frontiers | Frontiers in Climate

TYPE Original Research
PUBLISHED 15 July 2022
DOI 10.3389/fclim.2022.953262

OPEN ACCESS

EDITED BY
Karumuri Ashok,
University of Hyderabad, India

REVIEWED BY
Venkata Ratnam Jayanthi,
Japan Agency for Marine-Earth Science and Technology, Japan
Matthew Collins,
University of Exeter, United Kingdom
Subimal Ghosh,
Indian Institute of Technology Bombay, India

*CORRESPONDENCE
Kyle Joseph Chen Hall
hallkjc01@gmail.com

SPECIALTY SECTION
This article was submitted to Predictions and Projections, a section of the journal Frontiers in Climate

RECEIVED 26 May 2022
ACCEPTED 27 June 2022
PUBLISHED 15 July 2022

CITATION
Hall KJC and Acharya N (2022) XCast: A python climate forecasting toolkit. *Front. Clim.* 4:953262. doi: 10.3389/fclim.2022.953262

XCast: A python climate forecasting toolkit

Kyle Joseph Chen Hall[1]* and Nachiketa Acharya[2,3]

[1]International Research Institute for Climate and Society, Columbia University, Palisades, NY, United States, [2]Cooperative Institute for Research in Environmental Sciences, University of Colorado Boulder, Boulder, CO, United States, [3]National Oceanic and Atmospheric Administration (NOAA) Physical Sciences Laboratory, Boulder, CO, United States

Climate forecasts, both experimental and operational, are often made by calibrating Global Climate Model (GCM) outputs with observed climate variables using statistical and machine learning models. Often, machine learning techniques are applied to gridded data independently at each gridpoint. However, the implementation of these gridpoint-wise operations is a significant barrier to entry to climate data science. Unfortunately, there is a significant disconnect between the Python data science ecosystem and the gridded earth data ecosystem. Traditional Python data science tools are not designed to be used with gridded datasets, like those commonly used in climate forecasting. Heavy data preprocessing is needed: gridded data must be aggregated, reshaped, or reduced in dimensionality in order to fit the strict formatting requirements of Python's data science tools. Efficiently implementing this gridpoint-wise workflow is a time-consuming logistical burden which presents a high barrier to entry to earth data science. A set of high-performance, easy-to-use Python climate forecasting tools is needed to bridge the gap between Python's data science ecosystem and its gridded earth data ecosystem. XCast, an Xarray-based climate forecasting Python library developed by the authors, bridges this gap. XCast wraps underlying two-dimensional data science methods, like those of Scikit-Learn, with data structures that allow them to be applied to each gridpoint independently. XCast uses high-performance computing libraries to efficiently parallelize the gridpoint-wise application of data science utilities and make Python's traditional data science toolkits compatible with multidimensional gridded data. XCast also implements a diverse set of climate forecasting tools including traditional statistical methods, state-of-the-art machine learning approaches, preprocessing functionality (regridding, rescaling, smoothing), and postprocessing modules (cross validation, forecast verification, visualization). These tools are useful for producing and analyzing both experimental and operational climate forecasts. In this study, we describe the development of XCast, and present in-depth technical details on how XCast brings highly parallelized gridpoint-wise versions of traditional Python data science tools into Python's gridded earth data ecosystem.

We also demonstrate a case study where XCast was used to generate experimental real-time deterministic and probabilistic forecasts for South Asian Summer Monsoon Rainfall in 2022 using different machine learning-based multi-model ensembles.

KEYWORDS

geospatial data, machine learning, statistical learning, Python tools, climate forecasting, parallel computing, global climate models

Introduction

For sub-seasonal to seasonal (S2S) climate forecasting, using machine learning techniques to model relationships between gridded global climate model (GCM) outputs and observed climate variables is a core research technique. While there are a number of ways to make these insights, like pattern regression and aggregation, it is often desirable to apply a given technique independently at each point in space because it circumvents the need for domain sensitivity analysis, and dramatically reduces the number of predictors used. Gridpoint-wise modeling methods are commonly used to generate both research products and operational climate forecasts. In climate forecasting, these gridpoint-wise operations generally fall into one of several categories: statistical post-processing, GCM bias correction, or multi-model ensemble forecasting.

Background

There are ongoing efforts by the Python community to facilitate the manipulation of gridded multidimensional datasets in Python. Base functionality, like arithmetic, file reading and writing, and visualization, is handled by Xarray, a powerful open-source library (Hoyer and Hamman, 2017). Xarray is supported by PanGEO, a large open-source community focusing on geospatial data applications in Python (Odaka et al., 2020). Xarray data structures have been largely adopted as the fundamental units of gridded data in Python, enough so that an ecosystem of climate data analytics libraries designed to produce and consume them has emerged. This ecosystem also includes climate data analytics libraries designed for dynamical modeling like MetPy (May et al., 2022), ensemble production and forecast verification packages like ClimPred (Brady and Spring, 2021), distributed model training tools like Dask-ML (Rocklin, 2015), and many others.

Python also implements a diverse set of statistical and machine learning libraries, designed for the production and consumption of traditional flat datasets. Scikit-Learn, a commonly used machine learning library, is one of many such Python data science utilities that operates nearly exclusively on two-dimensional datasets (Pedregosa et al., 2011). Like Scikit-learn, the majority of Python's data science utilities operate on two-dimensional data, and are not designed to accommodate gridpoint-wise operations on Xarray data structures. The tools that do accommodate multidimensional data do not accommodate Xarray data types. This gap between Python's data science utilities and its gridded data ecosystem is notable. More details about Scikit-Learn and the libraries upon which XCast is built are available in Appendix C.2.

Motivations

The implementation of gridpoint-wise operations, like those commonly used in climate forecasting workflows, and especially in the statistical post-processing (calibration, mulit-model ensembling) of GCM's outputs, serves as a significant barrier to entry to climate data science. Moreover, with the recent advances in machine learning and climate science, there are numerous state-of-art machine learning techniques used in climate forecasting (Acharya et al., 2014; Kashinath et al., 2020; Gibson et al., 2021) which are not easy to implement without a strong computer programming background. While there are powerful utilities for manipulating gridded data in Python, and numerous libraries and toolkits for two-dimensional statistical modeling and machine learning, extremely few support this gridpoint-wise approach to climate prediction. The need to bridge the gap between Python's data science utilities and its gridded data utilities is evident.

Gridpoint-wise operations like those used in multi-model ensemble forecasting are also time consuming and computationally expensive. Their spatial independence theoretically allows them to be parallelized, but implementing true parallel computation in Python is a specialized task which requires a large time investment. If climate scientists must also become computer scientists to implement their experiments, their domain-specific research will suffer. The inaccessibility of modern High-Performance Computing (HPC) tools therefore presents a significant barrier to entry to climate data science. We aim to significantly decrease the barriers to climate data science by bridging the gap between Python's data science utilities

and its gridded data utilities with a flexible, easy-to-learn, and highly-performant set of tools for geospatial data science.

Contributions: XCast

We, the authors, have co-developed just such a highly-performant, easy-to-learn, and flexible set of geospatial data science tools called XCast (https://xcast-lib.github.io). XCast allows the user to train machine learning methods like regressions, decision trees, neural networks, and more, gridpoint by gridpoint, directly on gridded datasets with significantly less preprocessing. It speeds up these time-consuming workflows by making modern high-performance computing methods like chunk-wise parallelism and cluster computing available to the user through an intuitive application programming interface (API). XCast's API purposefully mirrors the APIs of traditional Python data science tools like Scikit-Learn to leverage users' prior knowledge and minimize the time investment required to start using it (Pedregosa et al., 2011). The API similarities allow climate scientists and other users of traditional Python data science utilities, to transition from single-point data science workflows to gridpoint-wise data science workflows almost seamlessly. XCast effectively utilizes Dask, a high-performance computing library for Python to abstract away nearly all use of parallelization and multiprocessing, relieving the user of the burden of having to design efficient workflows (Rocklin, 2015). Through Dask, XCast is capable of leveraging high-capacity computing environments like clusters and supercomputers. It also makes this kind of gridpoint-wise analysis possible on an individual workstation or laptop. The simple interface lets scientists focus on the science behind the predictability of climate phenomena, rather than on efficient computing. XCast implements climate forecasting tools like cross-validation, gaussian kernel smoothing, rescaling, and regridding, as well as a forecast verification module.

Road to XCast: Predecessors and development

Although it can be used in a broad array of scientific domains like meteorology, atmospheric science and dynamical modeling, XCast derives its name from its initial purpose, climate forecasting, and the fundamental data structure it consumes, the Xarray Data Array. It grew out of PyELM-MME, a platform for Extreme Learning Machine (ELM)-based Multi-Model Ensemble (MME) forecasting (Acharya and Hall, 2021a). The PyELM-MME platform was intended to make ELM-based Multi-Model Ensemble climate forecasting accessible (Acharya et al., 2014). After PyELM-MME was presented at the 2021 NCAR UCAR SEA Improving Scientific Software conference, user feedback and peer review made it apparent that a generalized version, which would make numerous statistical and machine learning-based MME methodologies accessible, was necessary. PyMME, the successor to PyELM-MME, emerged to fill that gap. It was designed to make machine-learning based MME forecasting methods, in general, accessible, and was presented during a poster session for the third NOAA workshop on Leveraging AI in the Environmental Sciences (Acharya and Hall, 2021b). However, during the PyMME development process, a major pain point became apparent, through user feedback and suggestions by community stake holders: the gridpoint-wise operations implemented were not usable outside of the PyMME forecasting setting. Since both PyMME and PyELM-MME were implemented specifically in Jupyter Notebook environments, their functionality was not easily transferable to other use cases. The need for a generalized gridpoint-wise data science library was evident.

We realized that, since the gridpoint-wise approach to data science operations is so broadly applicable in the earth sciences, any MME climate forecasting platform should be built as a special case of a more general gridpoint-wise data science toolkit. There is no reason to require earth scientists in other domains to retrofit an MME forecasting system to suit their own needs, when a library can be designed to make all gridpoint-wise data science operations accessible at once. Abandoning the Jupyter Notebook-based forecasting platform design, we implemented XCast as a general-purpose Python library so it could be used in any given environment, and decided to distribute it with Anaconda in order to make the installation process fast, easy, and compatible with other similar libraries. The first stable version of XCast, version 0.5.0, was released in March, 2022, and is available on Anaconda for installation on any operating system. Figure 1 shows the timeline of XCast's use and development up to that point.

Design and implementation of XCast

Unlike most other climate forecasting tools, XCast is designed to be fully modular. While there is a general pattern to XCast forecasting workflows, shown in Figure 2, each of XCast's classes and functions can be used independently, as part of custom user-defined Xarray-based workflows. Appendix C.1 lists all of XCast's available statistical methods and machine learning models, but additional flat estimators and skill metrics can be implemented easily. Appendix C.2 discusses these third-party dependencies in more detail. XCast's utilities are easy to integrate with other verification and forecasting Python modules, allowing the user to independently verify and validate XCast-based predictions. This enforces transparency, and provides accountability, which are critical aspects of a credible forecasting tool. Additionally, XCast's classes and functions are wrapped in intuitive, Scikit-Learn-like interfaces. The behavior of class methods and functions is consistent with

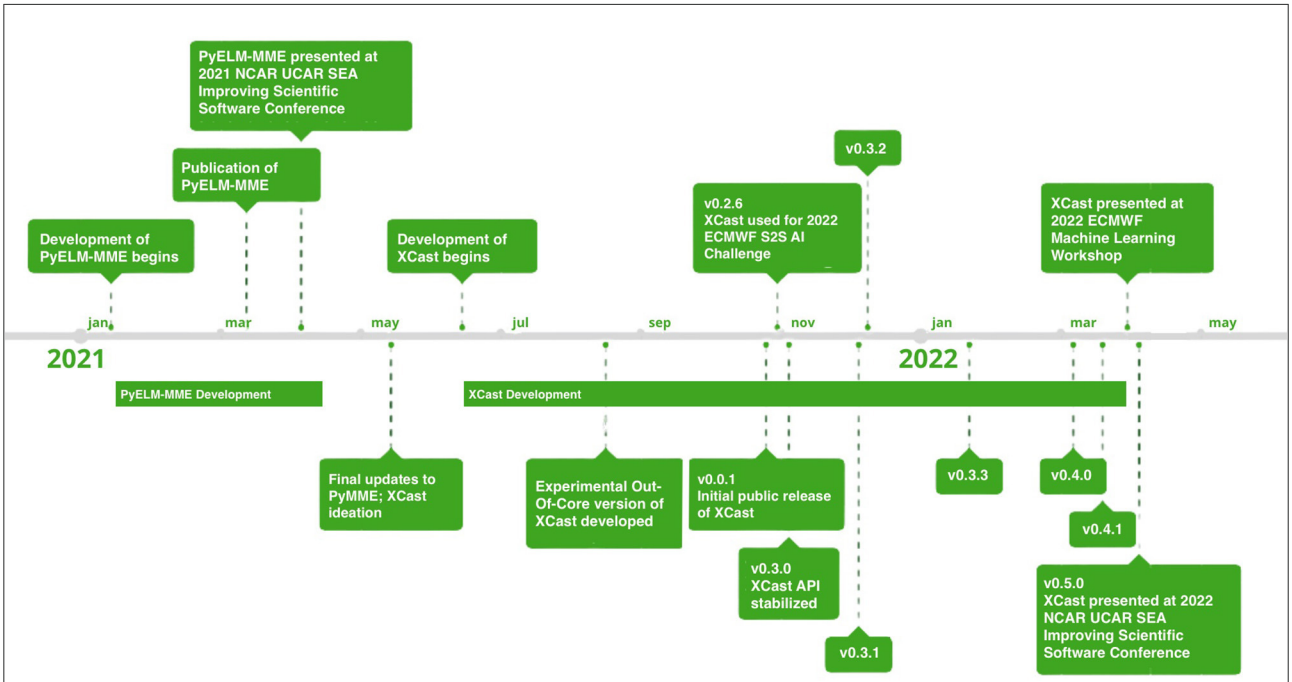

FIGURE 1
A timeline of the development of XCast.

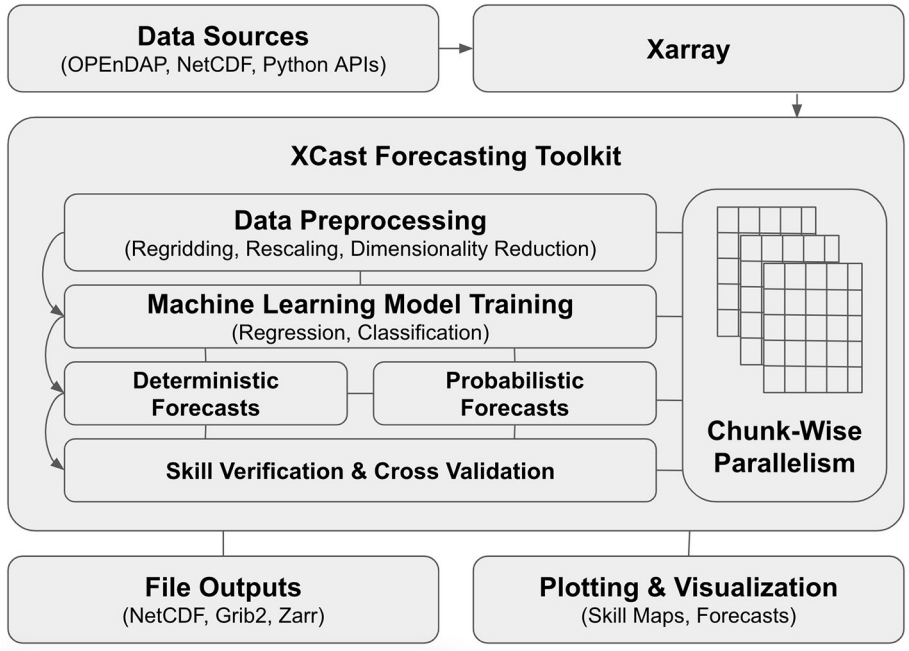

FIGURE 2
The workflow of the XCast Toolkit.

that of those found in Scikit-Learn and other Python data science utilities. This allows users to transition between single-point estimators and XCast estimators seamlessly, without checking documentation and interrupting the development process. XCast's Scikit-Learn-like interface also abstracts out all of the low-level high-performance computing functionality

that it's built with, giving the user performance benefits without requiring significant intervention by them.

More information about XCast's functionality, implementation, and installation, as well as examples and other case studies, are available in the project's documentation (https://xcast-lib.github.io/), and the project's Github repository (https://github.com/kjhall01/xcast).

Application programming interface design

Leveraging the prior knowledge of XCast's target user base makes it easy to learn. Its interface implements in three archetypical data structures: Estimators, Transformers, and Functions, all of which are patterns commonly used in traditional Python data science libraries.

Estimators

Estimators are Python objects representing statistical and machine learning models, like for example, neural networks. Estimators can be either Regressors or Classifiers—where regressors attempt to make deterministic guesses based on inputs, and classifiers attempt to make probabilistic guesses as to which category of output the inputs are associated with. Scikit-Learn and other traditional Python data science utilities also use this terminology, so users may already be familiar with it. Estimators generally have three methods- fit(..), predict(…) and predict_proba(…). Fit, as the name implies, fits the underlying statistical model on user-provided training data. Predict attempts to make deterministic predictions with a previously-fit model, based on user-provided predictor-like inputs. Predict_proba is available on classifiers, and returns probabilistic predictions.

Transformers

Transformers are similar to estimators, with the exception that they generally do not find relationships between inputs and outputs, but rather identify and apply patterns based on a single input dataset. Rather than fit-predict, transformers implement a fit-transform workflow, where fit identifies a pattern or transformation based on an input dataset, and transform applies it to new data. Principal Components Analysis is an example of a transformer. The majority of XCast's preprocessing functionality is implemented as Transformers. Rescaling and dimensionality reduction are represented as objects to be fit on input datasets, so they can be reused for out-of-sample data. Some preprocessing functionality which does not depend on a given input dataset, like regridding, is implemented as functions.

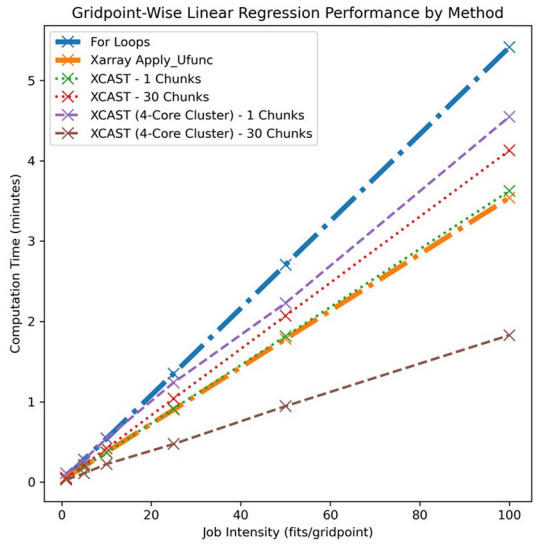

FIGURE 3
The results of an experiment analyzing the comparative performance of six implementations of gridpoint-wise model fitting. Training time in fraction of minutes is plotted against the ensemble size as an analog for job intensity.

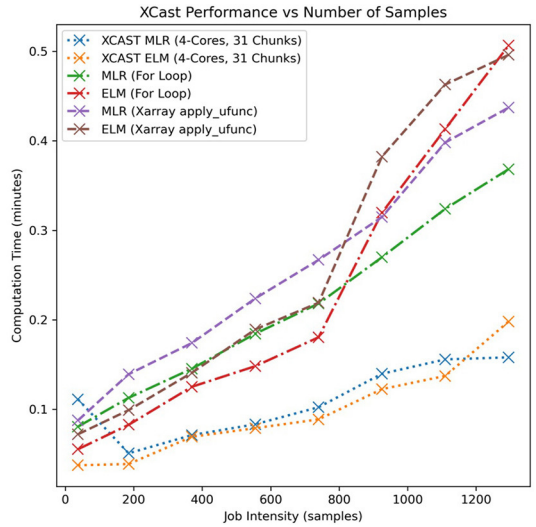

FIGURE 4
Comparison of training time for a 31-chunked, 4-core configuration of XCast with that of Xarray and For-Loop-based implementations. Number of training samples at each point is used as an analog for job intensity, and two different statistical methods are shown.

Functions

Gridpoint-wise functions are a data type in XCast which apply some flat function to user-provided input data directly gridpoint-by-gridpoint, without needing to identify any kind of

pattern or transformation beforehand. They are largely defined with the "metric" decorator function, to make new metrics easy to build. Functions can accept one or multiple input datasets, depending on the underlying function, and usually return one or multiple output datasets. Gaussian Kernel Smoothing, Skill Metrics, and One-Hot Encoding are all examples of XCast functions.

Similarity to Scikit-learn

In order to minimize the time required to learn how to use XCast, its API was specifically designed to leverage users' prior knowledge. Xarray Data Arrays were chosen as the atomic unit of XCast's analyses because of the high degree of Xarray adoption by the Python earth science community. Numerous other Python climate analytics libraries also adopt Xarray Data Arrays, which makes those libraries inherently compatible with XCast. Any user of those libraries, or of Xarray itself, would already have the requisite skills to use gridded data with XCast. The interface is also intentionally designed to mimic that of SciKit-learn, perhaps the most popular and well-supported Python data science library. The similarity between XCast's API and that of SciKit-Learn (Pedregosa et al., 2011), and the adoption of the Xarray DataArray, make the transition from flat, two-dimensional SciKit-Learn data science to multidimensional gridpoint-wise data science simple - all one need do is swap NumPy arrays (Harris et al., 2020) for Xarray DataArrays. The difference between NumPy arrays and Xarray DataArrays is made evident by the discussion in Appendix C.2.

Parallelism

In pursuit of high-performance, XCast uses Dask (Rocklin, 2015) to implement chunk-wise parallelism. Gridpoint-wise operations are uniquely parallelizable, since the computations at each point are completely independent of one another. The parallelism in XCast significantly decreases the time investment for gridpoint-wise data science operations, and its natural compatibility with Dask allows XCast to be scaled up to institutional computer clusters with ease.

The gridpoint-wise approach to multidimensional gridded data operations exemplifies the benefits of parallel processing. Theoretically, each grid point's computation could be performed by a separate computer. Although unrealistic, that level of parallelism would effectively eliminate the spatial dimensions of the dataset, and reduce the time required to complete all the gridpoint-wise operations to the time required by a single gridpoint. Fortunately, it is possible to implement a more generalized version of the concept: chunk-wise parallelism (Dab and Slama, 2017). Rather than parallelizing gridpoint-wise operations by splitting individual grid points, this type of problem can be parallelized by splitting a dataset into spatially local groups of grid points called chunks. Each chunk can then be distributed to an independent processor, which reduces the time required proportionally to the number of chunks. In fact, parallelizing operations by splitting individual grid points can be understood as chunk-wise parallelism with a chunk size of one.

In cases where only a small number of processors are available to implement chunk-wise parallelism, the performance gain or reduction in computation time is not strictly proportional to the number of chunks. When there are more chunks than processors available, each processor must perform computations for several chunks in serial- losing time, compared to perfect parallelism. Practically, this often still represents a significant performance boost when compared to fully serial operations. Even further, since XCast implements chunk-wise parallelism by leveraging Dask, it can easily be scaled to institutional supercomputers and computer clusters. Using Dask to scale XCast's gridpoint-wise operations to powerful machines lets the user further approach the perfect case of parallelism. Additionally, XCast gives explicit control over the level of parallelization to the user, by allowing them to specify the size and number of chunks so they can optimize XCast's performance based on the specifications of their work station. Further explanation of Dask's pure-python multiprocessing implementation and documentation links can be found in Appendix C.2.

Analyzing the impact of XCast parallelism on model training time

Gridpoint-wise operations, especially machine learning model fitting, can be extremely time consuming. Depending on the type of model and the context, training a single instance can take seconds or minutes. If one were to train such a model at each point on a relatively coarse $1^0 \times 1^0$ global gridded dataset, the training time would increase dramatically- 64,800 vs. 1 min. Cross-validation compounds this problem even further. Gridpoint-wise parallelism makes this kind of analysis possible- dividing model training efficiently among multiple compute cores decreases the time required significantly. It also introduces extra overhead, which, depending on the case, can cancel out performance gains. In order to identify which circumstances make XCast's chunk-wise parallelism worthwhile, we analyzed how different implementations of gridpoint-wise operations in Python scale with job intensity.

Experimental design

During this experiment, multiple distinct methods were used to fit an ensemble of several linear regression models individually at each grid point of a gridded seasonal precipitation dataset. Although linear regression is generally solved analytically, many types of machine learning models

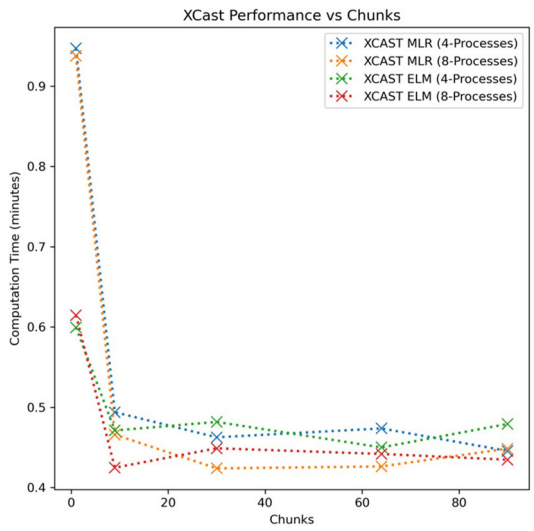

FIGURE 5
XCast training time plotted against the number of chunks in the training data. Two Dask clusters configurations with different numbers of parallel processes are compared.

```
from dask.distributed import Client
client = Client("127.0.0.1:8080")

X, Y, T = xc.NMME_IMD_ISMR()
X, Y = xc.align_chunks(X, Y, 5, 5)
mlr = xc.rMultipleLinearRegression()
mlr.fit(X, Y, rechunk=False)
mlr.predict(X, rechunk=False)
```

FIGURE 6
An example of an XCast model training workflow which leverages Dask multiprocessing.

are stochastic and iterative in nature. Large ensembles of those methods are commonly used to account for their stochasticity. Increasing ensemble size usually corresponds with a proportional increase in training time, so in order to examine how well each of these implementations scales, ensemble size was used as an analog for training time. The lengths of time required to fit gridpoint-wise linear regression ensembles of successively increasing size were calculated, recorded, and plotted for each method.

Four XCast configurations and two benchmark methods were compared during this experiment. The first benchmark was a naïve for-loop implementation of gridpoint-wise model training, and was represented by the blue line in Figure 3. This method iterated over the spatial dimensions in serial. The second, represented by the orange line in Figure 3, was a native Xarray function called "apply_ufunc". This method applied arbitrary functions along predefined core dimensions of the dataset. Four configurations of XCast were also analyzed. The first involved fully contiguous data, and used only a single process (represented in green), the second used a single process but split the data into thirty chunks (represented in red), the third use fully contiguous data and four processes operating in parallel (represented in purple), and the fourth used thirty chunks and four parallel processes (represented in brown). All tests shown were performed on a Mid-2015 quad-core MacBook Pro with Intel i7 processors.

Interpreting results

The configuration of XCast which used a single process and a single contiguous data chunk (green) exhibited very similar performance to that of Xarray (orange). XCast and Xarray both use Dask to implement gridpoint-wise operations, but do so slightly differently. Intuitively, their performance should be similar; XCast does not claim to have implemented new or revolutionary parallel computing algorithms, just to have made the existing ones easily accessible to climate forecasters.

Comparatively, the XCast configurations which use a single process but multiple chunks (red) and multiple processes but a single chunk (purple) exhibit decreased performance. This is because rechunking data and initializing Dask clusters both introduce overhead. Additionally, neither of these configurations actually use parallelism since parallel computing with Dask requires both multiple processes and multiple data chunks. In both cases, the overhead results in a net performance loss. The only XCast configuration analyzed during this experiment which benefits from multi-process parallelism is that which implements both many data chunks and many parallel processes (brown). The performance benefit from parallelism outweighs the overhead introduced by rechunking and cluster initialization, and results in a net performance gain.

The for-loop implementation (blue) makes no attempt at parallelism or vectorization, and is a clear performance loser. Figure 4 further reinforces this result by demonstrating that this relationship holds true for two types of machine learning models, as well as under a different analog of training intensity-number of training samples.

Some initial investigation of the relationship between the number of chunks, the number of available cores, and the computation time was performed. The results are detailed in Figure 5; unfortunately, the strongest statement that can be made under this environment is that XCast does not benefit from parallelism when either (1) there is only one core available for computation or (2) the dataset is completely contiguous, or in other words, consists of only one chunk. This is intuitive, because XCast's parallelism works by distributing different chunks to different cores—any scenario where multiple cores cannot work concurrently will result in serial execution of any available chunks, eliminating parallelism.

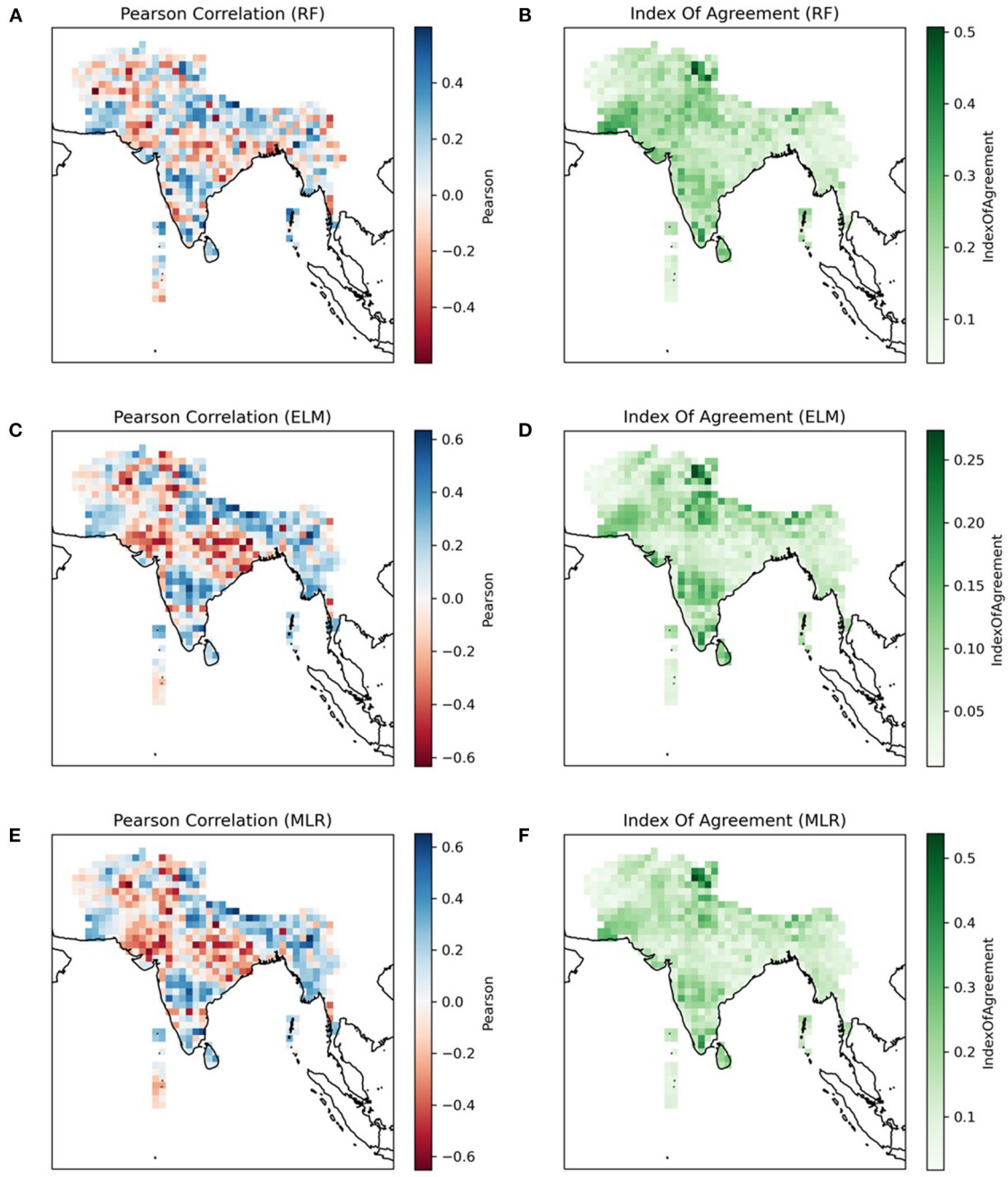

FIGURE 7
Pearson Correlation and Index of Agreement for Random Forest (RF) **(A,B)**, Extreme Learning Machine (ELM) **(C,D)**, and Multiple Linear Regression (MLR) **(E,F)**.

The difference between XCast and Xarray

Since XCast and Xarray are both implemented with Dask, one might wonder why Xarray's performance in the above analysis is not the same as that of XCast. Theoretically, it should be. Xarray's "appy_ufunc" function is designed to accommodate an incredibly broad space of functions and input arrays, including high-dimensional objects and multi-dimensional functions. This requires it to go through dozens of steps of validation, format checking, and compatibility checking. It also wraps an extra layer of Dask function calls, when compared

to XCast. Xarray.apply_ufunc calls dask.array.apply_gufunc, which in turn calls dask.array.blockwise. XCast estimators directly call dask.array.blockwise. The extra layers of function calls and generalizations make it difficult to successfully use Xarray's "apply_ufunc" in conjunction with a Dask cluster.

XCast, however, specializes in applying Dask's chunk-wise parallelism to four-dimensional XCast-formatted datasets; it makes no attempts to generalize to even higher dimensions. This comparative loss of generality allows XCast to focus specifically on compatibility with Dask. While it may theoretically be possible to use Xarray.apply_ufunc with multicore Dask clusters, the layers of abstract syntax and complicated function calls make it unnecessarily difficult, preventing all but the most dedicated programmers from taking advantage of it. XCast makes Dask chunk-wise parallelism easy to use.

Figure 6 is an XCast code sample which demonstrates how XCast requires no modification to take advantage of Dask. If the user instantiates a Dask cluster, XCast uses it without further intervention. In Figure 6, the first two lines of code import Dask and instantiate a Dask Cluster, which activates and manages multiple concurrent processes. The fourth line uses a convenience function in XCast, "NMME_IMD_ISMR" to load North American Multi-Model Ensemble (NMME) predictor data and India Meteorological Department (IMD) rainfall data for a period during the Indian Summer Monsoon (ISMR) (Kirtman et al., 2013; Sridhar et al., 2020). The "align_chunks" function serves to transpose and rechunk these two datasets, adopting a chunking scheme splitting the datasets into five groups along each spatial dimension for a total of 25 chunks. XCast then instantiates and trains a gridpoint-wise linear regression model, and uses it to make predictions. Without any direct changes to the XCast function calls, XCast proceeds to leverage the multiprocessing capabilities of the instantiated Dask cluster.

Case study: Showcasing XCast's functionality

XCast can be used in a variety of settings; it can address any problem that requires the gridpoint-wise application of statistical or machine learning tools. As a byproduct of the authors' area of study, the majority of use cases implementing XCast so far involve forecasting precipitation for different time scales and geographic domains. Here, we present a case study where XCast was used to generate precipitation forecasts using multi-model ensembles (MME). The main purpose of this case study is to showcase the functionality of XCast.

Seasonal forecasts of South Asian summer monsoon rainfall

In Spring 2022, XCast was used by the authors to generate deterministic and probabilistic forecasts of Summer Monsoon Rainfall (SMR) for the South Asian region. South Asian summer monsoon season takes place during the months of June-September (JJAS), and total rainfall during this season displays high interannual variability, which can make it difficult to generate skillful forecasts with unprocessed General Circulation Model (GCM) output (Acharya et al., 2011b). Fortunately, MME forecasts are well-accepted as an improvement on GCM forecasts (Weigel et al., 2008; Casanova and Ahrens, 2009; Acharya et al., 2011a). They are generally produced by averaging ensemble member outputs with equal weight or with weight according to prior skill, but there is also significant interest in using machine learning-based regression techniques for MME construction after the pioneering work by Acharya et al. (2014). In this exercise, we used XCast to produce an April-initialized real-time SMR forecast. The full details of this forecast can be found at https://www.kjhall01.github.io/SASCOF22/ (Acharya and Hall, 2022).

Machine learning methodologies

XCast is used here to train models from each of three core families of machine learning methodologies: Regression-based techniques, Decision Tree-based techniques, and Neural Network-based techniques. Multiple Linear Regression (MLR), a commonly used core regression-based technique, is used as a baseline for comparison. We selected Random Forest Regression (RF), which is a collection of randomly-initialized decision trees and has the potential to generalize well, to represent the larger space of decision tree-based methods (Breiman, 2001). Extreme Learning Machine (ELM) was selected to represent the neural network-based family of methods, because of its short training time and flexibility (Huang et al., 2008). Extreme Learning Machine substitutes time consuming, backpropagation-based training for frozen hidden layer weights and biases, and solves its output layer with the Moore-Penrose generalized inverse. For the experimental probabilistic forecast, we used a modified version of ELM known as Probabilistic Output Extreme Learning Machine (POELM) to produce probabilistic such tercile forecasts (Wong et al., 2020). POELM fits three output neurons, separately, on binary-encoded target vectors representing each of the tercile categories. It then applies a postprocessing rule which uses regularization to generate multi-class probabilities which sum to one.

This particular exercise only uses a small subset of the machine learning models available through XCast to highlight its utility, a full list can be found in Appendix C.1.

Dataset

Lead-2 (initialized in April for JJAS) hindcast data from four GCMs was obtained from the phase 2 of NMME project (Kirtman et al., 2013). Hindcast data during the period 1982–2018 was used as predictors. The real-time forecast for JJAS 2022 from each model was also used to produce an experimental operational forecast, out-of-sample. As the GCMs have different numbers of ensemble members, for each GCM, ensemble members were averaged to generate an ensemble mean before implemented MME. These NMME monthly hindcast and forecast datasets are available on a common 1○ resolution grid at http://iridl.ldeo.columbia.edu/SOURCES/Models/NMME/. An observed precipitation dataset, the latest version of daily gridded precipitation data from the India Meteorological Department, Pune (Sridhar et al., 2020) was used as the predictand. This data is at the 1○ spatial resolution, which allows for good spatial coverage of entire south Asia. Seasonal total observed rainfall for JJAS is calculated for 1982 to 2018 (unit: mm/Season).

Preprocessing

Seasonal precipitation data provided by SASCOF was anomalized and used as the predictand. XCast was used to apply Minmax preprocessing (Normalization) to the predictors. For the experimental probabilistic forecasts, a process called One-Hot Encoding was applied to the predictand. One-Hot Encoding involves identifying tercile category boundaries at the 33rd- and 66th- percentiles, and then generating binary-encoded vectors for each category which indicate whether or not a given sample falls into that category. Leave-One-Year-Out cross validation was used to construct a cross-validated hindcast dataset, which was then compared to the original predictand dataset to generate skill maps.

Hindcast skill assessment

It is necessary to evaluate the skill of the hindcasts, in order to understand of the expected skill of out-of-sample forecasts. To prevent over-fitting, and give a more reasonable and useful approximation of the skill of out-of-sample forecasts, a cross-validated hindcast dataset is generated and compared with the historical observed rainfall. In cross validation, for each of a number of "windows", a model is trained on all data except that which falls within the window. That model is then used to make out-of-sample predictions for that window, data which it has not yet been shown. These out-of-sample predictions are then reassembled into a dataset which can be meaningfully compared to the observed historical data. Skill metrics are then calculated by comparing these cross-validated hindcasts with the observations, to give an idea of how the methodology performs for out-of-sample forecasts.

Deterministic skill

Figure 7 shows the skill maps generated by comparing cross-validated hindcast data with the observed precipitation data for MLR, RF, and ELM, respectively. They show the Pearson Coefficient and Index of Agreement (IOA) for each methodology. Pearson's Coefficient is a commonly used measure of correlation, and IOA is a measure of error (Willmott, 1982). Together, these skill maps can give an indication of the level of skill of a forecast methodology. Since this exercise is meant to highlight how XCast works, we will not discuss the comparative skill levels of the forecast methodologies. While identifying the best method through skill comparison is important in a forecast development context, it is not the focus here. We aim to show that it can be easily done with the skill maps generated by XCast.

Probabilistic skill

The difficulty of generating a skillful deterministic forecast due to the high degree of interannual variability in South Asian SMR, necessitates the use of tercile probabilistic forecasts. Figure 8 shows the Generalized Receiver Operating Characteristic (GROC) score and the Rank Probability Skill Score (RPSS) of the cross-validated probabilistic hindcast dataset produced with POELM. GROC is a measure of discrimination (Mason and Graham, 2002), and RPSS is a measure of relative forecast quality (Murphy, 1969). Here, RPSS is computed relative to the climatological probability: 33, 33, 33%. Again, we make no claims about the quality of this forecast, rather just highlight the fact that XCast can be used to make probabilistic forecasts and to compute probabilistic skill scores efficiently for gridded data.

Real-time experimental forecast for summer monsoon 2022

Figure 9 shows the experimental deterministic and probabilistic forecasts produced for the year 2022 using ELM-based methods. In summary, normal to above normal rainfall is most likely during the upcoming 2022 summer monsoon season over most parts of the South Asia. Above normal rainfall is most forecasted for the foot hills of Himalayas, many areas of northwestern and central South Asia, and some areas in the east and south. Below normal rainfall, however, is forecasted for some areas of extreme north, northwest, south, and southeast parts of the region. The seasonal rainfall is most likely to be normal or of climatological probabilities over the remaining areas.

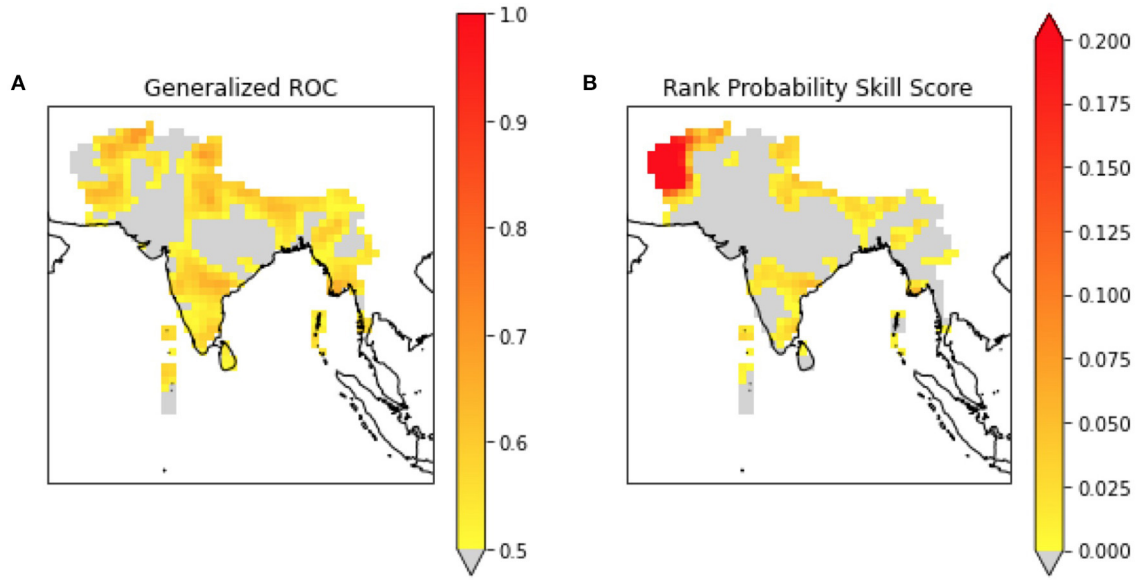

FIGURE 8
Generalized ROC **(A)** and Rank Probability Skill Score **(B)** for POELM-based probabilistic forecasts of South Asian Summer Monsoon Rainfall.

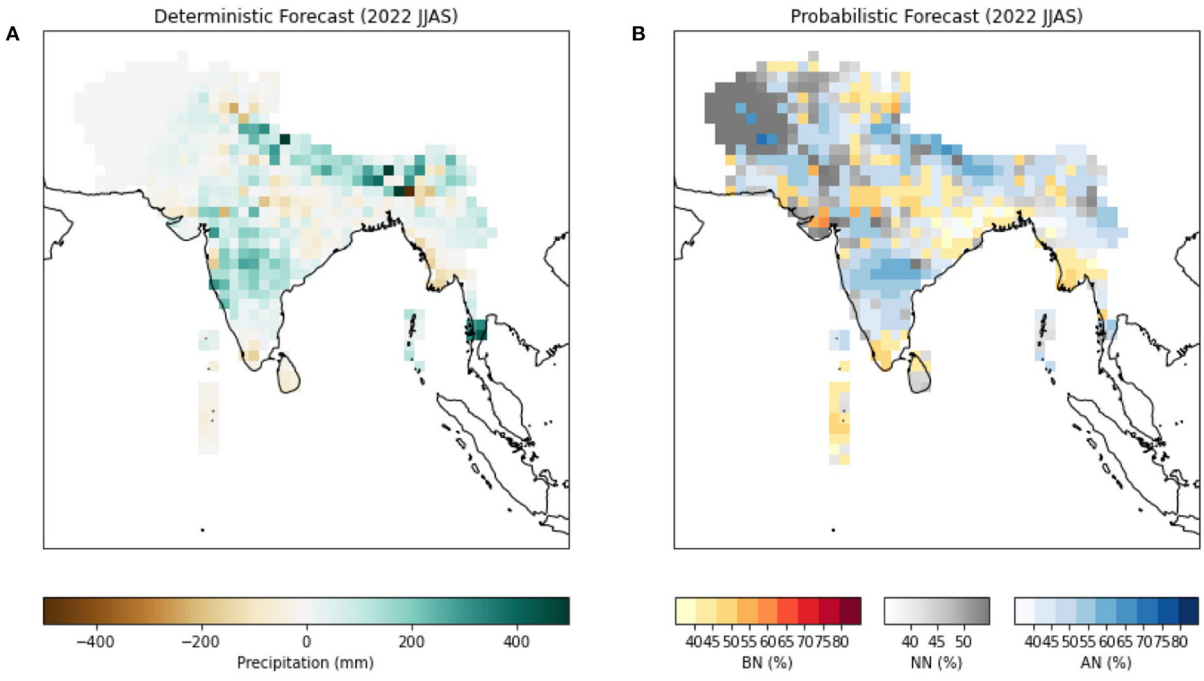

FIGURE 9
Deterministic **(A)** and probabilistic **(B)** forecasts of 2022 South Asian Summer Monsoon Rainfall produced with XCast.

Similar forecasts have been produced by other Global Producing Centers (GPC) like ECMWF and IRI. This real-time forecast was presented at the 22nd South Asian Seasonal Climate Outlook Forum (SASCOF) held April 26–28, 2022. SASCOF is organized by WMO's regional climate center for the purposes of preparing and disseminating consensus forecasts for upcoming monsoon seasons for South Asian countries.

Discussion

XCast is designed to be an easy-to-use, flexible climate forecasting Python library. Its API is meant to be easy to learn, because it leverages the prior knowledge of users familiar with Xarray and Scikit-Learn. It is also meant to make high-performance computing tools easy to incorporate in climate forecasting workflows: scaling XCast to institutional supercomputers is easy with its intrinsic compatibility with Dask. For those without institutional access, XCast can also optimize computation on a Dask "local cluster", letting users work directly on their laptops and individual workstations.

Previously, slow computation, unwieldy datasets, and opaque APIs prevented most scientists from pursuing this kind of gridpoint-wise climate forecasting workflow. XCast has made it significantly easier to pursue this type of gridpoint-wise scientific inquiry through modern HPC tools, and lowered barriers to computational earth science.

In the future, XCast's performance will be analyzed in a cluster computing setting, and the range of statistical and machine learning methods will be expanded to include cluster analysis, pattern regression, and unsupervised learning models. New features and improvements to XCast are still under development. There is certainly much more work and analysis to be done.

Data availability statement

The raw data supporting the conclusions of this article will be made available by the authors, without undue reservation.

Author contributions

KH designed, implemented, tested, and published the XCast library in collaboration with NA, who provided climate science and machine learning expertise and design feedback. NA led topic selection and experimental design of the case studies and provided insightful review and feedback. KH implemented them using XCast and produced all figures and served as primary author of this article. All authors contributed to the article and approved the submitted version.

Funding

This work was unfunded and should be considered neither a product of the IRI nor of CIRES/NOAA PSL. The authors pursued this work in their own time, outside of work hours, for the sake of scientific inquiry.

Acknowledgments

The authors humbly thank and acknowledge the NCAR UCAR Software Engineering Assembly for hosting the Improving Scientific Software conference, which has served as inspiration and accountability, the in which a prior work on this topic was published. Additionally, we thank the organizers of the South Asia Seasonal Climate Outlook Forum (SASCOF) for their observations data and the North American Multi-Model Ensemble member GPCs for the use of their forecast data. We also thank the PanGEO community for their work supporting XCast's core dependencies, as well as the developers of XClim, ClimPred, MetPy, XSkillScore and the rest of the Python Climate Data Science Ecosystem. We thank the IRI Data Library for providing access to NMME climate model data and global observations. Finally, we are grateful to the three reviewers for their insightful comments and suggestion that helped to improve the original version of the manuscript.

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

Supplementary material

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fclim.2022.953262/full#supplementary-material

References

Acharya, N., and Hall, K. J. C. (2021a). "PyELM-MME: a Python platform for extreme learning machine based multi-model ensemble," in *Proceedings of the 2021 Improving Scientific Software Conference (No. NCAR/TN-567+PROC)*.

Acharya, N., and Hall, K. J. C. (2021b). "PyMME: A Python platform for multi-model ensemble climate predictions," in *3rd NOAA Workshop on Leveraging AI in Environmental Sciences* (Zenodo).

Acharya, N., and Hall, K. J. C. (2022). *kjhall01/SASCOF22: SASCOF Extreme Learning Machine Forecasts (v1.0.0)*. South Asian Seasonal Climate Outlook Forum. Zenodo.

Acharya, N., Kar, S. C., Kulkarni, M. A., Mohanty, U. C., and Sahoo, L. N. (2011a). Multi-model ensemble schemes for predicting northeast monsoon rainfall over peninsular India. *J. Earth Syst. Sci.* 120, 795–805. doi: 10.1007/s12040-011-0111-4

Acharya, N., Kar, S. C., Mohanty, U. C., Kulkarni, M. A., and Dash, S. K. (2011b). Performance of GCMs for seasonal prediction over India - A case study for 2009 monsoon. *Theor. Appl. Climatol.*, 105, 505–520. doi: 10.1007/s00704-010-0396-2

Acharya, N., Srivastava, N. A., Panigrahi, B. K., and Mohanty, U. C. (2014). Development of an artificial neural network based multi- model ensemble to estimate the northeast monsoon rainfall over south peninsular India: an application of extreme learning machine. *Clim. Dyn.* 43, 1303–1310. doi: 10.1007/s00382-013-1942-2

Brady, R., and Spring, A. (2021). Climpred: verification of weather and climate forecasts. *J. Open Source Soft.* 6, 2781. doi: 10.21105/joss.02781

Breiman, L. (2001). Random forests. *Mach. Learn.* 45, 5–32. doi: 10.1023/A:1010933404324

Casanova, S., and Ahrens, B. (2009). On the weighting of multimodel ensembles in seasonal and short-range weather forecasting. *Monthly Weather Rev.* 137, 3811–3822. doi: 10.1175/2009MWR2893.1

Dab, A., and Slama, Y. (2017). "Chunk-wise parallelization based on dynamic performance prediction on heterogeneous multicores," in *2017 International Conference on High Performance Computing & Simulation (HPCS)*, 117–123. Available online at: https://ieeexplore.ieee.org/document/8035067

Gibson, P. B., Chapman, W. E., Altinok, A., Delle Monache, L., DeFlorio, M. J., and Waliser, D. E. (2021). Training machine learning models on climate model output yields skillful interpretable seasonal precipitation forecasts. *Commun. Earth Environ.* 2, 159. doi: 10.1038/s43247-021-00225-4

Harris, C. R., Millman, K. J., Van Der Walt, S. J., Gommers, R., Virtanen, P., and Cournapeau, D. (2020). Array programming with NumPy. *Nature* 585, 357–362. doi: 10.1038/s41586-020-2649-2

Hoyer, S., and Hamman, J. (2017). Xarray: N-D labeled arrays and datasets in Python. *J. Open Res. Softw.* 5, 10. doi: 10.5334/jors.148

Huang, G. B., Li, M. B., Chen, L., and Siew, C. K. (2008). Incremental extreme learning machine with fully complex hidden nodes. *Neurocom. Puting* 71, 576–583. doi: 10.1016/j.neucom.2007.07.025

Kashinath, K., Mustafa, M., Albert, A., Wu, J.-L., Jiang, C., Esmaeilzadeh, S., et al. (2020). Physics-informed machine learning: case studies for weather and cli-mate modelling, Philos. *T. Roy. Soc. A* 379, 20200093. doi: 10.1098/rsta.2020.0093

Kirtman, B. P., Min, D., Infanti, J. M., Kinter, J. L., Paolino, D. A., Zhang, Q., et al. (2013). The north american multi-model ensemble (NMME): Phase-1 seasonal to interannual prediction, phase-2 toward developing intra-seasonal prediction. *Bull. Amer. Meteor. Soc.* 95, 585–601. doi: 10.1175/BAMS-D-12-00050.1

Mason, S. J., and Graham, N. E. (2002). Areas beneath the relative operating characteristics (ROC) and levels (ROL) curves: Statistical significance and interpretation. *Quart. J. Roy. Meteor. Soc.* 128, 2145–2166. doi: 10.1256/003590002320603584

May, R. M., Arms, S. C., Marsh, P., Bruning, E., Leeman, J. R., Goebbert, K., et al. (2022). *MetPy: A Python Package for Meteorological Data*. doi: 10.5065/D6WW7G29

Murphy, A. H. (1969). On the ranked probability skill score. *J. Appl. Meteor.* 8, 988–989. doi: 10.1175/1520-0450(1969)008<0988:OTPS>2.0.CO;2

Odaka, T. E., Banihirwe, A., Eynard-Bontemps, G., Ponte, A., Maze, G., Paul, K., et al. (2020). "The Pangeo ecosystem: interactive computing tools for the geosciences: benchmarking on HPC," in *Tools and Techniques for High Performance Computing. HUST SE-HER WIHPC 2019 2019 2019. Communications in Computer and Information Science, Vol 1190*, eds G. Juckeland, and S. Chandrasekaran, S (Cham: Springer).

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., et al. (2011). *Scikit-learn: Machine Learning in Python, JMLR 12*, 2825–2830. Available online at: http://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html (accessed July 1, 2022).

Rocklin, M. (2015). "Dask: parallel computation with blocked algorithms and task scheduling," in *Proceedings of the 14th Python in Science Conference*. Available online at: http://citebay.com/how-to-cite/dask/

Seabold, S., and Josef, P. (2010). "statsmodels: econometric and statistical modeling with python," in *Proceedings of the 9th Python in Science Conference*. Available online at: https://conference.scipy.org/proceedings/scipy2010/bib/seabold.bib

Sridhar, L., Sundran, D., Kumari, A., Rashid Bazlur, M.d., Ahmed,A., Sreejith, O. P., et al. (2020). *Development of a new 1 o x1 o (1981-2019) Monthly Gridded Rainfall Data Set Over South Asian Region*. CRS Research Report No.001/2020. Pune: India Meteorological Department.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., et al. (2020). SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nat. Methods* 17, 261–272. doi: 10.1038/s41592-019-0686-2

Weigel, A. P., Liniger, M. A., and Appenzeller, C. (2008). Can multimodel combination really enhance the prediction skill of probabilistic ensemble forecasts? *Quart. J. Roy. Meteor. Soc.* 134, 241–260. doi: 10.1002/qj.210

Willmott, C. J. (1982). Some comments on the evaluation of model performance. *Bull. Am. Meteorol. Soc.* 63, 1309–1313. doi: 10.1175/1520-0477(1982)063<1309:SCOTEO>2.0.CO;2

Wong, S. Y., Yap, K. S., and Li, X. (2020). A new probabilistic output constrained optimization extreme learning machine. *IEEE Access* 8, 28934–28946. doi: 10.1109/ACCESS.2020.2971012