The manuscript presents a new framework for fast and rapid experimentation to develop data-driven hydrological models. The manuscript provides an important tool for hydrologic community to utilize machine learning models without expert knowledge. However, there are some minor details missing or not clear in the manuscript.

Response: We thank the reviewer for the valuable comments. We have revised the code and manuscript according to the guidelines of the reviewer. Please find the responses to each of the reviewer's comments below. We have updated the code and archived the latest release of the code at Zenodo at the URL https://zenodo.org/record/5595680. We have updated the documentation of the code, which is available at https://ai4water.readthedocs.io/en/latest/. The code to reproduce the figures shown in the manuscript is given in the "examples" folder of the code repository.

Methods section is not easy to follow. It is not clear how the python library is developed, which dependencies are required, underlying methods and protocols for data transfer, processing and visualization.

Response: We have added a visual diagram (Fig. 3), which shows the code architecture and the interaction between the different modules of *AI4Water*. We have also changed the numbering and arrangement of the chapters to match the conceptual flowchart shown in Fig. 3. All the sub-modules are now part of Chapter 3. We have also added details about the dependencies, underlying methods of data transfer, and technical details of the framework.

Details about dependencies

Lines 124 – 136: The large number of utilities in *AI4Water* increases the number of underlying libraries. The *Model* class is built on top of the *Scikit-learn*, *CatBoost*, *XGBoost*, and *LightGBM* libraries to build classical machine learning models. These models have been used in several hydrological studies (Huang et al., 2019; Ni et al., 2020; Shahhosseini et al., 2021). To build deep learning models using neural networks, *AI4Water* uses popular deep learning platforms, such as *TensorFlow* (Abadi et al., 2016) and *Pytorch*. A complete list of the dependencies for *AI4Water* is presented in Table 1. It is divided into two parts. The first half shows the minimal requirements for running the basic utilities, which include building and training the model and making predictions from it. The second part of Table 1 comprises an exhaustive list of dependencies required to utilize all the functionalities of *AI4Water*. However, these utilities are optional and do not hamper the basic package functionality. Moreover, the modular structure of *AI4Water* allows the user to install libraries corresponding to a particular sub-module while ignoring the others, which are not required. For example, to use the *HyperOpt* class for hyperparameter optimization, libraries related to postprocessing are not required. Table 1 also presents the exact version of the underlying libraries, which were used to test the 1.0 version of *AI4Water*. *AI4Water* handles the version conflicts of the underlying libraries, thereby making it version-independent. This implies that the user can use any version greater than the version number provided in Table 1.

underlying methods and protocols for data transfer

Lines 228 – 233: The *DataHandler* class prepares the input data for the machine learning model and acts as an intermediate between the *Model* class and other preprocessing classes, such as *Imputation* and *Transformation* classes. The *DataHandler* can read data from various files as long as the data are in a tabular format in those files. The complete list of allowed file types and their

accepted file extensions is provided in Table S5. Internally, the *DataHandler* class stores data as a *pandas DataFrame* object, which is a data model of pandas for tabular data (Mckinney, 2011). *DataHandler* can also save processed data as an HDF5 file, which can be used to inspect processed input data.

A visual architecture diagram of all classes and third-party libraries will be helpful.

Response: We have added a figure depicting all the modules along with their available classes and third-party libraries (Fig. 3). *AI4Water* comprises several sub-modules, such as *eda*, *preprocessing*, *postprocessing*, *datasets*, *et*, Experiments, and *hyperopt*. Two types of third-party libraries are required by *AI4Water*. The first type of libraries are global, which are used in all the modules. These include numpy (Harris et al., 2020), matplotlib (Hunter, 2007), pandas (Mckinney, 2011), h5py (Collette, 2013) and plotly (Sievert, 2020). The second type of libraries are module-specific. Because these modules perform different tasks, their third-party dependencies are different from each other. For example, the "*hyper*_opt" module, which performs hyperparameter optimization, is reliant on hyperopt (Bergstra et al., 2015), scikit-optimize (Head et al., 2018), and optuna (Akiba et al., 2019) libraries. Similarly, the experiment module, which is used to compare different machine learning models, depends on tpot (Olson and Moore, 2016) and auto-keras (Jin et al., 2019) libraries. We have added a section named "sub-modules and code structure," which comprehensively discusses the sub-modules present in *AI4Water* and their interactions with each other.

Lines 109 – 123:

Sub-modules and code-structure

The code architecture of *AI4Water*, that is, its sub-submodules, their available classes, and third-party libraries are illustrated in Fig. 3. *AI4Water* comprises 11 sub-modules, among which 10 are task-based, and one is a general-purpose module named "*utils*." These sub-modules can be divided into two categories. The sub-modules on the left-hand side of Fig. 3 are designed for model building, hyperparameter optimization, and model comparison, whereas those on the right-hand side perform pre-processing and post-processing. Each sub-module exposes one or more classes to the user. For example, the *hyper_opt* sub-module presents the *Real*, *Categroical*, *Integer*, and *HyperOpt* classes. The third-party libraries required for each sub-module were annotated inside them. There are five "generic" third-party libraries that are required in all sub-modules (lower part of Fig. 3). The *et* and *utils* sub-modules do not require specific third-party libraries and depend only on generic libraries. The arrows in Fig. 3 indicate interaction between the sub-modules. The origin of the arrow denotes the caller sub-module, whereas their end points denote the sub-module that is being called. The *Model* class interacts with the *preprocessing* and *postprocessing* sub-modules using its functions, the names of which are shown in green in Fig. 3. For example, the *DataHandler* class in the *preprocessing* sub-module was responsible for data preparation. The *Model* class interacts with *DataHandler* using *training*_data, *validation_data*, and *test_data* methods, which are responsible for fetching training, validation, and test data from the *DataHandler* class, respectively.

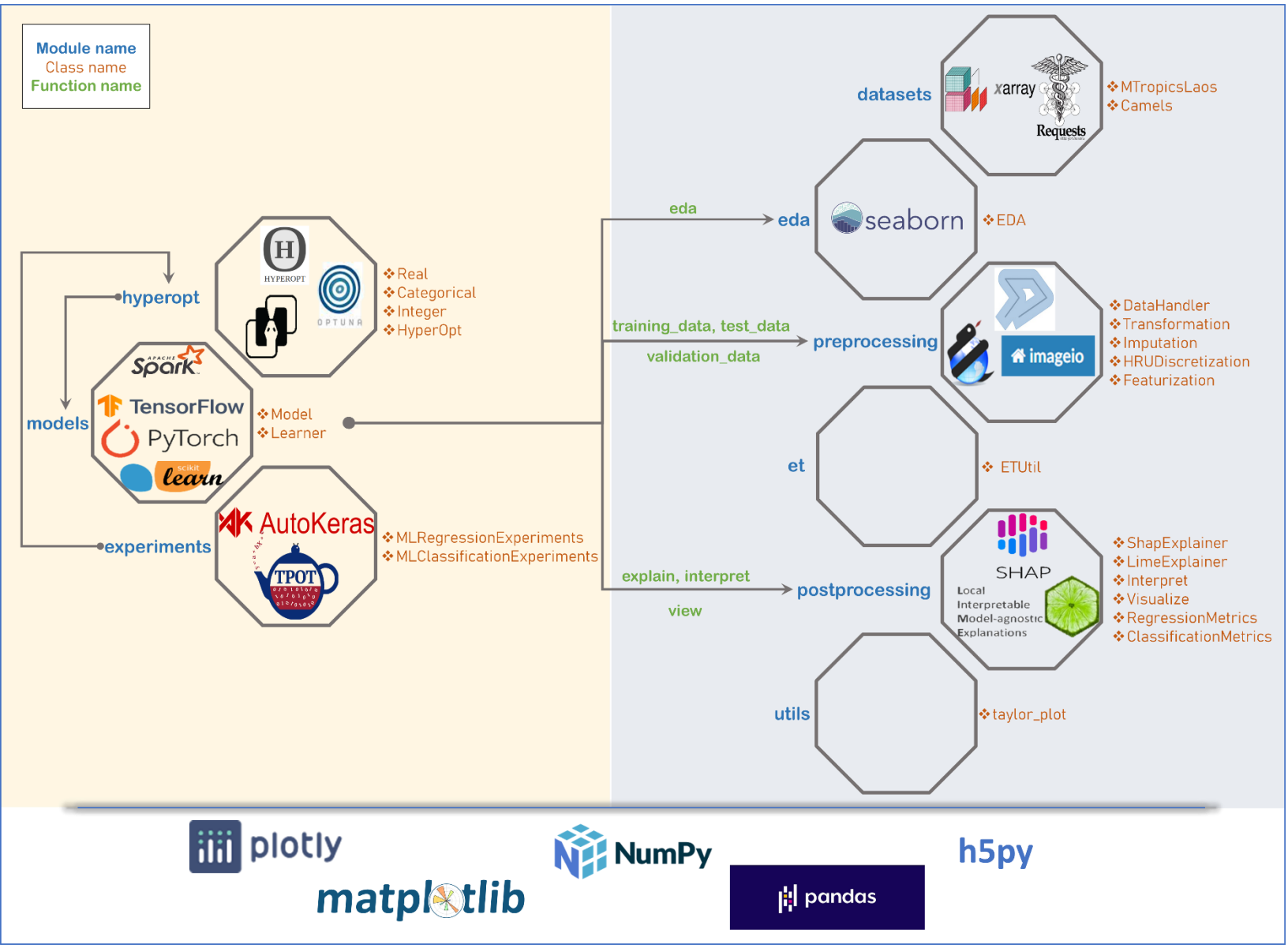

Figure 3: Framework architecture, sub-modules, classes and third-party libraries used by *AI4Water*. Each box represents a sub-module. The names of classes in each sub-module are written along with the corresponding box. The third-party libraries upon which the sub-module depends, are written inside the box. Empty boxes show that these sub-modules do not depend on a specific third-party library. The five generic libraries written at the bottom are used in all sub-modules.

Arrows represent the caller sub-module and the sub-module being called. The sub-modules on right hand side are related to pre-processing of data and post-processing of results. The *Model* class interacts with *preprocessing* and *postprocessing* sub-modules using its methods which are written in green color.

Technical details of data integration and API is not provided in detail.

Response: We have added technical details about the implementation of the *Model* class and its interaction with other sub-modules of *AI4Water*.

Lines 83 – 94: The *Model* class of *AI4Water* has two implementations and can have three backends. The two implementations are "model-subclassing" and "functional." The backends are either tensorflow, pytorch, or none. The backends, together with the implementations, determine the attributes that the *Model* class will inherit upon its creation. In model-subclassing implementation, the *Model* class inherits either from the tensorflow's *Model* class or the nn.module of pytorch. This implementation allows all the attributes from the corresponding backend to be also available from *AI4Water*'s *Model* class. For example, the "*count_params*" attribute of tensorflow's *Model* class can also be obtained from the *AI4Water*'s *Model* class. In functional implementation, the *Model* class of *AI4Water* does not inherit from the parent modules of tensorflow/pytorch. In this case, the built tensorflow/pytorch model object is exposed to the user as a "*_model*" attribute of the *Model* class. This is similar to tensorflow and pytorch libraries, both of which also have model-subclass and functional implementations. For models other than tensorflow or pytorch, the *Model* class does not have any backend. In these cases, the machine learning models are built using libraries such as

scikit-learn, xgboost, catboost, or lightgbm. The built model object is exposed to the user as "*_model*" attribute of the *Model* class.

A discussion about the interaction of *DataHandler* and Model class is also added in the revised manuscript.

Lines 227 – 233: The *DataHandler* class prepares the input data for the machine learning model and acts as an intermediate between the *Model* class and other preprocessing classes such as *Imputation* and *Transformation* classes. The *DataHandler* can read data from various files as long as the data are in a tabular format in those files. The complete list of allowed file types and their accepted file extensions is provided in Table S5. Internally, the *DataHandler* class stores data as *a pandas DataFrame* object, which is a data model of pandas for tabular data (Mckinney, 2011). *DataHandler* can also save processed data as an HDF5 file, which can be used to inspect processed input data.

Table S5. File types and their extensions accepted by *AI4Water*.

File extension	File type
.csv	Comma separated file
.xlsx	Microsoft Excel
.npz	Numpy zipped file
.parquet	Parquet
.feather	Feather
.nc	netCDF5
.mat	MATLAB

Which machine learning framework and version is used in the framework? Does system allow changing or updating the underlying ML library? They are briefly mentioned at the end but they are the most critical components of the framework.

Response: The inter-dependence of Python libraries is a complex issue and is difficult to resolve for a novice user. For example, Tensorflow 1.x is compatible with certain versions of a numpy library, whereas Tensorflow 2.x depends on some other version of the numpy. The same is true for other third-party libraries. Therefore, we have specified the exact versions with which the framework was tested. We have also modified the corresponding paragraph to add more details about this.

Lines 124 – 136: The large number of utilities in *AI4Water* increases the number of underlying libraries. The *Model* class is built on top of the *Scikit-learn*, *CatBoost*, *XGBoost*, and *LightGBM* libraries to build classical machine learning models. These models have been used in several hydrological studies (Huang et al., 2019; Ni et al., 2020; Shahhosseini et al., 2021). To build deep learning models using neural networks, *AI4Water* uses popular deep learning platforms, such as *TensorFlow* (Abadi et al., 2016) and *Pytorch*. A complete list of the dependencies for *AI4Water* is presented in Table 1. It is divided into two parts. The first half shows the minimal requirements for running the basic utilities, which include building and training the model and making predictions from it. The second part of Table 1 comprises an exhaustive list of dependencies required to utilize all the functionalities of *AI4Water*. However, these utilities are optional and do not hamper the basic package functionality. Moreover, the modular structure of *AI4Water* allows the user to install libraries corresponding to a particular sub-module while ignoring the others, which are not required. For example, to use the *HyperOpt* class for hyperparameter optimization,

libraries related to post-processing are not required. Table 1 also presents the exact version of the underlying libraries, which were used to test the 1.0 version of *AI4Water*. *AI4Water* handles the version conflicts of the underlying libraries, thereby making it version-independent. This implies that the user can use any version greater than the version number provided in Table 1.

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	1.19.2	array processing
pandas	1.2.4	array processing
matplotlib	3.4.2	visualization
h5py	2.10	storage
plotly	5.0	extended visualization
tensorflow	1.15, 2.1	building layers of neural networks
scikit-learn	0.24.2	building classical machine learning models
xgboost	1.4.2	implementing XGBoost based algorithms
catboost	0.26	implementing CatBoost based algorithms
lightgbm	3.2.1	implementing Light Gradient Boost based algorithms
Pyspark	3.1.2	Building classical machine learning models
tpot	0.11.7	Optimizing machine learning pipeline
imageio	2.9.0	spatial processing of shape files
shapely	1.7.1	spatial processing of shape files
pyshp	0.45	spatial processing of shape files
Scikit-optimize	0.8.1	Hyperparameter optimization using Bayesian
Optuna	2.8.0	Hyperparameter optimization
hyperopt	0.2.5	Hyperparameter optimization
shap	0.39.0	Model-agnostic interpretation

lime	0.2.0.1	Model interpretation
seaborn	0.11.1	visualization

How does the framework keep up with updates in third-party libraries and dependencies used in the framework?

Response: In *AI4Water*, we tried to remove the conflicts caused by the changes in the versions of third-party libraries. For example, significant changes were made in tensorflow code from version 1.x to 2.x. However, the user interface for building neural networks in *AI4Water* remained the same for both versions. This is because of the declarative model definition allowed in *AI4Water*. However, *AI4Water* cannot resolve issues that result from the changes in the requirements of third-party libraries. For example, the scikit-optimize library, which implements a Bayesian optimization algorithm, depends on a specific version of the scikit-learn library. This inter-dependency of the third-party libraries is difficult to predict. Similarly, different versions of tensorflow are dependent on different versions of numpy, which can be a major challenge for the user; therefore, we have mentioned the exact versions of all third-party libraries with which this framework has been tested. As this is an open source framework, we expect that future conflicts arising from the dependencies of third-party libraries can also be resolved. We have added the following lines in Chapter 7: Limitations and scope for expansion of the manuscript to highlight this challenge.

Lines 403 - 408: Another limitation of *AI4Water* is its dependence on a large number of third-party libraries. This can be challenging during installation when the interdependencies of libraries

conflict each other. Although we have provided the exact versions of the third-party libraries, which were used to test the current version of *AI4Water*, a conflict in future due to the changes in third-party libraries cannot be guaranteed. As *AI4Water* is an open-source project, we consider that such conflicts can be minimized with community inputs.

Does the library allow adding new data transformation, resampling, imputation or other functions?

Response: We used the object-oriented programming (OOP) paradigm to build this library. This paradigm allows the customization of any functionality of the *Model* class. The *Model* class interacts with the *DataHandler* class using *training_data*, *validation_data*, and *test_data* methods. Thus, if the users want to implement a custom transformation on the training data, they can subclassify the *Model* class and overwrite the *training_data* function. We have added code examples for implementing custom transformation, customizing the training loop of neural networks, and customizing the loss function. These examples are available as ipython notebooks in the example folder of the code repository. We have also added details regarding this in the manuscript.

Lines 384 - 386: For example, if users want to implement another transformation on the training data, they can subclass the *Model* class and overwrite the *"training_data"* method. Similarly, the user can customize the loss function by overwriting the "*loss*" method of *Model* class.

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., and Isard, M.: Tensorflow: A system for large-scale machine learning, 12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16), 265-283,
Akiba, T., Sano, S., Yanase, T., Ohta, T., and Koyama, M.: Optuna: A next-generation hyperparameter optimization framework, Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining, 2623-2631,
Bergstra, J., Komer, B., Eliasmith, C., Yamins, D., and Cox, D. D.: Hyperopt: a python library for model selection and hyperparameter optimization, Computational Science & Discovery, 8, 014008, 2015.
Collette, A.: Python and HDF5: unlocking scientific data, " O'Reilly Media, Inc."2013.
Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., and Smith, N. J.: Array programming with NumPy, Nature, 585, 357-362, https://doi.org/10.1038/s41586-020-2649-2, 2020.
Head, T., MechCoder, G. L., and Shcherbatyi, I.: scikit-optimize/scikit-optimize: v0. 5.2, Zenodo, 2018.
Huang, Y., Bárdossy, A., and Zhang, K.: Sensitivity of hydrological models to temporal and spatial resolutions of rainfall data, Hydrology and Earth System Sciences, 23, 2647-2663, 2019.
Hunter, J. D.: Matplotlib: A 2D graphics environment, Computing in science & engineering, 9, 90-95, 2007.
Jin, H., Song, Q., and Hu, X.: Auto-keras: An efficient neural architecture search system, Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 1946-1956,
McKinney, W.: pandas: a foundational Python library for data analysis and statistics, Python for high performance and scientific computing, 14, 1-9, 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, Journal of Hydrology, 586, 124901, 2020.
Olson, R. S. and Moore, J. H.: TPOT: A tree-based pipeline optimization tool for automating machine learning, Workshop on automatic machine learning, 66-74,
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, Scientific reports, 11, 1-15, 2021.
Sievert, C.: Interactive web-based data visualization with R, plotly, and shiny, CRC Press2020.