e{-17}$
Parameters	0.1528	$1.1609e{-01}$

Adam positively correlates with online performance. This result leads one to speculate that increased improvements in online skill may be realized from more advanced optimizers with enhanced gradient update schedules.

Finally, after answering the two questions motivating this case study, we can compare the results of the best performing model with that of previously published models of [41] when applied to the challenging limit of CRMs with 32-km horizontal extent. The model proposed by Rasp et al. was a single deep neural network. The hyperparameter space of this model was not fully explored online in large part due to the laborious process required to transfer those models into Fortran. The Rasp et al. model (provided by the authors) ran for 128 steps before crashing due to instability issues. The five best models achieved in this study ran to completion of a 5-year simulation, i.e., for 87,840 steps; of these, two of the five models further exhibited root mean square errors in simulated tropospheric temperature of less than 2 degrees Celsius. This dramatic improvement in stability is a direct result of the ease with which a wide variety of models (identified by SHERPA) can be transferred between Python and Fortran (thanks to FKB). We also note that this method is preferable to another approach that was recently proposed to begin stabilizing the same model, through small-amplitude Gaussian input perturbation [54], a strategy that, while promising, adds computational expense and introduces out-of-sample extrapolation issues that can be avoided with the brute-force optimization and wide-ensemble prognostic testing path to stabilization we have outlined here.

This case study has investigated two closely entangled questions: (1) does offline performance correspond to online model performance? and (2) what neural network hyperparameters most affect online performance? Both of these questions have been answered by leveraging the FKB library. The library offers the ability to expeditiously transfer models trained in Keras to Fortran, where they may be run online in existing simulators. In the absence of FKB, neither one of these questions could be approached without unreasonable human intervention, as the operational target is a climate model with over a hundred thousand lines of code written in Fortran.

7. Conclusion

The ubiquitousness of deep learning has resulted from extensive free and open-source libraries [17, 46, 58]. Deep learning's success and popularity merit its integration in large-scale computing packages, such as those written in Fortran. Instead of rewriting all existing libraries in Fortran, we introduced a two-way bridge between low-level, Fortran, and Python through the FKB Library. The library provides researchers the ability to implement neural networks into Fortran code bases while being able to transfer them back and forth with Keras.

Fortran, which has been a staple within computationally intensive fields for decades, will undoubtedly see continued use due to its fast computational ability and vast amounts of

legacy code. The FKB library enables users to access many features of the Keras API directly in Fortran, including the ability to create custom layers and loss functions to suit their needs. We demonstrate the integrability of FKB through our case study involving the SPCAM3 simulator. An advantage of FKB is its ease of use, demonstrated by its ability to be compiled in advance and once linked can be easily leveraged in existing large-scale simulators, as we have illustrated for the application of multiscale physical simulations of the global atmosphere.

Data Availability

Code is made publicly available in https://github.com/scientific-computing/FKB. Steps documenting the case study are documented in https://github.com/scientific-computing/FKB/blob/master/SPCAM_Instructions.md.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

The work of JO and PB is supported by NSF NRT (Grant 1633631). MP acknowledges NSF funding from OAC-1835863 and AGS-1734164. This research also used HPC resources of the Extreme Science and Engineering Discovery Environment (XSEDE), which was supported by the National Science Foundation under Grant no. ACI-1548562 [67] and allocation number TG-ATM190002.

References

[1] "FORTRAN," March 2011, https://www.ibm.com/ibm/history/ibm100/us/en/icons/fortran/.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in Neural Information Processing Systems*, pp. 1097–1105, 2012.

[3] J. Schmidhuber, "Deep learning in neural networks: an overview," *Neural Networks*, vol. 61, pp. 85–117, 2015.

[4] N. LaHaye, "Multi-modal object tracking and image fusion with unsupervised DeepLearning," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 12, pp. 3056–3066, 2019.

[5] X. X. Zhu, "Deep learning in remote sensing: a comprehensive review and list ofresources," *IEEE Geoscience and Remote Sensing Magazine*, vol. 5, pp. 8–36, 2017.

[6] J. Ott, A. Atchison, and J. Erik, "Exploring the applicability of low-shot learning in mining software repositories," *Journal of Big Data*, vol. 6, p. 35, 2019.

[7] J. Ott, "A deep learning approach to identifying source code in images and video," in *Proceedings of the 2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR)*, pp. 376–386, Gothenburg, Sweden, May 2018.

[8] J. Ott, "Learning lexical features of programming languages from imagery using con-volutional neural networks," in *Proceedings of the 26th Conference on Program Comprehension*, pp. 336–339, Gothenburg, Sweden, May 2018.

[9] J. Tompson, M. Stein, Y. Lecun, and K. Perlin, "Real-time continuous pose recovery of human hands using convolutional networks," *ACM Transactions on Graphics (ToG)*, vol. 33, p. 169, 2014.

[10] G. Urban, "Deep learning achieves near human-level polyp detection in screening colonoscopy," *Gastroenterology*, vol. 155, 2018.

[11] F. Agostinelli, S. McAleer, A. Shmakov, and P. Baldi, "Solving the Rubik's cube with deep reinforcement learning and search," *Nature Machine Intelligence*, vol. 1, no. 8, pp. 356–363, 2019.

[12] D. Silver, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, 2016.

[13] T. Beucler, "Enforcing analytic constraints in neural-networks emulating physical systems," 2020, http://arxiv.org/abs/1909.00912.

[14] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational Physics*, vol. 378, pp. 686–707, 2019.

[15] Y. Bar-Sinai, S. Hoyer, J. Hickey, and M. P. Brenner, "Learning data-driven discretizations for partial differential equations," *Proceedings of the National Academy of Sciences*, vol. 116, no. 31, pp. 15344–15349, 2019.

[16] S. H. Rudy, "Data-driven discovery of partial differential equations," *Science Advances*, vol. 3, 2017.

[17] M. Abadi, "Tensorflow: a system for large-scale machine learning," in *Proceedings of the 12th Symposium on Operating Systems Design and Implementation 2016*, pp. 265–283, Savannah, GA, USA, November 2016.

[18] F. Chollet, "Keras," 2015, https://github.com/fchollet/keras.

[19] P. Adam, *Automatic Differentiation in Pytorch*University of Warsaw, Warszawa, Poland, 2017.

[20] B. James, D. Yamins, and D. D. Cox, "Hyperopt: a python library for optimizing the hyperparameters of machine learning algorithms," in *Proceedings of the 12th Python in Science Conference*, pp. 13–20, Austin, TX, USA, June 2013.

[21] J. W. Hurrell, "The community earth system model: a framework for collaborative re-search," *Bulletin of the American Meteorological Society*, vol. 94, pp. 1339–1360, 2013.

[22] J.-C. Golaz, "The DOE E3SM coupled model version 1: overview and evaluation at standard resolution," *Journal of Advances in Modeling Earth Systems*, vol. 11, pp. 2089–2129, 2019.

[23] I. M. Held, "Structure and performance of GFDL's CM4. 0 climate model," *Journal of Advances in Modeling Earth Systems*, vol. 11, pp. 3691–3727, 2019.

[24] NEMO System Team, "NEMO ocean engine," *Scientific Notes of Climate Modelling Center*, vol. 27, 2019.

[25] A. J. Wallcraft, H. Hurlburt, E. J. Metzger, E. Chassignet, J. Cummings, and O. M. Smedstad, "Global ocean prediction using HYCOM," in *2007 DoD High Performance Computing Modernization Program Users Group Conference*, pp. 259–262, Pittsburgh, PA, USA, June 2007.

[26] M. A. Donelan, "Modeling waves and wind stress," *Journal of Geophysical Research: Oceans*, vol. 117, 2012.

[27] J. G. Powers, "The weather research and forecasting model: overview, system efforts, andFuture directions," *Bulletin of the American Meteorological Society*, vol. 98, pp. 1717–1737, 2017.

[28] E. Madenci and I. Guven, *The Finite Element Method and Applications in Engineering using ANSYSR©*, Springer, Berlin, Germany, 2015.

[29] L. B. orgesson, "Abaqus," in *Developments in Geotechnical Engineering*, vol. 79, pp. 565–570, Elsevier, Amsterdam, Netherlands, 1996.

[30] Y. D. Murray, "Users manual for LS-DYNA concrete material model 159," Federal Highway Administration, Washington, DC, USA, Tech. Rep, 2007.

[31] D. Komatitsch, *SPECFEM3D Cartesian v2.0.2*, Computational Infrastructure for Geodynamics, Davis, CA, USA, 2012.

[32] F. Archambeau, N. Mechitoua, and M. Sakiz, "Code saturne: a finite volume codefor the computation of turbulent incompressible flows-industrial applications," *International Journal on Finite Volumes 1*, vol. 1, 2004.

[33] J. Fischer and S. G. Kerkemeier, "nek5000 Web Page," 2008, http://nek5000.mcs.anl.gov.

[34] B. Brooks, R. E. Bruccoleri, B. D. Olafson, D. J. States, S. Swaminathan, and M. Karplus, "CHARMM: a program for macromolecular energy, minimization, and dynamics calculations," *Journal of Computational Chemistry*, vol. 4, pp. 187–217, 1983.

[35] M. Valiev, E. J. Bylaska, N. Govind et al., "NWChem: a comprehensive and scalable open-source solution for large scale molecular simulations," *Computer Physics Communications*, vol. 181, no. 9, pp. 1477–1489, 2010.

[36] A. Ferrari, P. R. Sala, A. Fasso, and J. Ranft, "FLUKA: a multi-particle transport code," Technical Report SLAC-R-773, Stanford University, Stanford, CA, USA, 2005.

[37] G.-Q. Jiang, J. Xu, and J. Wei, "A deep learning algorithm of neural network forthe parameterization of typhoon-ocean feedback in typhoon forecast models," *Geophysical Research Letters*, vol. 45, pp. 3706–3716, 2018.

[38] J. Ling, A. Kurzawski, and J. Templeton, "Reynolds averaged turbulence modelling using deep neural networks with embedded invariance," *Journal of Fluid Mechanics*, vol. 807, pp. 155–166, 2016.

[39] H. R. Vega-Carrillo, "Artificial neural networks in neutron dosimetry," *Radiation Protection Dosimetry*, vol. 118, pp. 251–259, 2005.

[40] Kaggle, "Kaggle ML & DS Survey," 2018, https://www.kaggle.com/kaggle/kaggle-survey-2018.14.

[41] S. Rasp, M. S. Pritchard, and P. Gentine, "Deep learning to represent subgrid processes in climate models," *Proceedings of the National Academy of Sciences*, vol. 115, pp. 9684–9689, 2018.

[42] "Kaggle State of Data Science and Machine Learning," 2019, https://www.docdroid.net/qzyxCr4/kaggle-state-of-data-science-and-machine-learning-2019.pdf.

[43] J. Bernal, *NEURBT: A Program for Computing Neural Networks for Classification using Batch Learning*, Technical Report 8037, National Institute of Standards and Technology, Gaithersburg, ML, USA, 2015.

[44] J. Bernal and J. Torres-Jimenez, "SAGRAD: a program for neural network training withSimulated annealing and the conjugate gradient method," *Journal of Research of the National Institute of Standards and Technology*, vol. 120, p. 113, 2015.

[45] P. Brierley, "Fortran90 MLP backprop code," http://www.philbrierley.com/phil.html.

[46] M. Curcic, "A parallel Fortran framework for neural networks and deep learning," *ACMSIGPLAN Fortran Forum*, vol. 38, pp. 4–21, 2019.

[47] S. Nissen, *Implementation of a Fast Artificial Neural Network Library (Fann)*Addison-Wesley, Boston, MA, USA, Dec.2003.

[48] D. J. Lary, M. D. Müller, and H. Y. Mussa, "Using neural networks to describe tracer correlations," *Atmospheric Chemistry and Physics*, vol. 4, pp. 143–146, 2004.

[49] L. Hertel, "Sherpa: robust hyperparameter optimization for machine learning," *SoftwareX*, https://arxiv.org/pdf/2005.04048.pdf, 2020.

[50] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," *Advances in Neural Information Processing Systems*, vol. 1, pp. 2951–2959, 2012.

[51] N. D. Brenowitz and C. S. Bretherton, "Prognostic validation of a neural networkunified physics parameterization," *Geophysical Research Letters*, vol. 45, pp. 6289–6298, 2018.

[52] D. John Gagne, C.-C. Chen, and A. Gettelman, "Emulation of bin Microphysical Processes with machine learning," in *Proceedings of the 100th American Meteorological Society Annual Meeting*, Boston, MA, USA, January 2020.

[53] D. John Gagne, "Machine learning parameterization of the surface layer: bridging the observation-modeling gap," in *Proceedings of the AGU's Fall Meeting*, San Francisco, CA, USA, 2019.

[54] N. D. Brenowitz, "Interpreting and stabilizing machine-learning parametrizations of con-vection," 2020, http://arxiv.org/abs/2003.06549.

[55] P. Baldi and P. Sadowski, "The dropout learning algorithm," *Artificial Intelligence*, vol. 210, pp. 78–122, 2014.

[56] N. Srivastava, "Dropout: a simple way to prevent neural networks from over fitting," *The Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

[57] Sergey Ioffe and C. Szegedy, "Batch normalization: accelerating deep network training by reducing internal covariate shift," 2015, http://arxiv.org/abs/1502.03167.

[58] F. Chollet, *Deep Learning mit Python und Keras: Das Praxis-Handbuch vom Entwickler derKeras-Bibliothek*, MITP-Verlags, Bonn, Germany, 2018.

[59] W. W. Grabowski, "Coupling cloud processes with the large-scale dynamics using the cloud-resolving convection parameterization (CRCP)," *Journal of the Atmospheric Sciences*, vol. 58, no. 9, pp. 978–997, 2001.

[60] M. Khairoutdinov, C. DeMott, and D. Randall, "Evaluation of the simulated interannual and subseasonal variability in an AMIP-style simulation using the CSU multiscale modeling framework," *Journal of Climate*, vol. 21, no. 3, pp. 413–431, 2008.

[61] M. Khairoutdinov, D. Randall, and C. DeMott, "Simulations of the atmospheric general circulation using a cloud-resolving model as a superparameterization of physical processes," *Journal of the Atmospheric Sciences*, vol. 62, no. 7, pp. 2136–2154, 2005.

[62] K. Thayer-Calder and D. A. Randall, "The role of convective moistening in the madden-julian oscillation," *Journal of the Atmospheric Sciences*, vol. 66, no. 11, pp. 3297–3312, 2009.

[63] D. John Gagne II, H. M. Christensen, A. C. Subramanian, and A. H. Monahan, "Machine learning for stochastic parameterization: generative adver-sarial networks in the lorenz '96 model," *Journal of Advances in Modeling Earth Systems*, vol. 12, 2020.

[64] S. Rasp, "Online learning as a way to tackle instabilities and biases in neural network parameterizations," 2019, http://arxiv.org/abs/1907.

[65] M. S. Pritchard, C. S. Bretherton, and C. A. DeMott, "Restricting 32–128 km horizontal scales hardly affects the MJO in the Superparameterized Community Atmospher-eModel v. 3.0 but the number of cloud-resolving grid columns

constrains vertical mixing," *Journal of Advances in Modeling Earth Systems*, vol. 6, pp. 723–739, 2014.

[66] P. Gentine, M. Pritchard, S. Rasp, G. Reinaudi, and G. Yacalis, "Could machine learning break the convection parameterization deadlock?" *Geophysical Research Letters*, vol. 45, no. 11, pp. 5742–5751, 2018.

[67] J. Towns, "XSEDE: accelerating scientific discovery," *Computing in Science Engineering*, vol. 16, pp. 62–74, 2014.