https://doi.org/10.5194/gmd-2021-139
Preprint. Discussion started: 17 June 2021

AI4Water v1.0: An open source python package for modeling hydrological time series using data-driven methods

Ather Abbas[1], Laurie Boithias[2], Yakov Pachepsky[3], Kyunghyun Kim[4], Jong Ahn Chun[5], Kyung Hwa Cho[1]

[1]Urban and Environmental Engineering, Ulsan national institute of science and technology, Ulsan, Republic of Korea.
[2]Geosciences Environnement Toulouse, Université de Toulouse, CNRS, IRD, UPS, 31400 Toulouse, France
[3]Environmental Microbial and Food Safety Laboratory, USDA-ARS, Beltsville, MD, USA.
[4]Watershed and Total Load Management Research Division, National Institute of Environmental Research, Hwangyeong-ro 42, Seogu, Incheon 22689, Republic of Korea
[5]APEC Climate Center, Climate Research Department, Busan, Republic of Korea.

Correspondence to: Jong Ahn Chun (jachun@apcc21.org) Kyung Hwa Cho. (khcho@unist.ac.kr)

Abstract. Machine learning has shown great promise for simulating hydrological phenomena. However, the development of machine learning-based hydrological models requires advanced skills from diverse fields, such as programming and hydrological modeling. Additionally, data pre-processing and post-processing when training and testing machine learning models is a time-intensive process. In this study, we developed a python-based framework that simplifies the process of building and training machine learning-based hydrological models and automates the process of pre-processing of hydrological data and post-processing of model results. Pre-processing utilities assist in incorporating domain knowledge of hydrology in the machine learning model, such as the distribution of weather data into hydrologic response units (HRUs) based on different HRU discretization definitions. The post-processing utilities help in interpreting the model's results from a hydrological point of view. This framework will help increase the application of machine learning-based modeling approaches in hydrological sciences.

1 Introduction

Theory-driven modeling approaches have been traditionally applied to simulate hydrological processes (Remesan and Mathew, 2015). However, with advancements in computation power and data availability, there has been a surge in the application of data-driven approaches to model hydrological processes (Lange and Sippel, 2020). Data-driven approaches that involve time series input data can be used to build several types of hydrological models. Various machine learning approaches have been successfully applied to predict surface water quality (Chen et al., 2020a), estimate stream flow (Kratzert et al., 2018), simulate surface and sub-surface flow (Abbas et al., 2020), forecast evapotranspiration (Ferreira and da Cunha, 2020), and model groundwater flow and transport (Chakraborty et al., 2020). Deep learning, which includes the application of large neural networks, has shown promising results for hydrological modeling (Moshe et al., 2020). A typical workflow of data-driven modeling comprises data collection, pre-processing, model selection, training of the algorithm with optimized hyperparameters, and deployment.

Recent advances in the field of data science have resulted in the growth of Python packages, which assist in accomplishing machine learning and deep learning tasks. According to the latest survey on Kaggle, an online platform for machine learning competitions, the most popular libraries among data scientists are *TensorFlow* (Abadi et al., 2016), *Pytorch* (Paszke et al., 2019), *Scikit-learn* (Pedregosa et al., 2011), and *XGBoost* (Chen and Guestrin, 2016). These libraries have accelerated research in the field of machine learning owing to their simple user interface and robust implementation of difficult algorithms such as back propagation (Chollet, 2018). However, feature engineering, data pre-processing, and post-processing of results are still the most time-consuming tasks in building and testing machine learning models (Cheng et al., 2019). Feature engineering includes modifying existing input data and generating new features based on existing data such that it improves learning using data-driven algorithms. This also incorporates background knowledge and context into the model in order to assist the algorithm in learning the underlying function. Infusion of background knowledge, such as basin architecture (Moshe et al., 2020) and land use (Abbas et al., 2020) in data-driven hydrological modeling leverages the algorithm and enhances its performance (Kratzert et al., 2018). The pre-processing step involves modifying the available data in a form suitable for feeding into the learning algorithm. (Nourani et al., 2020) showed how different smoothing and de-noising functions affect the performance of artificial neural networks for forecasting evaporation. The post-processing step includes the calculation of performance metrics, visualization of results, and interpretation.

Recently, several frameworks have been developed to accelerate the process of building and testing machine learning models, such as *Ludwig* (Molino et al., 2019) and *MLflow* (Zaharia et al., 2018). However, these frameworks are too general and do not deal with the intricacies of time series and hydrological modeling. Several studies have looked at pre-processing, building, training, and post-processing of machine learning models with time series data. These include libraries such as *sktime* (Löning et al., 2019), *Seglearn* (Burns and Whyne, 2018), *Tslearn* (Tavenard et al., 2020), *tsfresh* (Christ et al., 2018), and *pyts* (Faouzi and Janati, 2020). Some libraries have also been developed with a focus on hydrological issues. *Pastas* (Collenteur et al., 2019) is a library dedicated to analyzing groundwater time series data. *NeuralHydrology* (Kratzert et al., 2019) allows the application of several long short-term memory (LSTM)-based models for rainfall runoff modeling. However, most of these libraries either focus on the processing of data and feature extraction from time series or building and training of the model. A framework that combines pre-processing, feature extraction, building and training, post-processing of model results, and interpretation of data-driven models, particularly for solving hydrological problems, is missing.

For the advancement of machine learning in the field of hydrology, experimentation with readily available and fully documented benchmark datasets is required (Leufen et al., 2021). The collection of hydrological data is usually expensive and time-consuming. Several hydrological datasets are publicly available on different online platforms (Coxon et al., 2020). Although these datasets are documented and organized, they are not usually in a form that can be directly used in machine learning algorithms. Therefore, there is a need for a uniform and simplified interface to access and feed hydrological data to machine learning algorithms.

In this study, we developed a new framework for fast and rapid experimentation to develop data-driven hydrological models. In this study, we present *AI4Water*, a Python-based framework that assists in machine learning and deep learning-based modeling with a focus on hydrology. The specific objectives of *AI4Water* were to provide a uniform and simplified interface for 1) access and streaming of freely available datasets to data-driven algorithms, 2) pre-processing of hydrological data, 3) automatic feature extraction from hydrological data, 4) automatic model selection and its hyperparameter optimization, and 5) post-processing of results for visualization and interpretation of models.

2 Workflow and model structure

The core of *AI4Water* is *Model* class, which implements data preparation, building, and training of the model, and makes predictions from the model (Fig. 1). However, *AI4Water* includes several utilities for data pre-processing, feature generation, post-processing and visualization of results, hyperparameter optimization, and model comparison. All of these utilities can be used with *AI4Water* as well as independently. The *Datasets* utility helps in fetching and pre-processing several open-source datasets to be used in machine learning models. The *SpatialProcessing* utility allows distribution of weather data among hydrologic response units (HRUs) using different HRU discretization schemes. The *ETUtil* sub-module helps calculate potential evapotranspiration using various theoretical methods. The *SeqMetricss* module calculates several time series errors for regression and classification problems. *HyperOpt* assists in the implementation of various hyperparameter optimization algorithms. The *Experiment* class can be used to compare different machine learning models. Finally, *AI4Water* has an *Interpret* utility that can be used to interpret the model's results.

The large number of utilities in *AI4Water* increases the number of underlying libraries. *AI4Water* is built on top of *Scikit-learn*, *CatBoost*, *XGBoost*, and *LightGBM* libraries to build classical machine learning models. These models have been used in several hydrological studies (Ni et al., 2020; Huang et al., 2019; Shahhosseini et al., 2021). To build deep learning models using neural networks, *AI4Water* uses a popular deep learning platform called *TensorFlow* (Abadi et al., 2016). A complete list of dependencies for *AI4Water* is presented in Table 1. Table 1 is divided into two parts. The first half shows the minimal requirements for running the basic utilities, which include the model's building and training, and making predictions from the model. The second part of Table 1 consists of an exhaustive list of dependencies required to utilize all the functionalities of *AI4Water*. However, these utilities are optional and do not hinder basic package functionality. Table 1 shows the minimum required version for the underlying libraries. *AI4Water* handles the version conflicts of the underlying libraries, thus making it version-independent of its underlying libraries. This means that the user can use any version greater than the version number given in Table 1.

The success of machine learning is proportional to testing various hypotheses by training and testing machine learning models and analyzing the results (Zaharia et al., 2018). This can quickly lead to a large number of output files. *AI4Water* handles this by automatically saving all the model-related files starting from model creation to pre-processing until post-processing of each output in the respective folders. A detailed output directory structure is shown in Fig. 2. Upon every model run, a directory is created whose name is the date and time when the model is created. This naming convention allows for a simple and distinct directory structure for every new model. This parent directory is called "model path" and contains several sub-folders and files which are related to model configuration, model training, and post-processing of results (Fig. 2a). The results for each target variable are saved in a separate folder. Additionally, the files related to the model's optimized parameters and interpretations are saved in a separate directory. The saved configuration file along with the weights can later be used to reproduce the model's results. In case of hyperparameter optimization, a directory named "hpo path" is created, which consists of several "model paths". Each of these "model paths" correspond to each iteration of the optimization algorithm (Fig. 2b). In case of *Experiments*, when different models are compared, a separate "hpo path" is created for each of the models being compared. Fig. 2c shows the output file structure for an *Experiment* when different machine learning algorithms are compared. This ordered arrangement of results facilitates the fast comparison and analysis of the results.

2.1 Loading and saving models in a readable json file

All features of *AI4Water* can be accomplished using a configuration file. The configuration file (config.json) of *AI4Water* consists of a human-readable json file. All the information regarding pre-processing of data, building and training of the model, predictions, and post-processing results is written in this file. This file is generated every time a new model is built. One of the advantages of this configuration file is that any user can build and run the models without having to write the code explicitly. All examples presented in this study can be run using the corresponding configuration files. Fig. 3 shows examples of the three configuration files. Fig. 3a, shows an LSTM-based model built for rainfall-runoff modeling using the CAMELS (Fowler et al., 2021) dataset. Fig. 3b and c show the usage of the temporal fusion transformer and XGBoost models for the same task. The user can define commands such as the input and output features to use or the training duration for the model. All hyperparameters of the model can also be set using this configuration file.

2.2 Datasets

The first step in building a data-driven hydrological model is to obtain large and diverse data. There have been several efforts by the hydrological science community to build hydrological datasets that are publicly available. For example, for rainfall-runoff modeling, there exists the CAMELS dataset for several countries (Addor et al., 2017). The CAMELS dataset consists of daily weather data and streamflow records for multiple catchments. Another large rainfall runoff dataset is LamaH (Klinger et al., 2021), which consists of observations from 859 catchments in Europe. While the number of such open source datasets is large, the use of these data sources is slow as each database is available on different platforms and implements a different application programming interface (API). A core function of *AI4Water* is to provide a simple and homogeneous API to feed these datasets directly into machine learning models. Fig. 3 shows the usage of the CAMELS_AUS dataset, where the user needs to define only the name of the dataset and the input and output variables. This simple interface will help exploit the use of these datasets. Furthermore, benchmarking open-source datasets will likely accelerate the progress of machine learning in hydrological science. A brief summary of the rainfall-runoff datasets available in *AI4water* is given in Table 2.

2.3 Preprocessing

2.3.1 Exploratory data analysis

A crucial step in data-driven hydrological modeling workflow is the visualization of the data. This step assists in understanding the data, finding outliers, selecting relevant features, and guiding the machine-learning-based modeling process. *AI4Water* provides an *eda* function which can be employed to conduct a comprehensive analysis of input and output data. For example, the correlation plots illustrate which input variables are more correlated with each other. Heatmaps show the amount and position of the missing values. Histogram and box-whisker plots depict the distributions of both the input and output variables. This function can also perform a principal component analysis of the input data and plot the principal components. This helps in understanding the dynamics of the input data and filtering the relevant features.

2.3.2 Transformations

The transformation of data includes scaling, standardizing, and transforming the data onto a different scale. Transforming the data can significantly affect the performance of a data-driven model. The *scikit-learn* library provides several transformation functions such as *minmax*, *standardscaler*, *robust*, and *quantile*. In addition, several other transformation methods such as empirical mode transformation (EMD), ensemble empirical mode transformation (EEMD), wavelet transform (Sang, 2013), and fast Fourier transform (Sang et al., 2009) have been found to improve the performance of hydrological models. *AI4Water* provides a uniform interface for all of these transformation methods under the sub-module *Transformations*. The user can apply any of the available transformations to any of the input features by using a simplified and uniform interface. The predicted features were transformed back after the prediction. Fig. 4 shows a comparison of different transformations using a Taylor plot (Taylor, 2001). These results were generated by modeling in-stream *E. coli* concentrations in a small Laotian catchment (Boithias et al., 2021) using LSTM (Hochreiter and Schmidhuber, 1997). The input data was precipitation, relative humidity, air temperature, wind speed and solar radiation.

2.3.3 Imputation

Missing values are often found in real-world datasets. However, missing data cannot be fed to machine-learning algorithms. *AI4Water* provides various solutions for handling missing data that can be used using the *impute* method. These include 1) using the *pandas* library (McKinney, 2011) *scikit-learn* library-based methods, or 3) using dedicated algorithms to fill the missing input data. The *pandas* library allows the handling of missing values either by filling the missing values using the *fillna* method or interpolating the missing values using the *interpolate* method. Both these methods can be seamlessly used with the *impute* method in *AI4Water*. Several imputation methods for filling missing values are available in the *scikit-learn* library. These methods include *KNNImputer*, *IterativeImputer*, and *SimpleImputer*. *AI4Water* provides a uniform interface for all imputation methods without hindering their functionality.

Several other libraries have been developed that have dedicated algorithms for imputing missing time series data. These include *fancyimpute* (Rubinsteyn and Feldman, 2016) and *transdim* (Chen et al., 2020b). The *fancyimpute* library provides several state-of-the-art algorithms such as *SoftImpute* (Mazumder et al., 2010), *IterativeSVD* (Troyanskaya et al., 2001), *MatrixFactorization*, *NuclearNormMinimization* (Candès and Recht, 2009), and *Biscaler* (Hastie et al., 2015). The *transdim* library provides algorithms based upon neural networks for filling missing data. *AI4Water* provides a simple interface for using these libraries with their full functionalities, using the *impute* method.

2.4 Missing labels

In supervised machine-learning problems, the training data consist of examples. Each example consists of one or more input data and a corresponding label, which is the true value for the given example. Similar to the input data, it is common for the labels to have missing data. Although the missing values in target features can be handled similarly to that of input features, which has been explained in Section 2.3, this can lead to unrealistic results, particularly when the number of missing values is large. *AI4Water* allows the user to exclude examples with missing labels during model training. For multi-output prediction, one can encounter situations in which all target variables are not available for a given example. *AI4Water* allows the user to handle such situations by masking the missing observations during loss calculation. On the other hand, the user can also choose to skip these examples, though this can reduce the number of examples in water quality problems where the number of samples is already very small.

2.5 Resampling

Modeling hydrological processes at high temporal resolutions can result in a large amount of data (Li et al., 2021). Training with this large input data can be computationally expensive. However, temporally coarse input data contain little information. *AI4Water* handles large amount of data by either resampling the data at a lower temporal resolution using the *Resample* class, or by skipping every n-th input data, where 'n' represents the time-step. The later can be achieved by setting the value of the *input_steps* argument to a value greater than 1. The default value of this argument is 1, which results in the use of all input data.

2.6 Feature generation

The incorporation of scientific knowledge into machine learning models is an emerging paradigm for constraining predictions from machine learning models to reality (Wang et al., 2020). The guiding principle of *AI4Water* is to integrate domain-specific knowledge and hydrological data. *AI4Water* automates the calculation of several features and their inputs to the machine learning algorithm. The input data requirement for the calculation of these features is minimal as they are calculated from the raw data. The calculated features are in the form of a time series, which are then directly given as input to machine learning algorithms. The following sections describe the feature generation process in more detail.

2.6.1 Land use change and HRU discretization

In rainfall-runoff modeling, the method of discretization of the HRU plays an important role in many theory-driven models such as the Soil and Water Assessment Tool (SWAT) (Neitsch et al., 2011) and Hydrological Simulation Program FORTRAN (HSPF) (Donigian Jr et al., 1995). An HRU is a building block of a process-driven hydrological model in which all the processes are simulated. The area and formation of an HRU depend on its definition. For example, in the HSPF model, an HRU is defined as a unique land use in a unique sub-basin. On the other hand, the SWAT model considers slope classes and soil type distributions in an HRU. In catchments, which undergo changes in land use over time, the corresponding HRUs also change with time. Temporal changes in HRUs are a major challenge in most process-driven models (Kim et al., 2018). However, it has been shown that machine learning models can easily incorporate land-use changes with time and dynamic HRU calculations (Abbas et al., 2020). *AI4Water* contains a sub-module, *MakeHRUs*, which helps in distributing time series weather data into HRUs using different HRU definitions. Fig. 5 shows two discretization schemes that combine land use, soil type, and sub-basin. However, the user can also add other spatially varying features, such as slope, in the HRU definition. A complete list of the HRU definitions is provided in Table S1. Fig. 6 illustrates the HRU variation with time in a Laotian catchment (Abbas et al., 2020). The HRUs shown in Fig. 6 are defined as a unique land use with a unique soil type. Thus, every HRU has distinct land use and soil characteristics. As there are four land-use types and three soil types in the catchment, the total number of HRUs was 12. We can observe how the area of certain HRUs, e.g., "Alisol_Fallow", decreases with time at the expense of other HRUs (Fig. 6a). The relative contributions of each HRU for the years 2011, 2012, 2013, and 2014 is illustrated in Fig. 6b–e, respectively.

2.6.2 Evapotranspiration

The amount of evapotranspiration is an important factor that affects the total water budget in a catchment. The impact of evapotranspiration process representation in rainfall-runoff models has been studied extensively (Guo et al., 2017). Several potential and reference evapotranspiration calculation methods are available in the literature. *AI4Water* contains sub-module '*ETUtil*' which can be used to calculate the potential evapotranspiration using various methods. These include complex methods such as Penman-Monteith (Allen et al., 1998), which require large input variables, and simplified methods such as

Jensen and Haise (Jensen and Haise, 1965), which only depend on temperature. The *ETUtil* can furthermore calculate potential evapotranspiration at various time intervals, from 1 min to 1 yr. The names of the 22 evapotranspiration methods available in *ETUtil* and their data requirements are summarized in Table S2. The CAMELS Australia dataset (Fowler et al., 2021) comes with pre-calculated potential evapotranspiration using the Morton (Morton, 1983) method. We compared this method with three different potential evapotranspiration calculation methods using *ETUtil*, as depicted in Fig. 7.

2.7 Hyperparameter optimization

The hyperparameters of a machine learning algorithm are the parameters that remain fixed during model training and significantly influence its performance (Chollet, 2018). Thus, the choice of hyperparameters plays an important role in evaluating the performance of machine learning algorithms. Some of the most popular approaches for optimizing hyperparameters are random search, grid search, and the Bayesian approach. Random search involves randomly selecting parameters from the given space for a given number of iterations. Grid search, on the other hand, comprehensively explores all possible combinations of hyperparameters in the hyperparameter space. Although grid search can ensure global minima, the number of iterations increases exponentially with an increase in the number of hyperparameters. This renders the grid search practically unfeasible for deep neural network-based models, which are computationally expensive. The two commonly used Bayesian approaches are Gaussian processes (Snoek et al., 2012) and the tree of Parzen estimators, (TPE) (Bergstra et al., 2011).

The libraries used to implement these algorithms are *hyperopt* (Bergstra et al., 2013), *scikit-optimize* (Fabisch et al., 2018), *optuna* (Akiba et al., 2019), and *scikit-learn* (Pedregosa et al., 2011). These libraries implement different algorithms with different strengths. The *scikit-optimize* library allows the application of the Bayesian optimization approach using Gaussian Processes. The *scikit-learn* library can be used for random and grid-search-based approaches. The *hyperopt* module assists in Bayesian optimization using TPEs. The *HyperOpt* sub-module in *AI4Water* provides a uniform interface to interact with all of the aforementioned libraries. The integration of *HyperOpt* with its underlying modules not only complements the underlying optimization algorithms but also adds additional functionality, such as visualization. For example, the importance of hyperparameters is plotted using the functional analysis of variance (fANOVA) method proposed by (Hutter et al., 2014).

We demonstrate the use of the *HyperOpt* sub-module of *AI4Water* for optimizing the hyperparameters of an LSTM-based neural network for rainfall-runoff modeling. The input data consisted of climate data, whereas the target was streamflow. For this example, we used CAMELS data from a catchment in Australia (Fowler et al., 2021). We compared the performance of random search, grid search, and two Bayesian algorithms based on Gaussian Processes and TPEs. The convergence plots of all four algorithms are shown in Fig. 8. The Bayesian approach using Gaussian processes was found to be the most useful for minimizing the objective function. The objective function was the minimum of the validation loss. We also observed that grid search, despite a large number of iterations, did not perform better than the other three methods.

https://doi.org/10.5194/gmd-2021-139
Preprint. Discussion started: 17 June 2021
© Author(s) 2021. CC BY 4.0 License.

2.8 Model comparison with Experiment

AI4Water consists of an *Experiment* sub-module, which makes it easier to compare different machine learning models. The basic purpose of the *Experiment* class is to compare different models by optimizing their hyperparameters. This is made possible as the *Experiment* class encompasses the *HyperOpt* class, which in turn encompasses the *Model* class (Fig. 9). Thus, the *Experiment* class can be used for combined algorithm selection and hyperparameter optimization (Thornton et al., 2013). The results from the *Experiment* class are organized within an "exp path" directory (Fig. 2). The *Experiment* class can be sub-classed to compare any number and type of models. It consists of three sub-classed experiments: *MLRegressionExperiment*, *MLClassificationExperiment*, and *TransformationExperiment*. The *MLRegressionExperiment* class runs and compares approximately 50 different classical machine learning algorithms for a regression task. The *MLClassificationExperiment* class compares classical machine learning algorithms for a classification problem. The *TransformationExperiment* class can be used to compare the application of different transformation techniques (Sect. 2.3.1) on different input and output features.

We conducted an experiment to compare the performance of classic machine learning algorithms in predicting antibiotic-resistant genes (ARGs) at a recreational beach (Jang et al., 2021b). The results of this experiment are shown in Fig. 10, which compares the correlation coefficients for the training and test sets. It can be seen that some algorithms can yield an R^2 as high as 0.65. Other algorithms provide training R^2 as high as 1.0, which indicates overfitting. In particular, we observed strong overfitting in the case of the decision tree regressor and Gaussian process regressor. It can also be inferred from Fig. 10 that ensemble methods such as AdaBoost (Freund and Schapire, 1997), gradient boosting (Friedman, 2001), bagging (Ho, 1998), extra trees (Geurts et al., 2006), and random forest (Liaw and Wiener, 2002) yield better performance than other methods. We also observed that simple linear models such as Lars, Lasso, and multi-layer perceptron are not able to model the dynamic and complex functions of the ARG distribution at the beach. On the other hand, complex non-linear models such as CATBoost (Prokhorenkova et al., 2018), XGBoost (Chen and Guestrin, 2016), and light gradient boosting machines (Ke et al., 2017) are able to adequately capture dynamic features related to the ARG distribution. We also observed that algorithms with cross-validation performed better than their counterparts without cross-validation.

2.9 Post processing

2.9.1 Visualizing results

The interpretation of the results of machine learning models is an area of active research. For classical machine learning algorithms, interpretation tools include the plotting of decision trees or input feature importance. For neural network-based models, explainability is considered an even bigger challenge. *AI4Water* consists of a sub-module called *Interpret*, which can be used to plot interpretable results. The *Interpret* class takes the trained *AI4Water*'s model as input and plots all possible results, which can help explain the model's behavior. The exact type of plots generated by the *Interpret* sub-module depends on the algorithm used by the model. For neural network-based models, which consist of a layered structure, the *Interpret* sub-module plots all the trained weights, the outputs of each layer, the gradients of weights, and the gradients of the activations of neural networks. This also includes plotting attention weights if the model consists of an attention mechanism. *AI4Water* automatically plots the results of the model when a model is used for prediction. These include the scatter and line plots of each target variable.

We demonstrate this by using a dual-stage attention model (Qin et al., 2017) for daily rainfall-runoff modeling in catchment number 401203 in the CAMELS Australia dataset (Fowler et al., 2021). The input data consisted of evapotranspiration, precipitation, minimum and maximum temperatures, vapor pressure, and relative humidity. The dual-stage model showed significant performance during training (R^2 = 0.93) and test (R^2 = 0.87), as shown in Fig. S1. The dual-stage attention model highlights the importance of the input variables for prediction. The attention weights for each of the input variables are shown in Fig. S2–S4. From these figures, we can infer that the highest attention is given to precipitation followed by evapotranspiration. Furthermore, we also observed that the input of the previous 3–4 days was the most important. This can be attributed to the higher attention weights during the first 3–4 lookback steps in these figures. We also observed periodic changes in attention weights for all input variables, which can be attributed to the seasonal variations of input variables.

2.9.2 Performance metrics

Performance metrics are a vital component of the evaluation framework for machine learning (Botchkarev, 2018). There are two major types of performance metrics related to the evaluation of a model's forecasting ability. These include scale-dependent and scale-independent error metrics. Scale-dependent metrics, such as mean absolute error, provide a good estimate of a single model's performance, but they cannot be used across the models because of their scale dependency (Prestwich et al., 2014). Scale-independent error metrics are more useful when comparing the performance of various models (Hyndman and Koehler, 2006). However, certain scale-independent error metrics cannot be defined, such as percentage errors or relative errors (Hyndman, 2006). The choice of a performance metric to evaluate the model depends on the problem definition and model objectives (Wheatcroft, 2019). *AI4Water* calculates over 100 regression metrics and numerous classification metrics to help the user analyze the general characteristics of the forecasts. These performance metrics are sub-packaged under *SeqMetrics* in *AI4Water*. These metrics are calculated automatically for all the target variables whenever a model is used for prediction using the *predict* method. The metrics are stored in a json file inside the path of the model (errors.json in Fig. 2). The names of the performance metrics calculated by *AI4Water* are listed in Table S3. Additionally, several statistical parameters of the predicted variable were calculated and stored in this json file.

3. Advanced usage

AI4Water was built using the object-oriented programming (OOP) paradigm. Its core logic was implemented by the *Model* class. The use of OOP allows a user to customize any steps of model building, training, or testing by sub-classing the *Model* class. This may include the implementation of a custom training loop or a customized loss function. Similarly, the pre-processing and data preparation steps implemented in the *Model* class can also be overwritten for specific usages. Additionally, *AI4Water* exposes the underlying machine learning libraries such as *TensorFlow* and *scikit-learn* to the user. Thus, users can directly use these libraries and implement the desired configuration. However, this requires a deeper understanding of the underlying libraries.

4. Test coverage and continuous integration

AI4Water version 1.0 was tested with continuous integration tools with GitHub Actions to ensure that it passes all the written tests and can be installed on computers. The tests were conducted on Windows and Linux-based operating systems. In addition, we tested the package on Python versions 3.6, 3.7, and 3.8. The package was also tested with *TensorFlow* versions 1.14 and above.

5. Limitations and scope for expansion

- The current version of *AI4Water* was designed only for supervised learning problems. However, there has been growing interest in unsupervised machine learning models, such as generative adversarial networks (GANs) and reinforcement learning. GANs have been shown to exhibit high performance for time series-related tasks such as filling missing data (Luo et al., 2018) or generating new high-resolution data (Chen et al., 2019). This aspect of GANs can be useful in water quality modeling, where data collection is costly and missing observations are common. Reinforcement learning can be applied to optimal policy design in hydrological systems, such as scheduling the release of water from a dam (Sit et al., 2020).
- Another limitation of *AI4Water* is the choice of the underlying libraries to build neural networks. Currently, *AI4Water* employs *TensorFlow* platform to build neural network layers in deep learning models. However, there are several other platforms for building neural network-based models, such as *PyTorch* (Paszke et al., 2019). Support for these platforms will increase the diversity of models that can be implemented using these platforms.
- *AI4Water* was designed for the rapid testing and experimentation of deep learning models. However, it should be noted that the current version of the framework is not suitable for the deployment of deep learning models in production.
- As all the options to use *AI4Water* are accommodated in a configuration file, this makes it suitable for developing a graphical user interface (GUI. Adding GUIs will further widen the user-base of *AI4Water* by being accessible to non-programmers.

https://doi.org/10.5194/gmd-2021-139
Preprint. Discussion started: 17 June 2021

6. Conclusion

Modeling hydrological processes using machine learning requires the development of a data pipeline that encompasses data retrieval, feature extraction, visualization prior to training, building, training, and testing the machine learning model, and visualization and interpretation of the model's results. The *AI4Water* software introduced in this work was designed to facilitate the development, reuse, and reproducibility of machine learning models for applications in hydrology. *AI4Water* was designed to integrate the domain-specific aspects of hydrological modeling with the professional level of machine learning and data processing software already developed and used by the Python community. We demonstrated the applicability of *AI4Water* with supervised learning examples related to hydrological modeling. Further development of the package is suggested with new features that may make *AI4Water* more versatile. The platform is expected to be practical for a wide range of users interested in hydrological modeling.

Code and data availability

The *AI4Water* source code can be found in a publicly available GitHub repository (https://github.com/AtrCheema/AI4Water) and its version 1.0 is archived at https://zenodo.org/record/4904517. The user manual is built into the source code *Docstring* and compiled into a "read the docs" web page (https://ai4water.readthedocs.io/en/latest/) using the MKDocs (Christine, 2014) software. The Jupyter notebooks replicating the examples described in the manuscript are available in the "examples" directory.

Team list

Ather Abbas
Laurie Boithias
Yakov Pachepsky
Kyunghyun Kim
Jong Ahn Chun
Kyung Hwa Cho

Author contribution

Ather Abbas: Conceptualization, code development, writing draft
Laurie Boithias: Review and editing
Yakov Pachepsky: Review and editing
Kyunghyun Kim: Review and editing

Jong Ahn Chun: Review and editing, supervision

Kyung Hwa Cho: Conceptualization, Funding acquisition, supervision, review and edition

Competing interests

The authors declare that they have no conflict of interest.

Acknowledgement

This study was supported by the Korea Environment Industry & Technology Institute (KEITI) through the Aquatic Ecosystem Conservation Research Program, funded by the Korea Ministry of Environment (MOE) (2020003030003).

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., and Zheng, X.: TensorFlow: A system for large-scale machine learning, in: Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, 2016.

Abbas, A., Baek, S., Kim, M., Ligaray, M., Ribolzi, O., Silvera, N., Min, J. H., Boithias, L., and Cho, K. H.: Surface and sub-surface flow estimation at high temporal resolution using deep neural networks, 590, https://doi.org/10.1016/j.jhydrol.2020.125370, 2020.

Addor, N., Newman, A. J., Mizukami, N., and Clark, M. P.: The CAMELS data set: Catchment attributes and meteorology for large-sample studies, 21, https://doi.org/10.5194/hess-21-5293-2017, 2017.

Akiba, T., Sano, S., Yanase, T., Ohta, T., and Koyama, M.: Optuna: A Next-generation Hyperparameter Optimization Framework, in: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, https://doi.org/10.1145/3292500.3330701, 2019.

Allen, R. G., Pereira, L. S., Raes, D., and Smith, M.: Crop evapotranspiration – Guidelines for computing crop water requirements. – FAO Irrigation and drainage paper 56 / Food and Agriculture Organization of the United Nations., 1998.

Bergstra, J., Bardenet, R., Bengio, Y., and Kégl, B.: Algorithms for hyper-parameter optimization, in: Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011, NIPS 2011, 2011.

Bergstra, J., Yamins, D., and Cox, D. D.: Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures, in: 30th International Conference on Machine Learning, ICML 2013, 2013.

Botchkarev, A.: Performance metrics (error measures) in machine learning regression, forecasting and prognostics: Properties and typology, 2018.

Burns, D. M. and Whyne, C. M.: Seglearn: A python package for learning sequences and time series, 19, 2018.

Candès, E. J. and Recht, B.: Exact matrix completion via convex optimization, 9, https://doi.org/10.1007/s10208-009-9045-5, 2009.

Chakraborty, M., Sarkar, S., Mukherjee, A., Shamsudduha, M., Ahmed, K. M., Bhattacharya, A., and Mitra, A.: Modeling regional-scale groundwater arsenic hazard in the transboundary Ganges River Delta, India and Bangladesh: Infusing physically-based model with machine learning, 748, https://doi.org/10.1016/j.scitotenv.2020.141107, 2020.

Cheng, Y., Li, D., Guo, Z., Jiang, B., Lin, J., Fan, X., Geng, J., Yu, X., Bai, W., Qu, L., Shu, R., Cheng, P., Xiong, Y., and Wu, J.: DLBooster: Boosting end-to-end deep learning workflows with offloading data preprocessing pipelines, in: ACM International Conference Proceeding Series, https://doi.org/10.1145/3337821.3337892, 2019.

Chen, H., Zhang, X., Liu, Y., and Zeng, Q.: Generative adversarial networks capabilities for super-resolution reconstruction of weather radar echo images, 10, https://doi.org/10.3390/atmos10090555, 2019.

Chen, K., Chen, H., Zhou, C., Huang, Y., Qi, X., Shen, R., Liu, F., Zuo, M., Zou, X., Wang, J., Zhang, Y., Chen, D., Chen, X., Deng, Y., and Ren, H.: Comparative analysis of surface water quality prediction performance and identification of key water parameters using different machine learning models based on big data, 171, https://doi.org/10.1016/j.watres.2019.115454, 2020a.

Chen, T. and Guestrin, C.: XGBoost: A scalable tree boosting system, in: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, https://doi.org/10.1145/2939672.2939785, 2016.

Chen, X., Yang, J., and Sun, L.: A nonconvex low-rank tensor completion model for spatiotemporal traffic data imputation, 117, https://doi.org/10.1016/j.trc.2020.102673, 2020b.

Chollet, F.: Deep Learning with Python, Manning, 2018.

Christ, M., Braun, N., Neuffer, J., and Kempa-Liehr, A. W.: Time Series FeatuRe Extraction on basis of Scalable Hypothesis tests (tsfresh – A Python package), 307, https://doi.org/10.1016/j.neucom.2018.03.067, 2018.

Collenteur, R. A., Bakker, M., Caljé, R., Klop, S. A., and Schaars, F.: Pastas: Open Source Software for the Analysis of Groundwater Time Series, 57, https://doi.org/10.1111/gwat.12925, 2019.

Coxon, G., Addor, N., Bloomfield, J. P., Freer, J., Fry, M., Hannaford, J., Howden, N. J. K., Lane, R., Lewis, M., Robinson, E. L., Wagener, T., and Woods, R.: CAMELS-GB: hydrometeorological time series and landscape attributes for 671 catchments in Great Britain, 12, https://doi.org/10.5194/essd-12-2459-2020, 2020.

Donigian Jr, A. S., Bicknell, B. R., Imhoff, J. C., and Singh, V. P.: Hydrological Simulation Program-Fortran (HSPF), 1995.

Fabisch, T. H., MechCoder, Louppe, G., Shcherbatyi, I., Fcharras, Vinícius, Z., Cmmalone, Schröder, C., Nel215, Campos, N., Young, T., Cereda, S., Fan, T., Rene-rex, Shi, K. (KJ), Schwabedal, J., Carlosdanielcsantos, Hvass-Labs, and Mik: scikit-optimize/scikit-optimize: v0.5.2, 2018.

https://doi.org/10.5194/gmd-2021-139
Preprint. Discussion started: 17 June 2021

Faouzi, J. and Janati, H.: Pyts: A python package for time series classification, 21, 2020.

Ferreira, L. B. and da Cunha, F. F.: New approach to estimate daily reference evapotranspiration based on hourly temperature and relative humidity using machine learning and deep learning, 234, https://doi.org/10.1016/j.agwat.2020.106113, 2020.

Fowler, K. J. A., Acharya, S. Chandra., Addor, Nans., Chou, Chihchung. P., and Murray C.: CAMELS-AUS: Hydrometeorological time series and landscape attributes for 222 catchments in Australia, 2021.

Freund, Y. and Schapire, R. E.: A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting, 55, https://doi.org/10.1006/jcss.1997.1504, 1997.

Friedman, J. H.: Greedy function approximation: A gradient boosting machine, 29, https://doi.org/10.1214/aos/1013203451, 2001.

Geurts, P., Ernst, D., and Wehenkel, L.: Extremely randomized trees, 63, https://doi.org/10.1007/s10994-006-6226-1, 2006.

Guo, D., Westra, S., and Maier, H. R.: Impact of evapotranspiration process representation on runoff projections from conceptual rainfall-runoff models, 53, https://doi.org/10.1002/2016WR019627, 2017.

Hastie, T., Mazumder, R., Lee, J. D., and Zadeh, R.: Matrix completion and low-rank SVD via fast alternating least squares, 16, 2015.

Hochreiter, Sepp. and Schmidhuber, Jürgen.: Long Short-term Memory, 9, 1735–1780, https://doi.org/https://doi.org/10.1162/neco.1997.9.8.1735, 1997.

Ho, T. K.: The random subspace method for constructing decision forests, 20, https://doi.org/10.1109/34.709601, 1998.

Huang, G., Wu, L., Ma, X., Zhang, W., Fan, J., Yu, X., Zeng, W., and Zhou, H.: Evaluation of CatBoost method for prediction of reference evapotranspiration in humid regions, 574, https://doi.org/10.1016/j.jhydrol.2019.04.085, 2019.

Hutter, F., Hoos, H., and Leyton-Brown, K.: An efficient approach for assessing hyperparameter importance, in: 31st International Conference on Machine Learning, ICML 2014, 2014.

Hyndman, R. J.: Another look at forecast-accuracy metrics for intermittent demand, 4, 2006.

Hyndman, R. J. and Koehler, A. B.: Another look at measures of forecast accuracy, 22, https://doi.org/10.1016/j.ijforecast.2006.03.001, 2006.

Jang, J., Abbas, A., Kim, M., Shin, J., Kim, Y. M., and Cho, K. H.: Prediction of antibiotic-resistance genes occurrence at a recreational beach with deep learning models, 196, https://doi.org/10.1016/j.watres.2021.117001, 2021a.

Jang, J., Abbas, A., Kim, M., Shin, J., Kim, Y. M., and Cho, K. H.: Prediction of antibiotic-resistance genes occurrence at a recreational beach with deep learning models, 196, https://doi.org/10.1016/j.watres.2021.117001, 2021b.

Jensen, M. E. and Haise, H. R.: Closure to "Estimating Evapotranspiration from Solar Radiation," 91, https://doi.org/10.1061/jrcea4.0000342, 1965.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., and Liu, T. Y.: LightGBM: A highly efficient gradient boosting decision tree, in: Advances in Neural Information Processing Systems, 2017.

Kim, M., Boithias, L., Cho, K. H., Sengtaheuanghoung, O., and Ribolzi, O.: Modeling the Impact of Land Use Change on Basin-scale Transfer of Fecal Indicator Bacteria: SWAT Model Performance, 47, https://doi.org/10.2134/jeq2017.11.0456, 2018.

Klinger, Christoph., Schulz, Karsten., and Herrnegger, Mathew.: LamaH | Large-Sample Data for Hydrology and Environmental Sciences for Central Europe, 2021.

Kratzert, F., Klotz, D., Brenner, C., Schulz, K., and Herrnegger, M.: Rainfall-runoff modelling using Long Short-Term Memory (LSTM) networks, 22, https://doi.org/10.5194/hess-22-6005-2018, 2018.

Kratzert, F., Herrnegger, M., Klotz, D., Hochreiter, S., and Klambauer, G.: NeuralHydrology – Interpreting LSTMs in Hydrology, in: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 11700 LNCS, https://doi.org/10.1007/978-3-030-28954-6_19, 2019.

Lange, H. and Sippel, S.: Machine Learning Applications in Hydrology, https://doi.org/10.1007/978-3-030-26086-6_10, 2020.

Leufen, L. H., Kleinert, F., and Schultz, M. G.: MLAir (v1.0)-a tool to enable fast and flexible machine learning on air data time series, 14, https://doi.org/10.5194/gmd-14-1553-2021, 2021.

Liaw, A. and Wiener, M.: Classification and Regression with Random Forest, 2, 2002.

Li, W., Kiaghadi, A., and Dawson, C.: High temporal resolution rainfall–runoff modeling using long-short-term-memory (LSTM) networks, 33, https://doi.org/10.1007/s00521-020-05010-6, 2021.

Löning, M., Kazakov, V., Bagnall, A., Lines, J., Ganesh, S., and Király, F. J.: Sktime: A unified interface for machine learning with time series, 2019.

Luo, Y., Cai, X., Zhang, Y., Xu, J., and Yuan, X.: Multivariate time series imputation with generative adversarial networks, in: Advances in Neural Information Processing Systems, 2018.

Mazumder, R., Hastie, T., and Tibshirani, R.: Spectral regularization algorithms for learning large incomplete matrices, 11, 2010.

McKinney, W.: pandas: a Foundational Python Library for Data Analysis and Statistics, 2011.

Molino, P., Dudin, Y., and Miryala, S. S.: Ludwig: A type-based declarative deep learning toolbox, 2019.

Morton, F. I.: Operational estimates of areal evapotranspiration and their significance to the science and practice of hydrology, 66, https://doi.org/10.1016/0022-1694(83)90177-4, 1983.

Moshe, Z., Metzger, A., Elidan, G., Kratzert, F., Nevo, S., and El-Yaniv, R.: Hydronets: Leveraging river structure for hydrologic modeling, 2020.

Neitsch, S. L., Arnold, J. G., Kiniry, J. R., and Williams, J. R.: Soil & Water Assessment Tool Theoretical Documentation Version 2009, https://doi.org/10.1016/j.scitotenv.2015.11.063, 2011.

Ni, L., Wang, D., Wu, J., Wang, Y., Tao, Y., Zhang, J., and Liu, J.: Streamflow forecasting using extreme gradient boosting model coupled with Gaussian mixture model, 586, https://doi.org/10.1016/j.jhydrol.2020.124901, 2020.

Nourani, V., Sayyah-Fard, M., Alami, M. T., and Sharghi, E.: Data pre-processing effect on ANN-based prediction intervals construction of the evaporation process at different climate regions in Iran, 588, https://doi.org/10.1016/j.jhydrol.2020.125078, 2020.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S.: PyTorch: An imperative style, high-performance deep learning library, in: Advances in Neural Information Processing Systems, 2019.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, É.: Scikit-learn: Machine learning in Python, 12, 2011.

Prestwich, S., Rossi, R., Armagan Tarim, S., and Hnich, B.: Mean-based error measures for intermittent demand forecasting, 52, https://doi.org/10.1080/00207543.2014.917771, 2014.

Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A. V., and Gulin, A.: Catboost: Unbiased boosting with categorical features, in: Advances in Neural Information Processing Systems, 2018.

Qin, Y., Song, D., Cheng, H., Cheng, W., Jiang, G., and Cottrell, G. W.: A dual-stage attention-based recurrent neural network for time series prediction, in: IJCAI International Joint Conference on Artificial Intelligence, https://doi.org/10.24963/ijcai.2017/366, 2017.

Remesan, R. and Mathew, J.: Hydrological Data Driven Modelling, https://doi.org/10.1007/978-3-319-09235-5, 2015.

Rubinsteyn, Alex. and Feldman, Sergey.: fancyimpute: An Imputation Library for Python, https://github.com/iskandr/fancyimpute, 2016.

Sang, Y. F.: A review on the applications of wavelet transform in hydrology time series analysis, https://doi.org/10.1016/j.atmosres.2012.11.003, 2013.

Sang, Y. F., Wang, D., Wu, J. C., Zhu, Q. P., and Wang, L.: The relation between periods' identification and noises in hydrologic series data, 368, https://doi.org/10.1016/j.jhydrol.2009.01.042, 2009.

Shahhosseini, M., Hu, G., Huber, I., and Archontoulis, S. v.: Coupling machine learning and crop modeling improves crop yield prediction in the US Corn Belt, 11, https://doi.org/10.1038/s41598-020-80820-1, 2021.

Sit, M., Demiray, B. Z., Xiang, Z., Ewing, G. J., Sermet, Y., and Demir, I.: A comprehensive review of deep learning applications in hydrology and water resources, 82, https://doi.org/10.2166/wst.2020.369, 2020.

Snoek, J., Larochelle, H., and Adams, R. P.: Practical Bayesian optimization of machine learning algorithms, in: Advances in Neural Information Processing Systems, 2012.

Tavenard, R., Faouzi, J., Vandewiele, G., Divo, F., Androz, G., Holtz, C., Payne, M., Yurchak, R., Rußwurm, M., Kolar, K., and Woods, E.: Tslearn, a machine learning toolkit for time series data, 21, 2020.

Taylor, K. E.: Summarizing multiple aspects of model performance in a single diagram, 106, https://doi.org/10.1029/2000JD900719, 2001.

Thornton, C., Hutter, F., Hoos, H. H., and Leyton-Brown, K.: Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms, in: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, https://doi.org/10.1145/2487575.2487629, 2013.

Troyanskaya, O., Cantor, M., Sherlock, G., Brown, P., Hastie, T., Tibshirani, R., Botstein, D., and Altman, R. B.: Missing value estimation methods for DNA microarrays, 17, https://doi.org/10.1093/bioinformatics/17.6.520, 2001.

Wang, L., Chen, J., and Marathe, M.: TDEFSI: Theory-guided Deep Learning-based Epidemic Forecasting with Synthetic Information, 6, https://doi.org/10.1145/3380971, 2020.

Wheatcroft, E.: Interpreting the skill score form of forecast performance metrics, 35, https://doi.org/10.1016/j.ijforecast.2018.11.010, 2019.

Zaharia, M., Chen, A., Davidson, A., Ghodsi, A., Hong, S. A., Konwinski, A., Murching, S., Nykodym, T., Ogilvie, P., Parkhe, M., Xie, F., and Zumar, C.: Accelerating the Machine Learning Lifecycle with MLflow, 2018.

Tables

Table 1. Complete list of third-party Python libraries, which are used by *AI4Water*. The first half the table enlists those libraries which are required while the second half consists of those libraries which are optional.

Library Name	Version	Usage
numpy	> 0.15	array processing
pandas	> 0.22	array processing
matplotlib	>0.12	visualization
h5py	>0.15	storage
seaborn	>0.18	visualization
scikit-learn	> 0.22	building classical machine learning models
tensorflow	> 1.14	building layers of neural networks
xgboost	>1.2	implementing XGBoost based algorithms
catboost	>0.23	implementing CatBoost based algorithms
lightgbm	>1.5	implementing Light Gradient Boost based algorithms
tpot	>4.5	implementing tpot algorithm
imageio	>2.3	spatial processing of shape files
shapely	>0.15	spatial processing of shape files

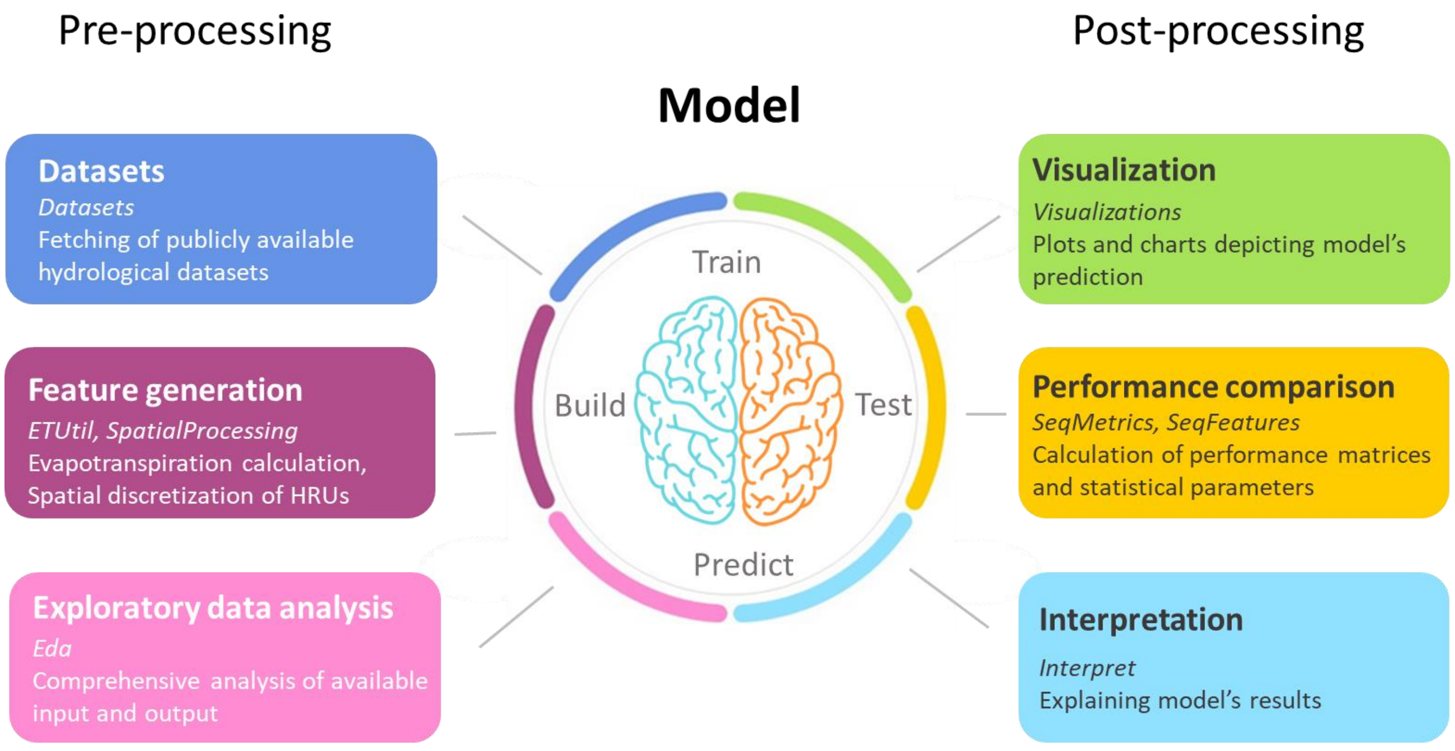
pyshp	>0.45	spatial processing of shape files
plotly	>0.23	extended visualization

535 **Table 2. Name and attributes of open source datasets included in *AI4Water*.**

Dataset Name	**Number of catchments**	**Number of Variables**	**Number of Observations**	**Location**
CAMELS_AUS	222	23	21184	Australia
CAMELS_BR	593	17	14245	Brazil
CAMELS_CL	516	12	38374	Chile
CAMELS_GB	671	10	16436	Britain
CAMELS_US	877	33	12784	United States of America
LamaH	859	5	12775	Europe

https://doi.org/10.5194/gmd-2021-139
Preprint. Discussion started: 17 June 2021

Figures


Pre-processing
Post-processing
Model
Datasets
Datasets
Fetching of publicly available hydrological datasets
Feature generation
ETUtil, SpatialProcessing
Evapotranspiration calculation, Spatial discretization of HRUs
Exploratory data analysis
Eda
Comprehensive analysis of available input and output
Train
Build
Test
Predict
Visualization
Visualizations
Plots and charts depicting model's prediction
Performance comparison
SeqMetrics, SeqFeatures
Calculation of performance matrices and statistical parameters
Interpretation
Interpret
Explaining model's results

540 **Figure 1: Conceptual framework of hydrological modeling using *AI4Water*. AI4Water consists of modules for pre-processing and post-processing. The names of the modules are written in italic. The pre-processing steps involve collecting data, conducting exploratory data analysis on data, and generating new features from the data. The core of the model consists of building, training, and predicting. After this step, the predicted steps are used for visualization, performance comparison, and model interpretation.**

https://doi.org/10.5194/gmd-2021-139
Preprint. Discussion started: 17 June 2021

545

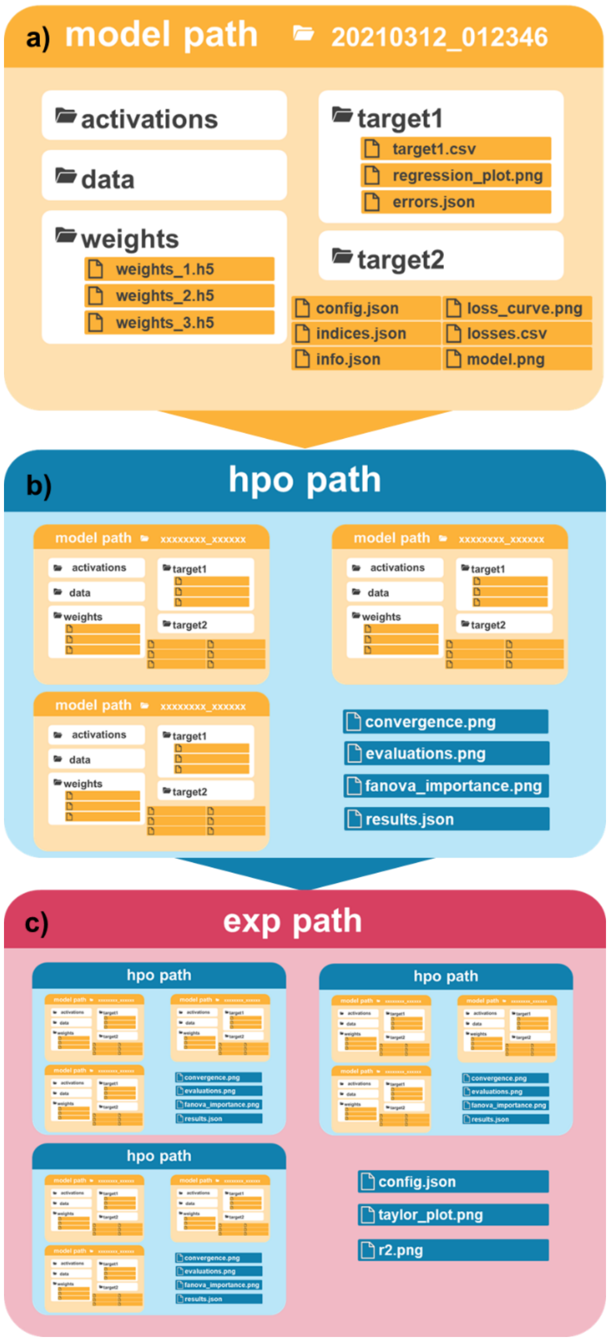


Figure 2: Output directory structure of *AI4Water*. A "model path" (a) is created upon creation of a new model. An "hpo path" (b) is created during hyperparameter optimization. An "exp path" (c) is created when several models are compared during an experiment. The "hpo path" consists of several "model paths" and an "exp path" consists of several "hpo paths".

https://doi.org/10.5194/gmd-2021-139
Preprint. Discussion started: 17 June 2021

(a)

```
{
  “model”: {“layers”: {“LSTM”: {“config”: {“units”: 64}}}},
  “inputs”: [‘et_mortan’, ‘precipitation’, ‘tmax’, ‘tmin’],
  “outputs”: [‘streamflow’],
  “data”: {“name”: “CAMELS_AUS”: station”: 224206}
}
```

(b)

```
{
  “model”: {“layers”: {“TFT”: {“config”: {“hidden_units”: 64, “encoder_steps”: 20, “decoder_steps”: 7}}}},
  “inputs”: [‘et_mortan’, ‘precipitation’, ‘tmax’, ‘tmin’],
  “outputs”: [‘streamflow’],
  “data”: {“name”: “CAMELS_AUS”: station”: 224206}
}
```

(c)

```
{
  “model”: {“XGBoostRegressor”: {“n_estimators”: 64, “max_depth”: 20}},
  “inputs”: [‘et_mortan’, ‘precipitation’, ‘tmax’, ‘tmin’],
  “outputs”: [‘streamflow’],
  “data”: {“name”: “CAMELS_AUS”: station”: 224206}}
}
```

Figure 3: Examples of declarative model definition in a config.json file. a) shows an example of an LSTM-based model using the CAMELS_AUS data (Fowler et al., 2020). b) and c) show contents of configuration file for using temporal fusion transformer (Lim et al., 2020) and XGBoost (Chen et al., 2018) for rainfall-runoff modeling using CAMELS_AUS data, respectively.

https://doi.org/10.5194/gmd-2021-139
Preprint. Discussion started: 17 June 2021

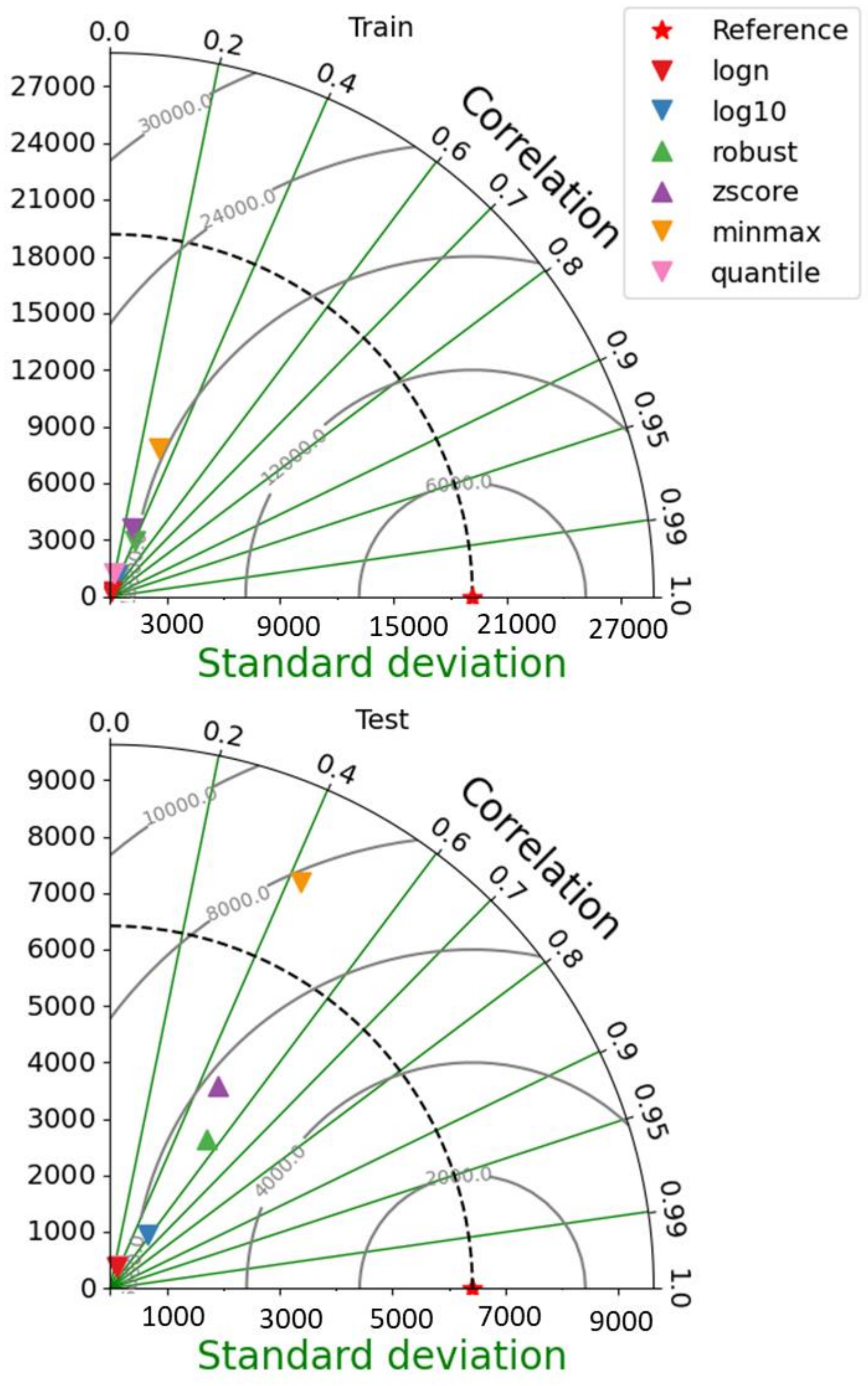
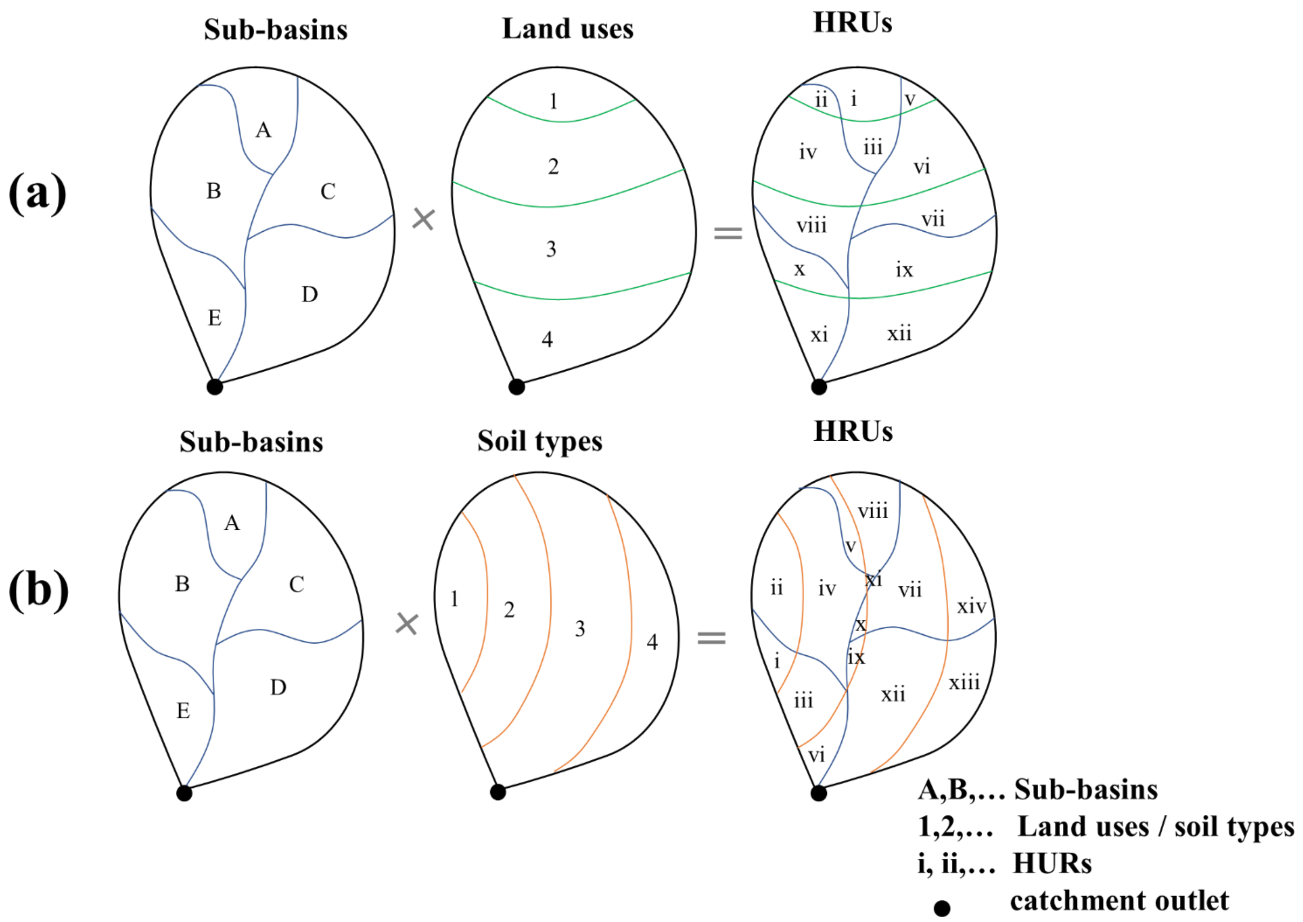
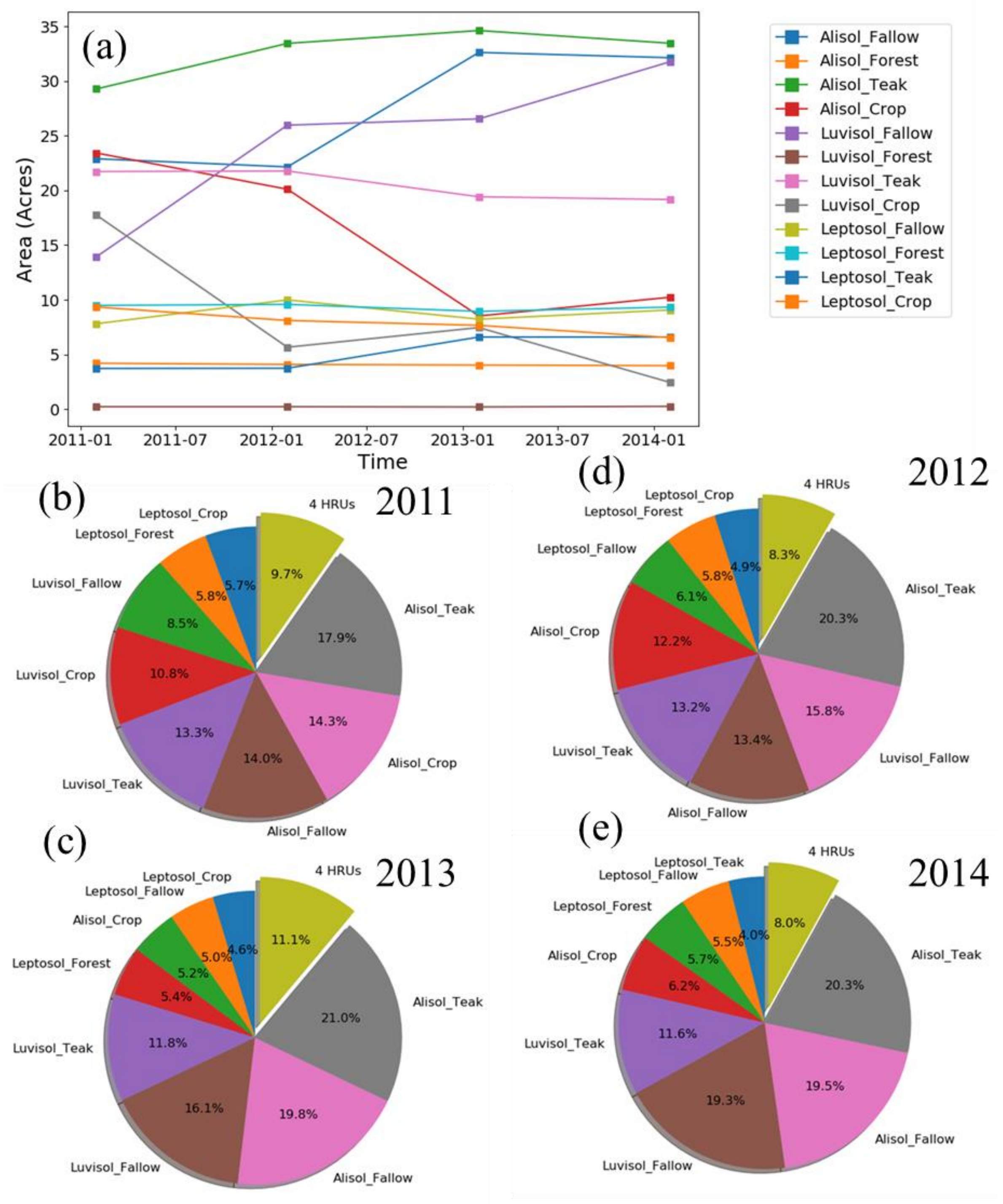


Figure 4: Comparison of different transformations of output data on the performance of a neural network on the simulation of in-stream *E. coli* concentration in a watershed in Lao PDR.



560
Figure 5: Example of HRU discretization schemes by combining a): sub-basins and land uses and b) by combining sub-basins and soil types.

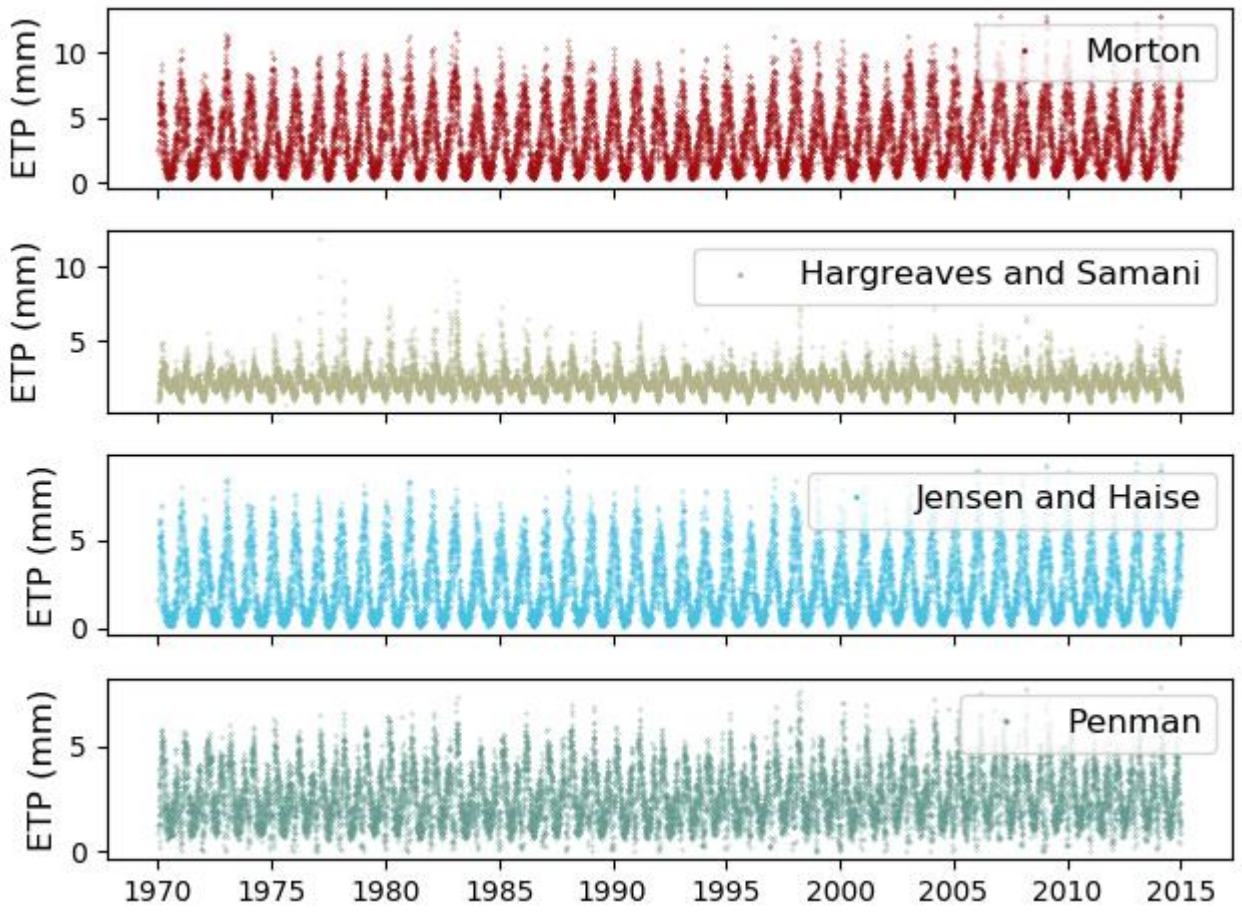
https://doi.org/10.5194/gmd-2021-139
Preprint. Discussion started: 17 June 2021
© Author(s) 2021. CC BY 4.0 License.



565 **Figure 6: Discretization of a catchment in Loas** (Boithias et al., 2021) **according to the HRU definition of "unique land use in unique soil". The catchment consists of three soil types and four land use types. The soil types are Alisol, Luivsol and Leptosol while the**

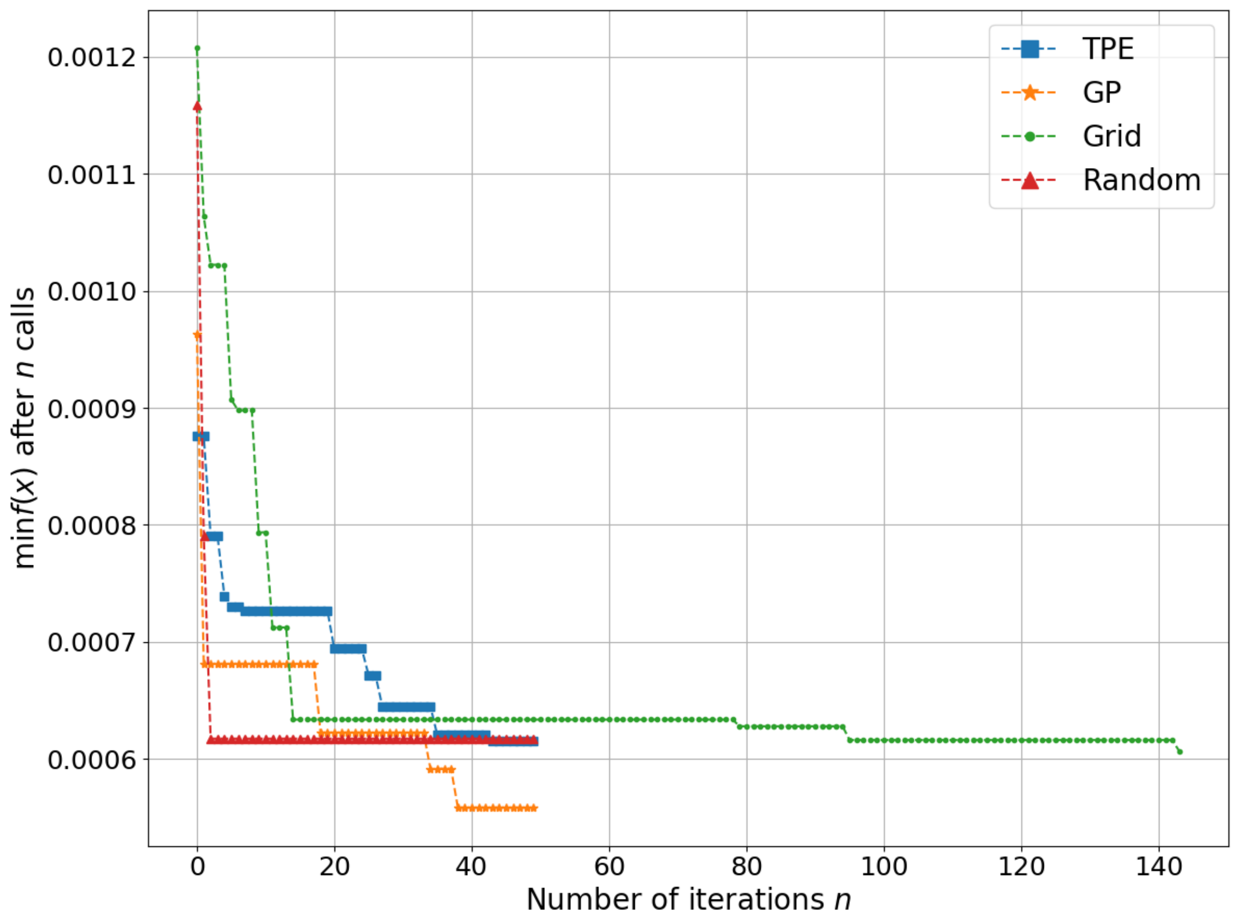
https://doi.org/10.5194/gmd-2021-139
Preprint. Discussion started: 17 June 2021
© Author(s) 2021. CC BY 4.0 License.

land use types are Fallow, Forest, Teak, and Crop. The combination of soil types and land use types results in 12 distinct HRUs. (a) shows annual variation of these 12 land use types while (b)–(e) show the percentage area of HRUs in the catchment in 2011, 2012, 2013, and 2014, respectively.



570 **Figure 7: Comparison of various evapotranspiration methods for the CAMELS_AUS dataset. CAMELS_AUS dataset comes with Morton method while the remaining three methods are calculated by *ETUtil* sub-module of *AI4Water*.**

https://doi.org/10.5194/gmd-2021-139
Preprint. Discussion started: 17 June 2021



575 **Figure 8: Comparison of four optimization algorithms for optimizing hyperparameters of an LSTM-based model for rainfall-runoff modeling. GP represents Bayesian with Gaussian Processes while TPE stands for tree of Parzen estimators. Grid and Random stand for grid search and random search-based optimization, respectively. The x-axis shows the number of function evaluations while min f(x) in the y-axis represents the objective function, which takes x hyperparameters and returns the minimum of validation loss.**

https://doi.org/10.5194/gmd-2021-139
Preprint. Discussion started: 17 June 2021

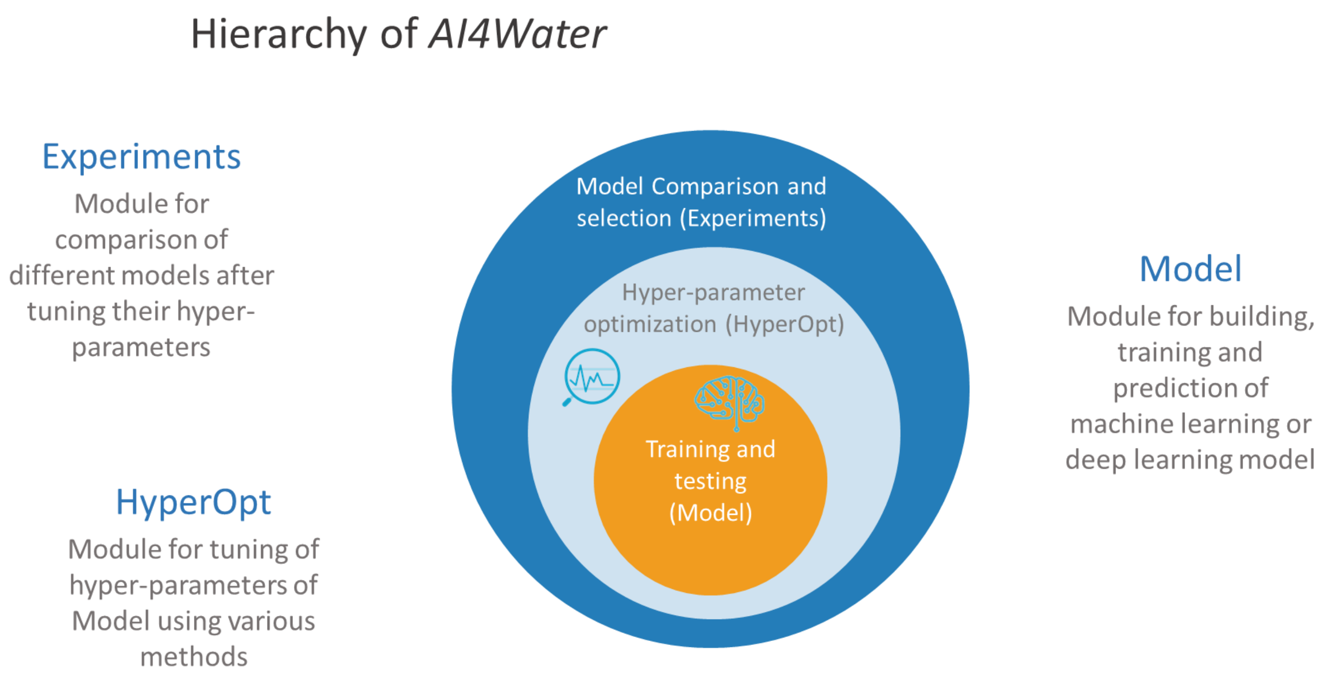
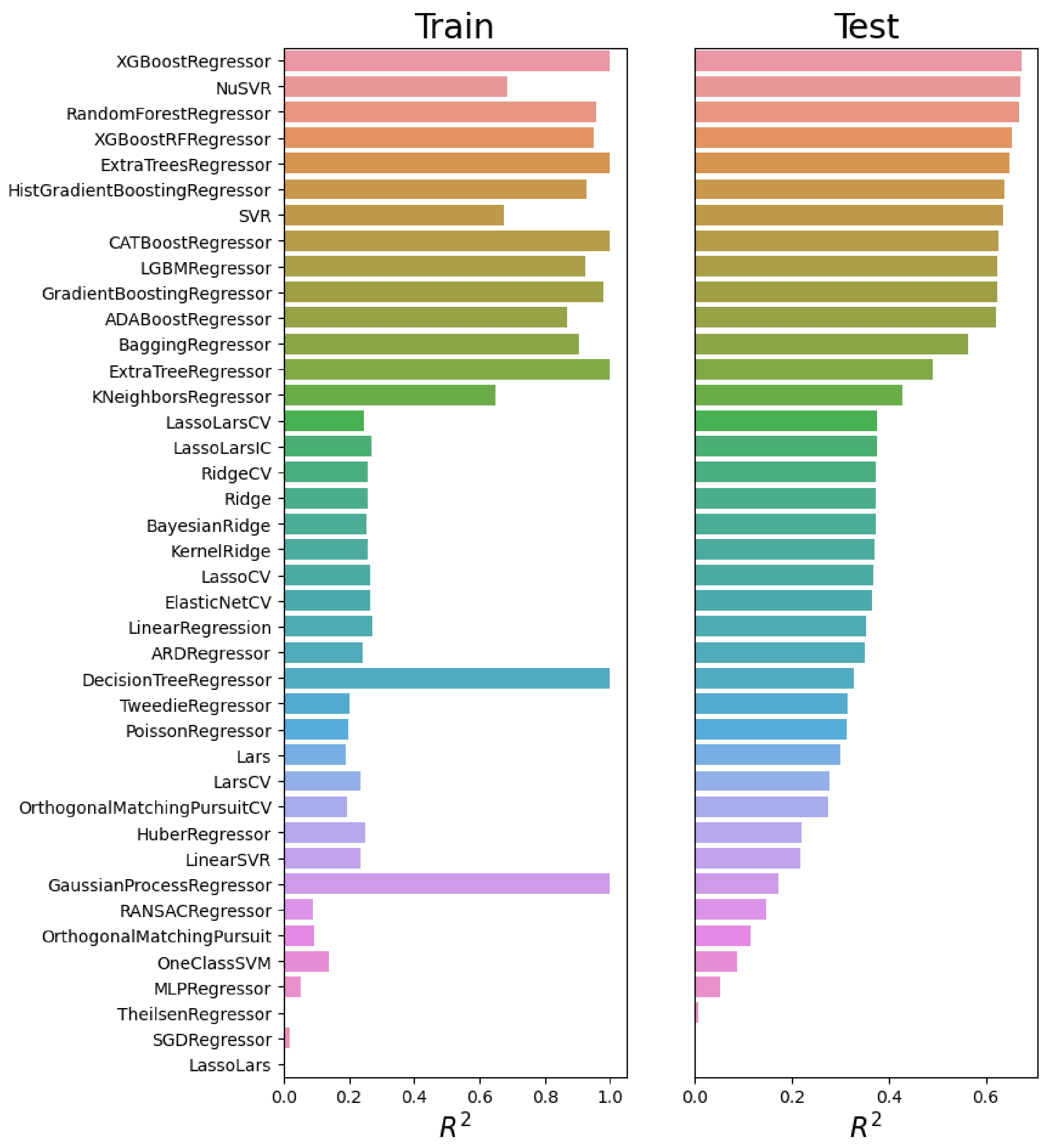

Hierarchy of *AI4Water*
Experiments
Module for comparison of different models after tuning their hyper-parameters
HyperOpt
Module for tuning of hyper-parameters of Model using various methods
Model Comparison and selection (Experiments)
Hyper-parameter optimization (HyperOpt)
Training and testing (Model)
Model
Module for building, training and prediction of machine learning or deep learning model

Figure 9: Hierarchy of model building and comparison in *AI4Water*. The Model involves building, training, and prediction. The hyperparameter optimization step iterates over *Model* until the best hyperparameters are obtained. *Experiments* are then designed to compare performance of different model architectures after tuning their hyperparameters.



585 **Figure 10: An ‘Experiment’ which compares ARG prediction performance at a recreational beach in Korea, using various machine learning algorithms. The y-axis represents abbreviations of the algorithms. The complete names of algorithms are given in Table S4. The hyperparameters of each of the algorithm were optimized during the ‘Experiment’.**