Advancing Reproducibility in Environmental Modeling: Integration of Open Repositories, Process Containerizations, and Seamless Workflows

A Dissertation
Presented to
The Faculty of the School of Engineering and Applied Sciences
University of Virginia

In Partial Fulfillment
of the requirements for the Degree of
Doctor of Philosophy (Civil and Environmental Engineering)

By
YoungDon Choi
July 27, 2021

Abstract

There is growing acknowledgment and awareness of the reproducibility challenge facing computational environmental modeling. To overcome this challenge, data sharing using open, online repositories that meet the FAIR (Findable, Accessible, Interoperable, and Reusable) guiding principles is recognized as a minimum standard to reproduce computational research. Even with these data sharing guidelines and well-documented workflows, it remains challenging to reproduce computational models due to complexities like inconsistent computational environments or difficulties in dealing with large datasets that prevent seamless, end-to-end modeling. Containerization technologies have been put forward as a means for addressing these problems by encapsulating computational environments, yet domain science researchers are often unclear about which containerization approach and technology is best for achieving a given modeling objective. Thus, to meet FAIR principles, researchers need clear guidelines for encapsulating seamless modeling workflows, especially for environmental modeling use cases that require large datasets. Toward these aims, this dissertation presents three studies to address current limitations of reproducibility in environmental modeling. The first study presents a framework for integrating three key components to improve reproducibility within modern computational environmental modeling: 1) online repositories for data and model sharing, 2) computational environments along with containerization technology and Jupyter notebooks for capturing reproducible modeling workflows, and 3) Application Programming Interfaces (APIs) for intuitive programmatic control of simulation models. The second study focuses on approaches for containerizing computational processes and suggests best practices and guidance for which approach is most appropriate to achieve specific modeling objectives when simulating environmental systems. The third study focuses on open and reproducible seamless environmental modeling workflows, especially when creating and sharing interoperable and reusable large-extent spatial datasets as model input. Key research contributions across these three studies are as follows. 1) Integration of online repositories for data and model sharing, computational environments along with containerization technology for capturing software dependencies, and workflows using model APIs and notebooks for model simulations creates a powerful system more open and reproducible environmental modeling. 2) Considering the needs and purposes of research and educational projects, and applying the appropriate containerization approach for each use case, makes computational research more reliable and efficient. 3) Sharing interoperable and reusable large-extent spatial datasets through open data repositories for model input supports seamless environmental modeling where data and processes can be reused across multiple applications. Finally, the methods developed and insights gained in this dissertation not only advance reliable and efficient computational reproducibility in environmental modeling, but also serve as best practices and guidance for achieving reproducibility in engineering practice and other scientific fields that rely on computational modeling.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Nearly all computational modeling fields are facing a reproducibility crisis (Monya; Baker, 2016; Hutton et al., 2016; McNutt, 2014; National Academies of Sciences, 2019; Stagge et al., 2019). According to a survey of 1,500 researchers, about 70% had tried but failed to reproduce published research and 90% agreed that the problem of reproducibility is a critical problem for scientific advancement (Monya; Baker, 2016). Within the hydrology and water resources fields, Stagge et al. (2019) analyzed 360 articles in six leading journals to understand if their data were available online and if the study results were reproducible. Their analysis showed that only 5.6% of the articles had data and model code available online along with directions for use, and only 1.1% were fully reproducible while 0.6% were partially reproducible. There are many possible reasons for this outcome; however, in this dissertation I argue that there is a need to extend the requirements to reproduce computational research from data sharing and well-documented workflows, the common practice now, to encapsulating all artifacts used in the original computational environments (Hut et al., 2017; Hutton et al., 2016). Reviewing recent research toward this goal of improving computational reproducibility, three distinct thrusts emerge: 1) open sharing of data and models online, 2) containerizing computational environments for core software and other secondary software, and 3) encapsulating computational workflows using Application Programming Interfaces (APIs) for programmatically control of complex computational research.

First, for the open sharing of data and models online, the FAIR principles have been presented as high-level guidelines to improve scientific data by making them Findable, Accessible, Interoperable, and Reusable (Wilkinson et al., 2016). The FAIR principles emphasize the necessities of both human and machine applicable data management environments and ongoing efforts on FAIR guiding principles have advanced data repositories with unique identifier mechanisms, data management plans, policies, and standards (Collins et al., 2018). Based on the use of unique identifiers such as the Digital Object Identifier (DOI), data can become “Findable.” Public machine-accessible APIs allow datasets and metadata to become “Accessible,” and the use of standard terms, metadata, and a wide range of data types allow data to become “Interoperable.” Finally, detailed documents together with metadata make data “Reusable.” Recently, numerous online repositories have accepted FAIR principles and enhanced their functionalities to be more Finable, Accessible, Interoperable, and Reusable. However, reproducibility research has led to a growing demand not only for data sharing with well-documented data, source code, software, and workflows, but also with tools for automatically encapsulating computational environments and workflows using containerization and literate programming (Kery et al., 2018; Knuth, 1984). For example, Bast (2019) suggested source code management and containerization tools are needed to reproduce computational environments for FAIRer principles, while Goble et al. (2020) suggested the FAIR principles need computational workflows to describe the execution of a computational workflow such as data collection, data preparation, data analysis, and modeling simulation. In hydrology, Hutton et al. (2016) recommended an online repository to easily find data and source code with unique persistent identifiers and computational workflows to describe the precise

procedure among data and modeling processes. In addition, Hut et al. (2017) suggested the use of containerization tools and open interfaces to complement the preservation of computational environments suggested in Hutton et al. (2016).

Second, containerization technologies have been developed and advanced for capturing computational environments for core model software along with other secondary, supporting software. Traditional approaches such as compiling software from source code to reproduce computational environments are difficult because they require a certain level of expertise about software dependencies, compilers, and computer environments to install and configure a complex computational modeling setup. While most model developers may know the specific requirements to reproduce their own model on another computer, for many modern scientific models it is challenging to completely document this procedure so that others can effectively and consistently reproduce it. To address this challenge, model developers have recently started using containerization tools such as Docker (Merkel, 2014), Singularity (Kurtzer et al., 2017), and Sciunit (That et al., 2017). Currently, Docker is the most popular containerization tool to encapsulate computational environments. Singularity is another containerization tool that is more popular for use in high performance computing (HPC) environments due to security concerns with Docker in these HPC environments. Sciunit is another containerization tool under active development that is more tailored for geoscience researchers with the goal of lowering the barrier to containerizing for this community. These containerization tools can be used to encapsulate the entire computational end-to-end environmental modeling workflow, allowing developers and researchers to more easily and confidently create reproducible modeling studies that can be repeated across machines. While these containerization tools offer an important opportunity, the challenge remains in deciding how to best utilize the tools for different modeling use cases and computational environments.

Third, APIs are growing in popularity for interacting with various complex environmental models. In environmental modeling, many studies mentioned that capturing the entire end-to-end workflows is important to achieve reproducibility and replicability. Current approaches to encapsulate workflows have focused on model execution and visualization. However, to complete "end-to-end" workflows, data preprocessing is critical for improving reproducibility, as the steps to create model input files are often nontrivial, requires a significant time investment (L. N. Leonard, 2015; Miles & Band, 2015). Therefore, the ability to improve preprocessing in "end-to-end" workflows is an important step for achieving reproducibility and replicability. Model APIs offer a means to programmatically interact with models in creating these end-to-end workflows. The recent popularity of literate programming tools offer a way to capture a modeling workflow as a narrative that intermingles code, making using of model APIs, text-based documentation, and inline visualization of model output directly within the same narrative (Kery et al., 2018; Knuth, 1984; Pimentel et al., 2019). For example, Jupyter (Avila et al., 2020; Pérez & Granger, 2007) and RMarkdown (Baumer et al., 2014; Rstudio Team, 2020) are used to incorporate code, data, description, and visualization needed to reproduce a computational experiment. Jupyter notebooks are growing quickly in use and popularity in computational fields as a means to document modeling workflows (Kluyver et al., 2016). When combined with model APIs, literate

programming tools like Jupyter notebooks offer a powerful means for creating end-to-end reproducible workflows.

In this dissertation, I aim to present a general framework to integrate the three components, online data repositories, computational environments, and modeling workflows through Application Programming Interfaces (APIs), for FAIRer environmental modeling. Online repositories are continuously maturing through FAIR principles to meet the requirements for reproducibility. Therefore, the second two studies in this dissertation focus on advancing other components of the framework, namely computational environments (the second component) and seamless environmental modeling workflows (the third component). This dissertation is, therefore, organized around three objectives, each addressing one of these specific research gaps, and each objective the focus of a separate chapter in the dissertation (Figure 1.1). These objectives are (1) to explore the integration of three key components within modern environment modeling, (2) to determine best practices for using tools and approaches available for containerizing environmental modeling software and executing it in different computing environments, and (3) to create and share interoperable and reusable large-extent spatial sample datasets as model input for seamless environmental modeling.

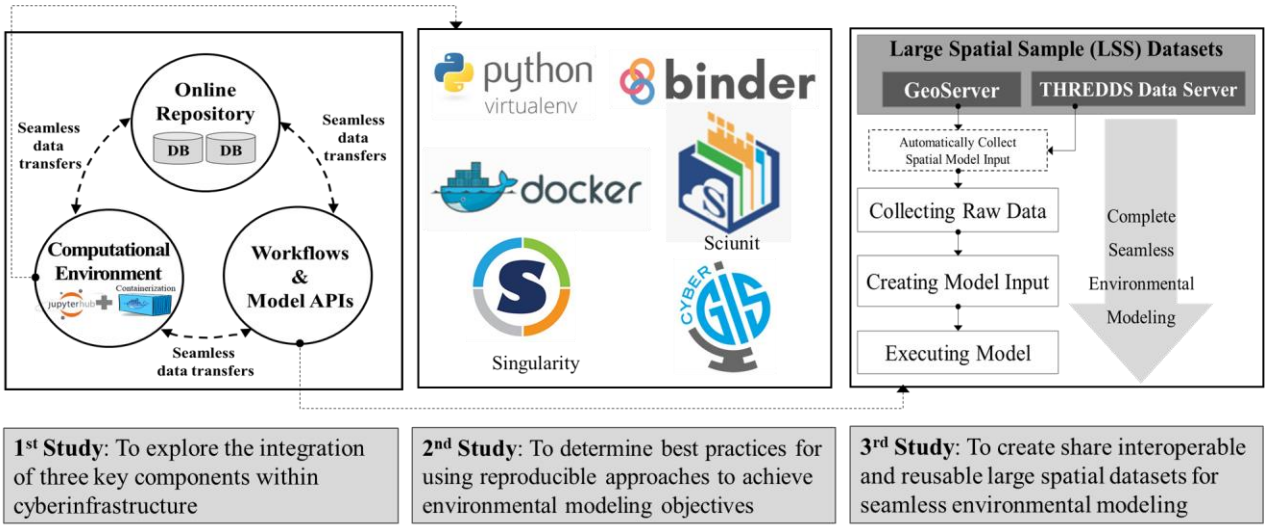

Figure 1.1. Overview of the integration of three key components and the advancement of each component through three targeted studies.

The first study (Chapter 2) presents the high-level concept and general design for integrating 1) online repositories for data and model sharing, 2) computational environments along with containerization technology and notebooks for capturing reproducible computational studies, and 3) APIs for simulation models to foster intuitive programmatic control. An example implementation of this framework is provided using HydroShare as the online repositories, CUAHSI JupyterHub and CyberGIS-Jupyter for water as computational environments, and pySUMMA as an example Python-based model API. The example implementation is applied for a SUMMA hydrologic modeling use case to demonstrate how the general approach can advance reproducible environmental modeling.

The second study (Chapter 3) focuses on determining best practices for containerization of computational environments. Many containerization approaches currently exist, however the challenge is deciding how best to utilize containerization tools for different modeling use cases and computational environments. This study presents best practices for reproducibility by comparing 11 approaches to achieving environmental modeling objectives. Five approaches were explored for using different combinations of software containerization methods on a local computing environment. Six additional approaches were considered that leveraged remote computing environments such as clusters and cloud computing architectures along with different containerization methods (Li, 2020; Prasad et al., 2020; Shuler & Mariner, 2020). Remote resources utilized were the CUAHSI JupyterHub, CyberGIS-Jupyter for water, MyBinder, and University of Virginia HPC environment. The results of both quantitative and qualitative performance tests across the 11 approaches are presented using a hydrologic modeling use case example. The findings of this study are described in terms of best practices for using reproducible workflows for different environmental modeling objectives.

The third study (Chapter 4) focuses on improving seamless environmental modeling, especially in terms of accessing and processing large-extent data inputs for environmental models. Seamless environmental modeling seeks to integrate model processes alongside end-to-end modeling workflows and large, seamless datasets to enable "models of everywhere" (Blair et al., 2019; L. N. Leonard, 2015; Miles & Band, 2015; Slater et al., 2019) and provide consistent data across model applications (Mizukami et al., 2017; Samaniego et al., 2017). This study creates and shares interoperable and reusable large-extent spatial datasets on GeoServer and THREDDS data server (TDS) to support open and reproducible seamless environmental modeling workflows. As an example application, three state-scale (North Carolina, Maryland, and Virginia) spatial datasets were created and used in RHESSys seamless modeling workflows on the CyberGIS-Jupyter for water platform. Using three watershed models as case studies (Coweeta subbasin18 in NC, Scotts Level Branch in MD, and Spout Run in VA), the results of data consistency, both in terms of model input and output data, are presented to demonstrate the feasibility of the approach for distributing large-extent, spatial data for building environmental models.

Following this introduction are chapters for each of the three studies followed by an overall conclusions section. Each chapter is written as a standalone study to enable publication in peer-reviewed journals.

A version of this chapter was published in the January 2021 issue of *Environmental Modelling & Software*. The publication can be found at https://doi.org/10.1016/j.envsoft.2020.104888.

Chapter 2

Toward Open and Reproducible Environmental Modeling by Integrating Online Data Repositories, Computational Environment, and Model Application Programming Interfaces

2.1 Introduction

There is a growing acknowledgment and awareness of the reproducibility challenge facing computational environmental modeling fields (Hutton et al., 2016; Stagge et al., 2019) as well as in other computational modeling disciplines (Monya; Baker, 2016; McNutt, 2014; National Academies of Sciences, 2019). According to a survey of 1,576 researchers, about 70% had tried but failed to reproduce published research and 90% agreed that the problem of reproducibility is a critical problem for scientific advancement (Monya; Baker, 2016). Within the hydrology and water resources fields, Stagge et al. (2019) analyzed 360 articles in six leading journals to understand if their data were available online and if the study results were reproducible. Their analysis showed that only 5.6% of the articles had data and model code available online along with directions for use, and only 1.1% were fully reproducible while 0.6% were partially reproducible. There are many possible reasons for this outcome; however, we argue along with others that advances in the cyberinfrastructure that enable modern computational science is critical to achieving reproducible research (Hut et al., 2017; Hutton et al., 2016).

Reviewing recent research toward this goal of improving the underlying cyberinfrastructure necessary to support reproducible computational studies, we see three distinct thrusts: 1) open sharing of data and models online, 2) encapsulating computational environments through containers and self-documented computational notebooks, and 3) creating Application Programming Interfaces (APIs) for programmatically control of complex computational models. A major effort to improve the open sharing of data and models is the FAIR (Findable, Accessible, Interoperable, Reusable) guiding principles for scientific data management and stewardship (Wilkinson et al., 2016). However, FAIR principles speak primarily to openness, which is essential but insufficient on its own for addressing reproducibility of computational software and computational environments (Bast, 2019). Ince et al. (2012) argued that, even with well-developed data and software sharing capabilities, it remains challenging to reproduce published results due to difficulties in documenting computational environments needed to repeat past studies. Moreover, they found this especially true for operating system environments and software dependencies that can cause unpredictable differences with even slight changes in model source code or configuration.

To address this need, a second thrust in recent research is aimed at overcoming the difficulties with sharing complete computational software environments. Research that has focused on improving the sharing of well documented data and software workflows for computational studies includes Stodden and Miguez (2013), for example, who proposed sharing data, algorithms, and workflows to utilize and verify published results. Similarly, Gil et al. (2016) suggested the best practices of sharing data, software, and documents in an open and transparent way using a high-level roadmap of approaches to strengthen reproducibility in the geosciences. In the meantime, the broader information technology community has introduced the concept of containers as a means for encapsulating computational environments (Kurtzer et al., 2017; Merkel, 2014). The result of this work has benefited computational modeling fields and led to efforts to improve the preservation of operating system and software dependencies, strengthening reproducibility in computational research (Boettiger, 2015; Brinckman et al., 2019). Containerization technologies such as Docker (Merkel, 2014) have been used to reproduce computational modeling environments without requiring users to install additional dependencies (Boettiger, 2015; Signell & Pothina, 2019). Software tools like Sciunit (Essawy et al., 2018; Yuan et al., 2018) ease the process of containerizing, sharing, and tracking scientific applications, lowering the barrier to entry for researchers to use containerization tools.

Containerization has also led to the ability to create new modeling environments and deploy them through interactive, online analysis environments such as JupyterHub (Kluyver et al., 2016). Jupyter notebooks are quickly growing in use and popularity in computational fields as a means to document studies as a mix of formatted text, mathematical equations, and executable code with in-line visualizations resulting from the code (Kluyver et al., 2016). JupyterHub is a cloud-based software that utilizes containerization to support the execution of multiple Jupyter notebooks simultaneously. Recent advances leveraging Jupyter for environmental modeling include work by Castronova et al. (2018) who created the CUAHSI JupyterHub to support online hydrologic modeling and analysis, Yin et al. (2017) who created a TauDEM (Tarboton, 1997) modeling environment with JupyterHub, Eynard-Bontemps et al. (2019) who created the PANGEO project that supports big data studies in the geosciences and heavily leverages JupyterHub, and Bandaragoda et al. (2019) who used JupyterHub within a larger knowledge infrastructure to support earth system modeling. Recent work has also begun to explore combining external computational environments including high performance computing (HPC) and high throughput computing (HTC) cyberinfrastructure for model execution directly through Jupyter notebooks (Lyu et al., 2019). That work also takes advantage of containerization concepts to easily port preconfigured model execution environments to available computational resources.

The third thrust we observe in recent research is efforts to create APIs for computational environmental models. While many models have Graphical User Interfaces (GUIs) for improving the usability of the models, APIs are different in that they facilitate programmatically interacting with a simulation model to configure input files, execute models, and analyze model outputs. Python (https://www.python.org) and R (https://rstudio.com) are common programming languages used for creating model APIs. Python has examples including model APIs for the Stormwater Management Model (PySWMM, B. E. McDonnell, 2017), MODFLOW (FloPy, Bakker et al., 2016), Hydrologic Simulation Program in Fortran (PyHSPF, Lampert & Wu, 2015),

and Precipitation Runoff Modeling System (PRMS-Python, Volk & Turner, 2019). R has examples including model APIs for TOPMODEL (topmodel, Buytaert, 2011), SWAT (SWATmodel, Fuka et al., 2014), and TUW model (TUWmodel, Viglione & Parajka, 2020). These model APIs help by abstracting low-level programmatic details of input file manipulation and model execution operations from end users. In this way, they are particularly useful when combined with computational notebooks for creating self-documented modeling studies that can be more easily understood and reproduced by both modelers and non-modelers alike.

While work along each of these thrusts – online data repositories, computational environments leveraging containerization and computational notebooks, and model APIs – is important individually, integrating these three thrusts offers a powerful approach for reproducible computational modeling. Recent research has started to explore this integration includes (1) the GI-RHESSys (Green Infrastructure-Regional Hydro-Ecological Simulation System) Jupyter environment created for Green Infrastructure (GI) landscape designs and modeling output using JupyterHub (Leonard et al., 2019), (2) the Landlab model (Hobley et al., 2017) with recent work to implement Landlab within JupyterHub as a knowledge infrastructure (Bandaragoda et al., 2019), and (3) the HydroTerre system (L. Leonard & Duffy, 2016) that links an online data repository with the Penn State Integrated Hydrologic Model (PIHM). While these examples focused on supporting individual modeling use cases, they reveal general patterns of infrastructure components necessary to implement their systems. Our aim is to build on this past work by first presenting this general pattern as a general approach that can be followed for building new modeling systems. Second, we provide an example implementation of the general approach that can be easily expanded to support any computational environmental model that is containerized and has an accompanying model API.

The objective of this research is, therefore, to put forward a general approach or framework for integrating online data repositories, computational environments, and model APIs to enable more open and reproducible environmental modeling. In the Methodology section, we first present a high-level design of the approach describing each of the three components in more detail while also discussing different options available for online repositories, notebook-based and containerized modeling environments, and model APIs. We then present an example implementation that makes use of HydroShare as an online repository, CUAHSI JupyterHub and CyberGIS-Jupyter for water as computational environments, and pySUMMA as an example model API. In the Results and Discussion section, we present the results of applying the example implementation to reproduce a prior hydrologic modeling study (Clark et al., 2015b) and discuss the difficulty and nuance in claiming to achieve reproducibility. We also present limitations of the work that could be a focus of future research. Finally, we conclude by summarizing the findings and emphasizing their contribution to the larger goal of making past and future studies simpler to reproduce through advances in cyberinfrastructure.

2.2 Methodology

In this section, we describe the general approach being put forward for open and reproducible environmental modeling (Section 2.1) and then present an example implementation of this general approach for hydrologic modeling (Section 2.2).

2.2.1 Overview of General Approach and Description of System Components

The general modeling system approach considered in this research consists of three primary components (Figure 2.1). Component 1 is the online repository where data, models, and notebooks can be openly shared with the community. Component 2 is the JupyterHub computational environment where containerized models can be executed using notebooks. Component 3 consists of a collection of model APIs, one for each model supported within the system, that allow for programmatic configuration, execution, and visualization through computational notebooks. The three components are integrated through seamless data transfers to create a powerful framework for open and reproducible modeling analyses. In practice, we anticipate that this general approach or framework may have many different physical implementations, where different technologies may serve the needs of specific subcommunities within the broader environmental modeling field. We demonstrate one such implementation in Section 2.2 for the hydrology community. In the following subsections, we describe each of these components in more detail while also providing examples of each that are available for integration.

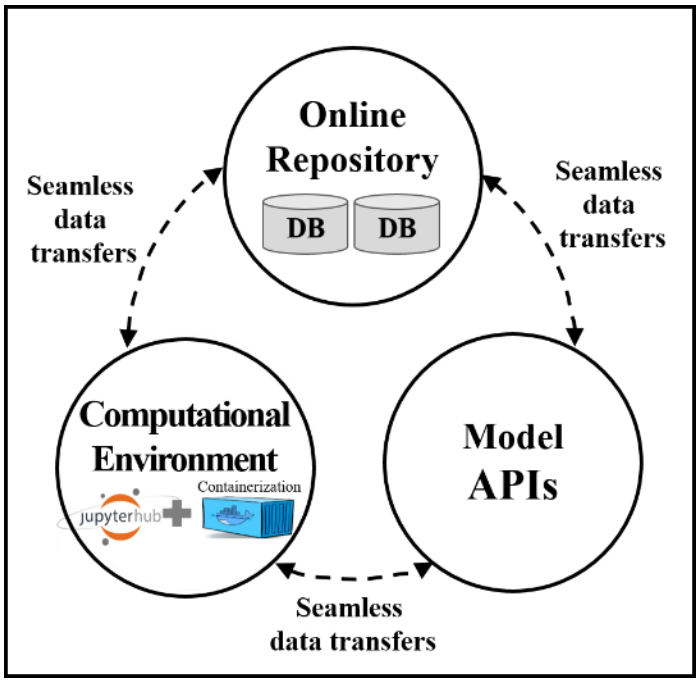

Figure 2.1. A general modeling approach consisting of three primary components with seamless data transfers for open and reproducible environmental modeling

2.2.1.1 Online Repository

Online repositories allow for storing, sharing, and publishing data, metadata, and other resources required to reproduce computational research findings. These online repositories often support a rich set of user-friendly features such as metadata capture, persistent digital object identifiers (DOIs), and extensive APIs for programmatically creating, updating, and deleting resources. They also often support various data types such as documents, figures, code, audio, and video with metadata tailored to each data type. Some examples of online repositories used by researchers include DataOne member nodes (https://www.dataone.org), FigShare (https://figshare.com), Harvard Dataverse (https://dataverse.harvard.edu), and HydroShare (https://www.hydroshare.org).

Many online repositories serve broad scientific communities and, therefore, maintain only general and widely applicable capabilities. Others are more targeted to specific communities and, as a result, can offer more specific functionality. Environmental modeling, for example, is not a common use case for many repositories that focus on more general data sharing needs (e.g. FigShare). Environmental models, however, have their own characteristics that consist of software, input and output files, and data processing workflows. Morsy et al. (2017) described these unique needs of models being stored in data repositories and presented a data model design including metadata descriptions for key modeling objects to support flexible and applicable model sharing framework. This design is implemented within the HydroShare data repository, allowing for describing and sharing more specific model resource types.

2.2.1.2 Computational Environment

A computational environment serves as a gateway for model configuration, execution, and post-processing. In the case of model execution, environmental modeling often includes complex simulation models along with data pre- and post-processing software, all with software dependencies that range from the operating system, to modules used within a model engine, to libraries used by data processing and analysis software (e.g., Python libraries). Without the ability to replicate a computational environment, slight inconsistencies in software dependencies can result in well-documented model studies failing when ported to a new machine. Without the use of recent innovations like containers, documenting the exact computational environment used in an analysis is difficult, time consuming, and error prone. To overcome these challenges, Docker (Merkel, 2014) and Singularity (Kurtzer et al., 2017) have emerged as containerization techniques used to encapsulate a computational modeling environment, as described further in the implementation (see Section 2.2).

Along with containers, computational gateway interfaces are also critical to lowering the barrier to entry and supporting more open and reproducible modeling in online computational environments. With the emergence of JupyterHub as a gateway innovation, there has been an increased interest in cloud-based modeling environments for creating, editing, and running computational notebooks. Markham (2019) reviewed five popular cloud services that support computational notebooks (Table 2.1). We reviewed two additional cloud services, 1) CUAHSI JupyterHub (hereafter CUAHSI JH) and 2) CyberGIS-Jupyter for water (hereafter CyberGIS JW),

and included them in Table 2.1 as well. The environments range from scientific services (e.g., the CUAHSI JH and CyberGIS JW that are used in this work) to more general services such as Binder (Jupyter Project et al., 2018). Large technology companies including Google and Microsoft have provided notebook execution environments such as Google Colab and Microsoft Azure Notebooks, demonstrating the popularity and growing interest in a variety of fields. Many cloud services have adopted the default Jupyter interface available from the Jupyter project without modification, while others have modified this interface to customize it for their own purposes (Markham, 2019). Furthermore, many cloud services support Python, R and other languages as well. Interface similarity in Table 2.1 considers available menus, buttons, and other visual elements that make up the user interface, and how different they are from a default Jupyter interface. All services listed in Table 2.1 are candidates for integration into an implementation of the modeling system described in this paper.

Table 2.1. Comparison of interface similarity and supported languages of cloud services for executing computational notebooks (expanded from Markham, 2019)

Cloud Services	CUAHSI JH	CyberGIS JW	Binder	Kaggle Kernels	Google Collaboratory	Microsoft Azure Notebooks (free plan)	CoCalc (free plan)
Interface similarity to Jupyter	100%	100%	100%	70%	60%	100%	95%
Supported Languages	Python 3 R	Python 3 R	Python 3 R, Julia, Many others	Python 3 R	Python 3 Swift	Python 3 R, F#	Python 3 R, Julia, Many others

2.2.1.3 Model APIs

An API defines a set of protocols or tools to communicate with an operating system, database, network, and other lower-level aspects of a software system (Reddy, 2011). The abstraction provided by an API has benefits (Brooks, 2013) including 1) flexibility and efficiency for data access, 2) personalization to customize the functionality that users access the most, and 3) reusability of code to work more productively. Examples of widely used APIs include the Google Maps API for map services and the Twitter API for social networking services. Services also widely exist for scientific systems relevant to environmental modeling including the HydroShare REST (Representational State Transfer) API for sharing and publishing water data as well as APIs for a growing number of environmental models.

In this study, we focused on Python-based model APIs and reviewed a series of model APIs including PRMS-Python (Volk & Turner, 2019) and PyHSPF (Lampert & Wu, 2015) to better understand how they are designed and structured. Doing this can help to inform the design and structure of future APIs created to support specific environmental models. We observed that model API functionalities fell into three categories: model input, model execution, and model output (Table 2.2). For PRMS-Python, as an example, input files often have corresponding Python modules that can be used for data manipulation. For PyHSPF, as an example, the Python modules do not have a one-to-one correspondence with the core model files and modules. Instead, the API

designs include a higher-level abstraction to consider core classes needed for interacting with the model.

Table 2.2. General categories for a model API mapped to examples from PRMS-Python and PyHSPF

General Categories	API Objective	PRMS	PRMS-Python	HSPF	PyHSPF
(a) Model Input	- Generating and manipulating model input	-control file -data file -parameters file	-prms_config.txt -data.py -parameters.py	-control file -watershed data -management file	- wdmutil.py - watershed.py - hspfmodel.py :
(b) Model Execution	- Executing and refining models	-shell script	-simulation.py -scenario.py -optimization.py	-shell script	- forecaster.py - extract.py - calibratormodel.py :
(c) Model Output	- Visualizing and analyzing model output	-text file	-optimizer.py -utils.py	-text file	- gisplots.py - forecastplots.py - autocalibrator.py :

From this review, we suggest that communities of modelers (e.g., researchers or groups of researchers) who are considering building a model API for a specific environmental model begin with answering the following questions. 1) What configuration and input files should be exposed through the API to allow for programmatic changes and what are the logical classes for organizing these model input configuration attributes? 2) What methods and attributes should the API expose for executing the model and refining the model through, for example, calibration or sensitivity analysis? 3) What are common visualizations of the model output that many users would wish to produce? Creating a model API with this functionality in a well thought through design will serve as a solid foundation for future extensions to the software. Furthermore, the extent to which environmental model APIs can adopt conventions for the organization of their design and structure will allow users to more easily learn new model APIs by having some consistency across model APIs.

2.2.2 Example Implementation

In this section, we present one possible physical implementation of approach described in the prior section. This example implementation uses HydroShare as the online repository, CUAHSI JH and CyberGIS JW the computational environments, and pySUMMA as one of potentially many model APIs within the system. While this example implementation targets the needs of the hydrologic modeling community, we anticipate that multiple other permutations of the technologies described in the prior section could be assembled to meet the needs of other environmental modeling communities.

2.2.2.1 HydroShare as the Online Repository

We used HydroShare as the data repository in our example implementation due to both its flexibility and tailored functionalities for supporting environmental modeling use cases.

HydroShare is an online repository tailored for the needs of the hydrologic community, but general enough to satisfy other environmental modeling needs (Tarboton et al., 2014). HydroShare defines a "Resource" as "the fundamental unit of digital content in HydroShare that contains data and/or model files and their corresponding metadata" (Horsburgh et al., 2016). HydroShare resources support various content types such as geographic raster (GeoTIFF), multidimensional arrays (NetCDF), geographic features (Shapefile), and time series. HydroShare also defines a composite resource type that supports combining data of different content types into a single HydroShare resource, as well as a collection resource type that supports aggregation of related HydroShare resources into a list that can be referenced with a single unique identifier. Furthermore, realizing that data associated with models have their own characteristics, HydroShare defines unique model resource types of a model program (the software) and a model instance (the input and output files for a specific model run) (Morsy et al., 2017). Resources with these two resource types are related through the "*ExecutedBy*" attribute of a model instance, which points to the specific model program resource used to execute that model instance. This design allows for a one-to-many link between a model program that is used to execute many different model instances built for different geographic locations or to address different research questions.

The methodology for sharing computational modeling resources is shown in Figure 2.2. First, the user creates a model program resource for each version of a model program software used in the analysis. This resource can include the source code, executable, and container for the model program itself, or a link to one or more of these resources shared in a system external to HydroShare (e.g., in GitHub, BinderHub or DockerHub). Second, the model instance resources are created to store and describe the input data required to execute the model and can optionally store the output after the model is executed. Then the model instance is linked to a specific model program resource using the "*ExecutedBy*" metadata term. A separate composite resource is used to store Jupyter notebooks that describe the overall analysis workflow. Finally, a collection resource is used to combine and conveniently share all of the resources used to complete the study.

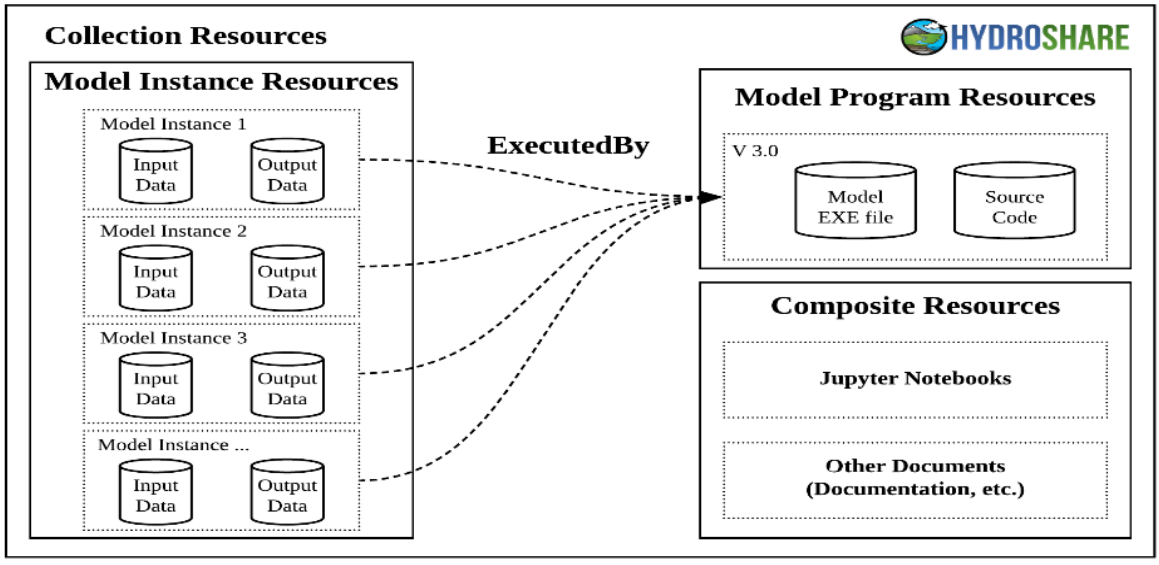

Figure 2.2. A methodology for sharing resources used for a modeling analysis through HydroShare

2.2.2.2 JupyterHub as the Computational Environment

We integrated both the CUAHSI JH and CyberGIS JW as computational environments in our example implementation. We chose these environments because both are publicly available and aimed at scientific modeling in the water and environmental communities. Moreover, both systems allow for seamless data transfer with HydroShare as a data repository supporting the necessary interoperability between these two components of the general framework. This data transfer is enabled through the HydroShare REST API and the standardization of HydroShare resource data structures.

The CUAHSI JH is a cloud computing environment on the Google Cloud Platform specifically designed to support research and education in the water sciences (Figure 2.3). To support a variety of applications, it leverages environment profiles that allow users to choose the ideal computing configuration for their work. Each of these profiles is a separate containerized environment that has been built with a specific set of software to support various water science use cases. Currently, the CUAHSI JH consists of seven profiles that range from scientific Python and R to HydroLearn (https://www.hydrolearn.org), educational modules and hydrologic modeling. In addition, the CUAHSI JH supports persistent data, meaning user-created content is stored between sessions and shared between profile environments. Moreover, this environment enables users to install custom software using conda virtual environments. For this study, we created a "Python 3.7 SUMMA Modeling" profile to support SUMMA 3.0 modeling environment using a Dockerfile in CUAHSI JH.

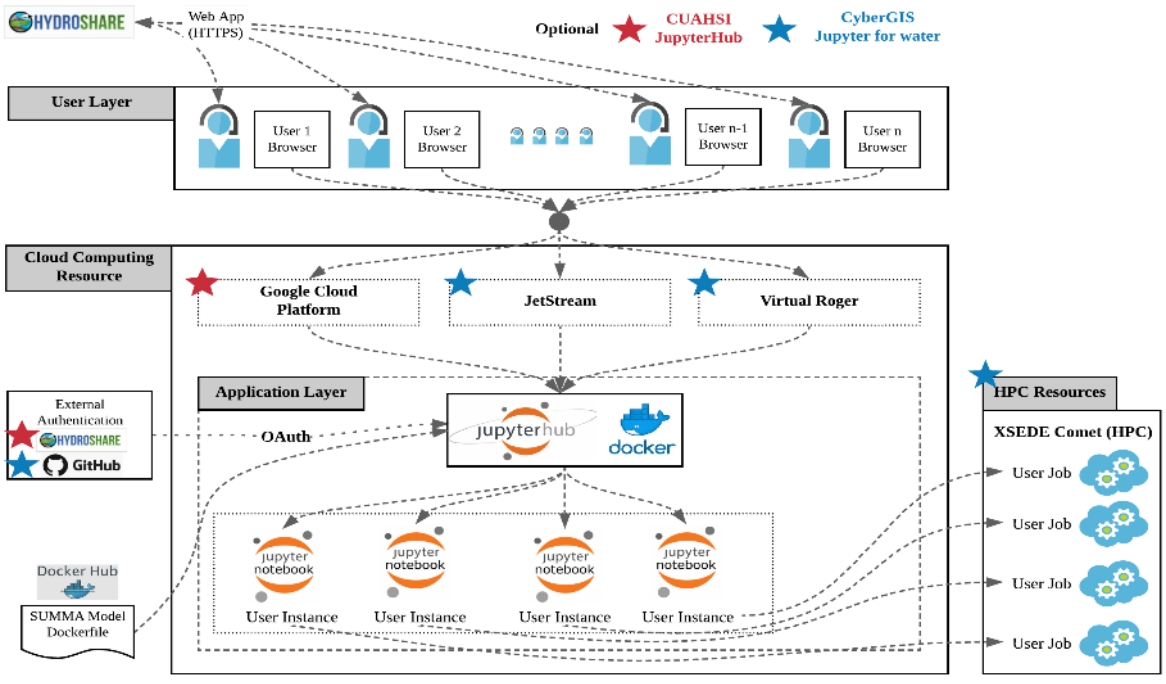

Figure 2.3. The CUAHSI JH and CyberGIS JW environments with model execution environments configured as Docker images to support concurrent model execution through Jupyter notebooks.

Another model execution environment interoperable with HydroShare, CyberGIS JW, is a well-tailored CyberGISX (https://cybergisxhub.cigi.illinois.edu) instance to serve the fast-emerging needs for data-intensive and reproducible research in the environmental modeling community (Figure 2.4). Overall, CyberGIS JW is similar to CUAHSI JH, but CyberGIS JW also includes interoperability with advanced cyberinfrastructure resources such as Virtual ROGER (a cyberGIS supercomputer hosted by the CyberGIS Center for Advanced Digital and Spatial Studies at the University of Illinois) and XSEDE Comet (an HPC resource on the Extreme Science and Engineering Discovery Environment) for model execution support. Lyu et al. (2019) describe how to use HTC through a Jupyter notebook using SUMMA as an example case in CyberGIS-Jupyter (beta), which is the previous version of CyberGIS JW. Currently CyberGIS JW is supporting LandLab (Hobley et al., 2017) and RHESSys (Tague et al., 2004) modeling environments. For this study, we created a SUMMA modeling environment using a Dockerfile. Users can use this SUMMA modeling environment via a SUMMA kernel. For use of HPC resources, CyberGIS JW requires a Singularity image to support a computational modeling environment in XSEDE because CyberGIS JW and XSEDE are separately placed. Also, CyberGIS JW needs a particular library to connect to computational resources for submitting jobs and data exchange in XSEDE.

2.2.2.3 pySUMMA as the Model API

The model API pySUMMA was created through this research as an example model API. pySUMMA wraps the hydrologic model Structure for Unifying Multiple Modeling Alternative (SUMMA) (Clark et al., 2015a). SUMMA was selected for this study because it is a general hydrologic modeling environment offering the ability to conduct model experiments with controlled and systematic evaluation of multiple model representations of hydrologic processes and scaling behavior. The SUMMA model simulates both the thermodynamics, the storage and flux of energy such as the heat balance of the vegetation canopy, snow, and soil affected by the radiative fluxes, as well as the hydrology, the storage and transmission of water (for example, vertical and lateral transmission of water through vegetation canopy, snow, soil, aquifer and river within a catchment system). The flexible hierarchical spatial structure of SUMMA supports different spatial configurations including the size and shape of model elements with Grouped Response Units (GRUs) (Kouwen et al., 1993) and Hydrologic Response Units (HRUs). In addition, the flexible structure enables researchers to consider the lateral flux of water across the model domain and complex topographical properties like hillslopes and riparian areas. This flexibility within SUMMA enables hydrologists to find solutions for the application of scaling behavior in relation to different physical processes.

SUMMA also enables hydrologists to select the appropriate physical process methods and model complexity. This process implements a modular structure that is supported by the conservation equations to calculate each process in a controlled and systematic way. This unified process helps users to concentrate important physical parameterizations with higher complexity and, conversely, to simplify specific processes to minimize uncertainty according to the purpose and characteristics of biophysics and hydrology. Moreover, the structure of SUMMA, which

consists of a core (solver) and outer branches, enables the output of a numerical solution from SUMMA so that the user can evaluate the accuracy and efficiency of the model. Therefore, SUMMA supports flexibility to simulate different options of physical processes and numerical solutions.

We designed and implemented pySUMMA as a model API for SUMMA using the questions proposed in Section 2.1.3 for guiding the design of a new model API (Table 2.3). For the model input category, there are six configuration files to manipulate SUMMA input: 1) *File Manager*, 2) *Decisions*, 3) *Forcing File List*, 4) *Model Output*, 5) *Param Trial*, and 6) *Local Attribute* files. To expose the first four configuration files through the API, we created *file_manger.py, decisions.py, force_file_list.py*, *output_control.py* and *option.py*. For the rest of the configuration files, we created *assign_trial_params* and *assign_attributes* methods in *Simulation.py*. In the model execution category, we created *Simulation.py* to use the model execution command conveniently from the shell script format so that users do not need to edit manually every time. We also created two options to execute the SUMMA model, 'local' and 'docker', to satisfy different user requirements. Finally, the output format of SUMMA is NetCDF; therefore, we created *plotting.py* for visualization considering the output variables and their output structure in NetCDF.

Table 2.3. Implementation of a model API for SUMMA

General Categories	Questions	SUMMA	pySUMMA
(a) Model Input	(1) What configuration and input files should be exposed through the API to allow for programmatic changes?	-file manager -decision file -forcing file list file -model output file -param trial file -local attribute file	-file_manager.py -decisions.py -force_file_list.py -output_control.py -option.py
(b) Model Execution	(2) What methods and commands should the API expose for executing the model?	-shell script -SUMMA compilation (summa.exe) or Docker	-simulation.py -SUMMA compilation (summa.exe) or Docker
(c) Model Output	(3) What are common visualizations of the model output that many users would wish to produce?	-output NetCDF	-plotting.py

The classes of pySUMMA are shown in Figure 2.4. A Simulation module (*Simulation.py*) is used as the initial Python module to start a pySUMMA API and combine most pySUMMA functionalities, such as manipulating configuration files and executing SUMMA. After creating a pySUMMA simulation object, users can manipulate six configuration files. A File manager module (*file_manager.py)* reads and manipulates a *File Manager* file which controls the location of every configuration file for the SUMMA model. For example, the *File Manager* file sets the directory and configuration files including the decision, forcing, parameter, and attribute files. A Decisions module (*decisions.py)* reads and manipulates a Decisions file which sets different physical process parameterizations. Through the *available_options* object in *decisions.py*, users can determine what options are available for model parameterizations and select model parameterizations from a list of options for each physical process (SUMMA Online Document,

2020). Four input configuration modules (*file_manager.py, decisions.py, force_file_list.py, and output_control.py*) have the same pattern of classes. For example, a File manager module (*file_manager.py)* is composed of *FileManagerOption* and *FileManager* classes and a Decisions module (*decisions.py)* is composed of *DecisionOption* and *Decision* classes. Each class is connected to an Option module (*option.py*) to avoid repetition of functions such as comparing, setting and writing each configuration file. After setting the SUMMA configuration, the simulation module (*Simulation.py)* is used for model execution. The *run()* method of the *Simulation* class is used to execute the SUMMA model. This execution can be done in both "local" and "docker" computational environments. The environments are set using the *run_option* parameter for the *run()* method as discussed later in the Results and Discussion section.

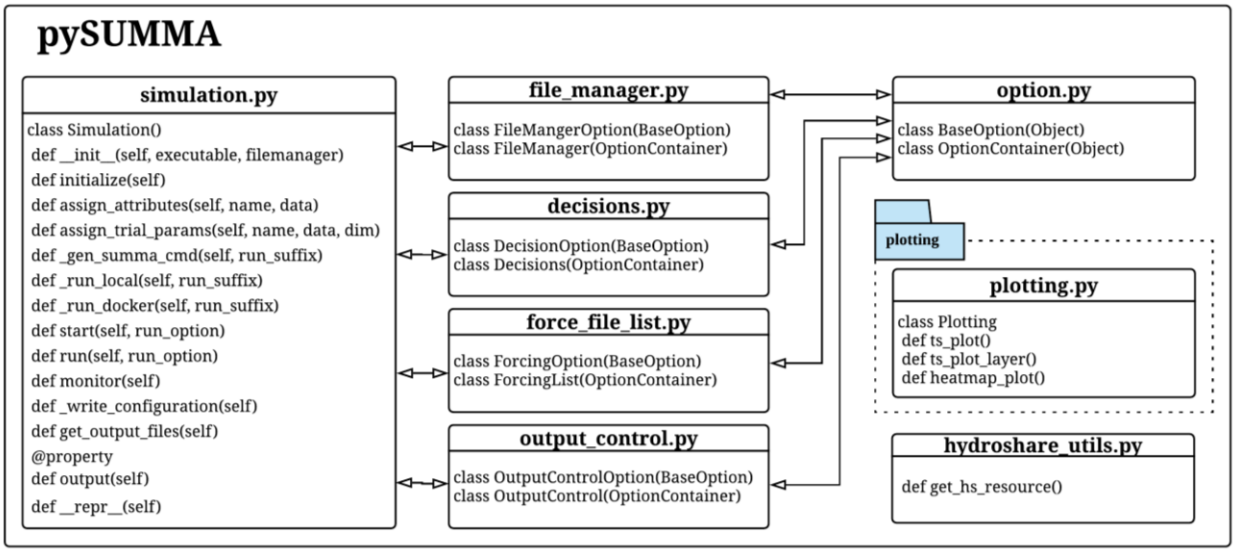

Figure 2.4. pySUMMA library classes

Once a SUMMA model run has been completed, the plotting module (*plotting.py)* can be used to visualize the results. There are two different data output structures for SUMMA: 1) time, HRU (or GRU) number, and variable; 2) time, HRU (or GRU) number, soil (or snow) layer number, and variable. To visualize each of these output structures, the *Plotting class* consists of three methods: *ts_plot(), ts_plot_layer(),* and *heatmap_plot().* We used the seaborn library (statistical data visualization library) to create a 2D heat map with soil or snow layer and time as the axis for displaying a selected variable. Lastly, the model output module (*output_control.py*) is used to manipulate the output variables of SUMMA and the utilities module (*hydroshare_utils.py*) has functions to download test cases of SUMMA (model instance resources) and execution files (model program resources) from HydroShare.

2.3 Results and Discussion

In this section, we present a modeling case study application of the example implementation system described in Section 2.2. Then, we discuss how this approach addresses the challenge of achieving more reproducible studies summarized in the Introduction section by evaluating the

approach against definitions, concepts, and metrics for reproducibility proposed by others. Lastly, we discuss the limitations of our approach that present opportunities for future research.

2.3.1 Case Study Description

Clark et al. (2015b) describe a set of thirteen modeling experiments exploring various hydrologic modeling scenarios using SUMMA. The study area for these modeling experiments is the Reynolds Mountain East Area (A=32.7km^2) in the Reynolds Creek Experimental Watershed in Idaho, USA (Figure 2.5). In this paper, we focus on these modeling experiments as a case study with the goal of applying our approach so that each Clark et al. (2015b) experiment can be reconstructed and shared openly in a way that is easier to reproduce.

The first step toward this goal is the creation and organization of HydroShare resources to share all models and data files required for the analysis. The second step is to create Jupyter notebooks that describe the modeling experiments. These notebooks include text and equations to describe the modeling experiments while also including executable Python code using the pySUMMA API and inline visualizations that can be repeated and extended by others. We created seven Jupyter notebooks, each one documenting an experiment in the Clark et al. (2015b) study, and published them through HydroShare as a collection resource (Choi et al., 2020).

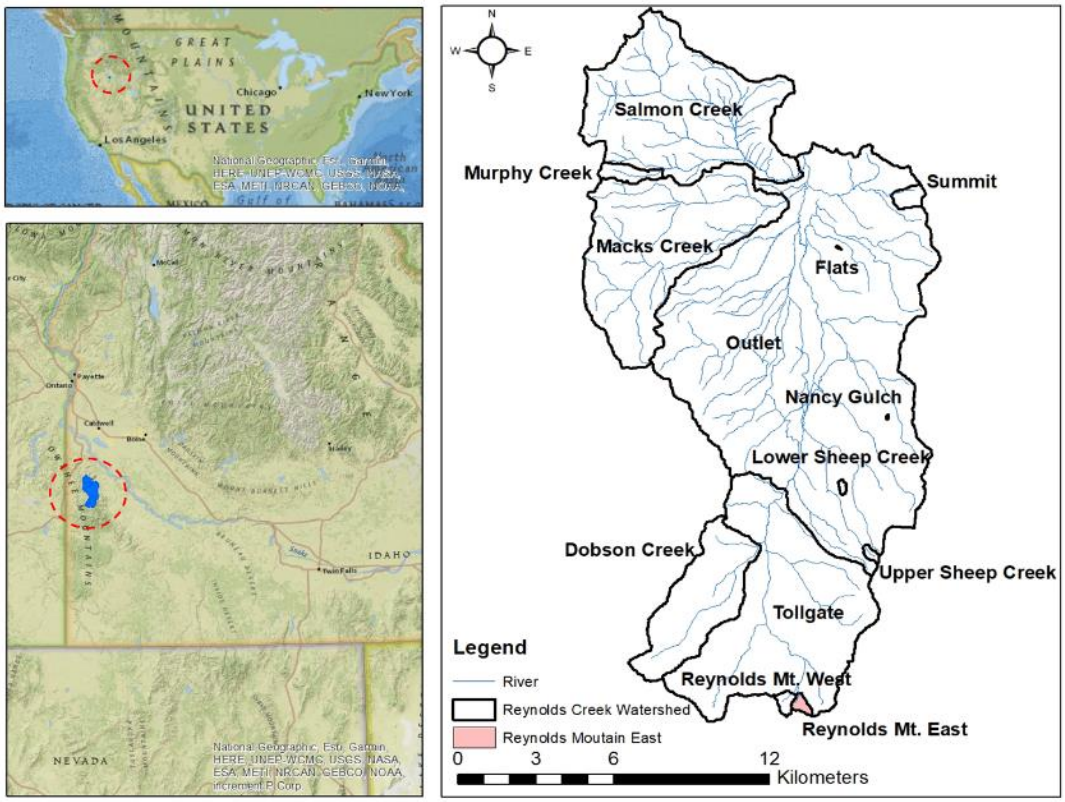

Figure 2.5. Reynolds Mountain East Area in the Reynolds Creek Experimental Watershed

2.3.2 Model and Data Resources

Our first step in reproducing the Clark et al. (2015b) modeling experiments was to publish the specific SUMMA model version used in the analysis as a resource on HydroShare. To do this, we created a HydroShare resource using the Model Program resource type and upload the SUMMA 3.0.0 source code to the resource. We then published the resource through HydroShare so that it is persistent and immutable with a unique Digital Object Identifier (DOI) (Choi et al., 2020). Figure 2.6 shows the landing page for this resource on HydroShare that includes detailed metadata describing 1) the source code and compiled software engine, 2) metadata for the software, 3) a link showing the model was derived from a particular branch of a GitHub repository for SUMMA, and 4) a citation for referencing the resource. This same SUMMA 3.0.0 was also installed on the CUAHSI JH allowing users to execute the SUMMA model directly from CUAHSI JH.

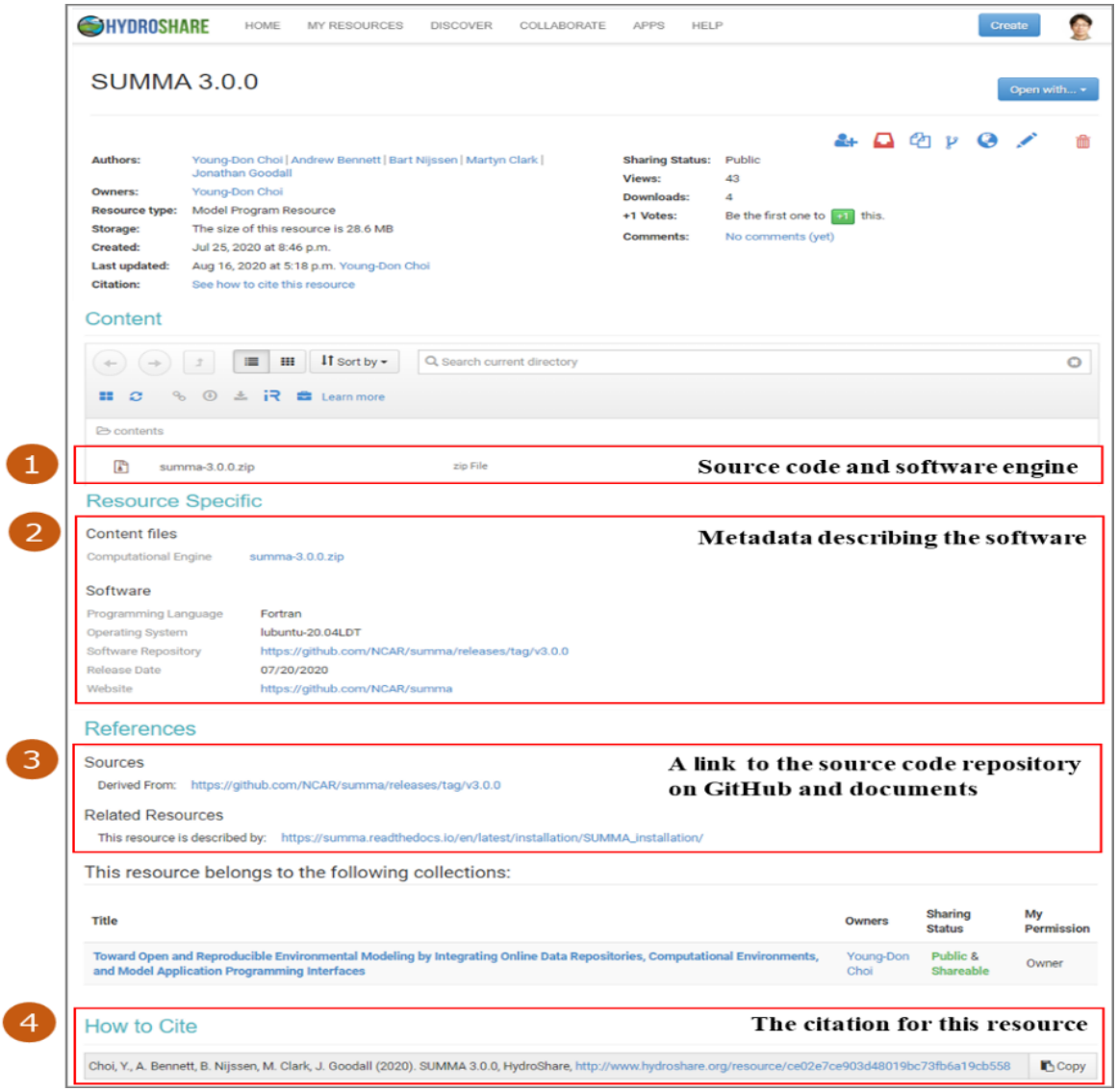

Figure 2.6. The HydroShare landing page for a SUMMA model program resource used in the example analysis (Choi et al., 2020).

We next created multiple resources in HydroShare to store the specific model inputs for each different SUMMA model experiment used in the Clark et al. (2015b) paper. There were four synthetic and nine field study test cases available as an online supplement to the Clark et al. (2015b) paper. From these data, we created seven unique model instance resources in HydroShare (Table 2.4) and grouped them into a collection resource (Choi et al., 2020). Each model instance resource includes 1) input data for the SUMMA model, 2) a reference to the Clark et al. (2015b) paper, 3) a composite resource link that points to the Jupyter notebook used to execute the SUMMA model, and 4) a link to the model program resource used to execute the model instance.

Table 2.4. Mapping between the modeling experiments of Clark et al. (2015b) and Model Instance Resources on HydroShare used to store the input files for that model experiment

Figures from Clark et al. (2015b)	Resource Name on HydroShare to Reproduce each Clark et al. (2015b) Figure
Figure 1 (top)	The impact of the canopy shortwave radiation parameterizations of SUMMA Model in Aspen stand at Reynolds Mountain East
Figure 1 (bottom)	The impact of LAI parameter on the below canopy shortwave radiation of SUMMA Model in Aspen stand at Reynolds Mountain East
Figure 2	The impact of the canopy wind parameter for the exponential wind profile of SUMMA Model in Aspen stand at Reynolds Mountain East
Figure 7	The impact of Stomatal Resistance Parameterization on ET of SUMMA Model in Aspen stand at Reynolds Mountain East
Figure 8 (left)	The impact of Root Distributions Parameters on ET of SUMMA Model in Aspen stand at Reynolds Mountain East
Figure 8 (right)	The impact of Lateral Flow Parameterizations on ET of SUMMA Model in Aspen stand at Reynolds Mountain East
Figure 9	The impact of Lateral Flow Parameterizations on Runoff of SUMMA Model in Aspen stand at Reynolds Mountain East

Once this step is complete, the model and data resources required to reproduce the Clark et al. (2015b) experiments are publicly accessible in HydroShare with metadata to describe each resource and a unique URL to locate each resource. HydroShare also allows for publishing these resources in which case the resources become immutable and are assigned a Digital Object Identifier (DOI). This pattern can be adopted by other environmental modeling studies whereby both the model and data resources required to reproduce the study are uploaded into HydroShare, given metadata to describe each resource (including relationships between resources such as the "*ExecutedBy*" relationship between model program and model instance resources), and published with a DOI.

2.3.3 Demonstrating Reproducibility

This section describes the steps that should be taken to reproduce one of the experiments described in Clark et al. (2015b). As a preparation step before starting a SUMMA simulation using Jupyter notebooks on CUAHSI JupyterHub, we recommend creating a pySUMMA conda virtual environment by running the steps described in the notebook "Creating_ pySUMMA _conda_virtual_environment_in_CUAHSI_JupyterHub.ipynb" in the HydroShare composite resource for CUAHSI JH notebooks. Once this preparation step is completed, the basic algorithm

to run a notebook is shown in Figure 2.7. First, the pySUMMA *hydroshare_utils* module is used to download the model instance that will be used in the notebook directly from HydroShare. After downloading the SUMMA model instance, it is possible to create a pySUMMA simulation object using the *Simulation* class of pySUMMA and supplying SUMMA executable (summa.exe) and the *File Manager* file path. After creating the pySUMMA simulation object, the SUMMA model can be executed using the *run()* method, which takes a *run_option* argument as *local*. When CUAHSI JH was created by using Docker, SUMMA was automatically complied and created SUMMA executable in '*/usr/local/bin/summa.exe*'. Therefore, after setting the *executable* variable to the location of "summa.exe", users can set a *run_option* as *local*. By changing the *executable* variable as *"/usr/bin/summa.exe*", it is possible to execute the same notebook on CyberGIS JW.

```
# Downloading the model instance required for the modeling scenario directly from HydroShare based on its unique Resource ID
from pysumma import hydroshare_utils
import os
resource_id = '13d6b84a9553410297a67fa366a56cb2'
instance = hydroshare_utils.get_hs_resource(resource_id, os.getcwd())

# Creating the pySUMMA simulation instance
import pysumma as ps
executable = '/usr/local/bin/summa.exe'
file_manager = os.path.join(os.getcwd(), instance, 'settings/summa_fileManager_riparianAspenSimpleResistance.txt')
S = ps.Simulation(executable, file_manager)

# Executing the SUMMA example model in the CUAHSI JupyterHub
S.run(run_option='local', run_suffix='_simpleResistance')
```

Figure 2.7. The basic step for a SUMMA model run using Jupyter notebooks

As an example, we present here the results from running two different experiments included in the Clark et al. (2015b) paper. The first reproduces Figure 7 from Clark et al. (2015b) and is published as a HydroShare resource with the title "The impact of Stomatal Resistance Parameterization on ET of SUMMA Model in Aspen stand at Reynolds Mountain East." The second reproduces Figure 9 (left) from Clark et al. (2015b) and is published as a HydroShare resource with the title "The impact of Root Distributions Parameters on ET of SUMMA Model in Aspen stand at Reynolds Mountain East."

Figure 2.8 gives the results from the first experiment that explores the impact of three different stomatal resistance parameterizations on total evapotranspiration: Ball-Berry (Ball et al., 1987), Jarvis (Jarvis, 1976), and the simple resistance method. Figure 2.8a is the original result from the SUMMA paper (Clark, Nijssen, Lundquist, Kavetski, Rupp, Woods, Freer, Gutmann, Wood, Gochis, et al., 2015) and Figure 2.8b is a reproduced figure resulting from applying this framework. These stomatal resistance parameterizations have different physical characteristics: both the Jarvis and Ball Berry methods consider photosynthesis, while the simple soil resistance method mainly considers the soil water conditions. The results show that the simple soil resistance method is higher than the other methods during the night hours. Comparing the two plots shows the complexity associated with reproducing past computational modeling studies. While the results are consistent, they are not exact. The precise reason for the differences in the model results is difficult to determine. We suspect it due in part to upgrades to SUMMA or SUMMA dependencies between the versions of the SUMMA 2.0 used in the Clark et al. (2015b) paper and the SUMMA 3.0 used to create the newer plot. More vexing is that some of the observed data points appear to

have shifted with no good explanation for why. One possible explanation could be the fact that different visualization tools were used to create each plot: Interactive Data Language (IDL) for the plot on the left and matplotlib for the plot on the right. We suspect differences like this would not be uncommon when trying to reproduce any past computational study given the difficulties in recreating the exact computational and analysis environment including data preparation routines, computational modeling software, and post-processing analysis and visualization tools. This, in fact, speaks to the difficulty of the problem and the need for innovation in cyberinfrastructure approaches that is at the heart of this study. This said, it is also important to stress that the goal of reproducibility may not be to obtain the *exact* same results, but rather *consistent* results that would produce in the same conclusion. This is an idea expressed by high level reports on computational reproducibility (National Academies of Sciences, 2019) that we will discuss further in Section 3.4.

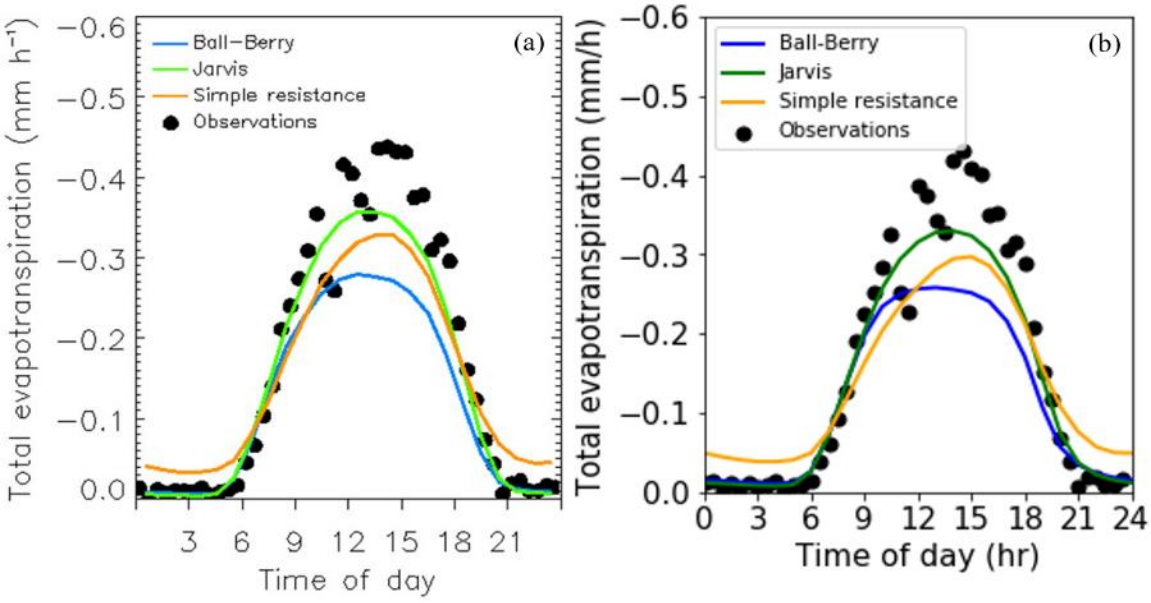

Figure 2.8. Reproducibility of Figure 7 from Clark et al. (2015b) showing the impact of the three different stomatal resistance parameterizations on total evapotranspiration (a): published result, (b): reproduced result

Figure 2.9 shows the results from the second experiment from Clark et al. (2015b), which explores the impact of the root distribution parameters with different stomatal resistance parameterizations for total evapotranspiration. In this case, we reproduced the plot that shows the impact of root distribution parameters (Figure 2.9b) and compared it to the previous result (Figure 2.9a). Again, we see consistent (although not exact) results between the two model runs. Given that the modeling experiment is now implemented within the system, it is also possible to more easily extend and repurpose it for other purposes. To this point, we demonstrate reuse of past modeling studies by creating two additional plots for determining the effect of different root distribution (Figure 2.9c) and stomatal resistance parameterizations (Figure 2.9d) on total evapotranspiration. These plots show how higher root distribution exponents in the soil profile indicate that the roots are deeper in the soil, which makes it easier for plants to extract soil water. As a result, during the higher evapotranspiration periods (10:00-17:00), the Jarvis method more closely matched the observation data. However, during the period when evapotranspiration is decreasing (17:00-20:00), the Ball-Berry method was more precise compared to the simple

resistance method. Over the complete time period, the analysis shows that the Jarvis method had the best fit with observations.

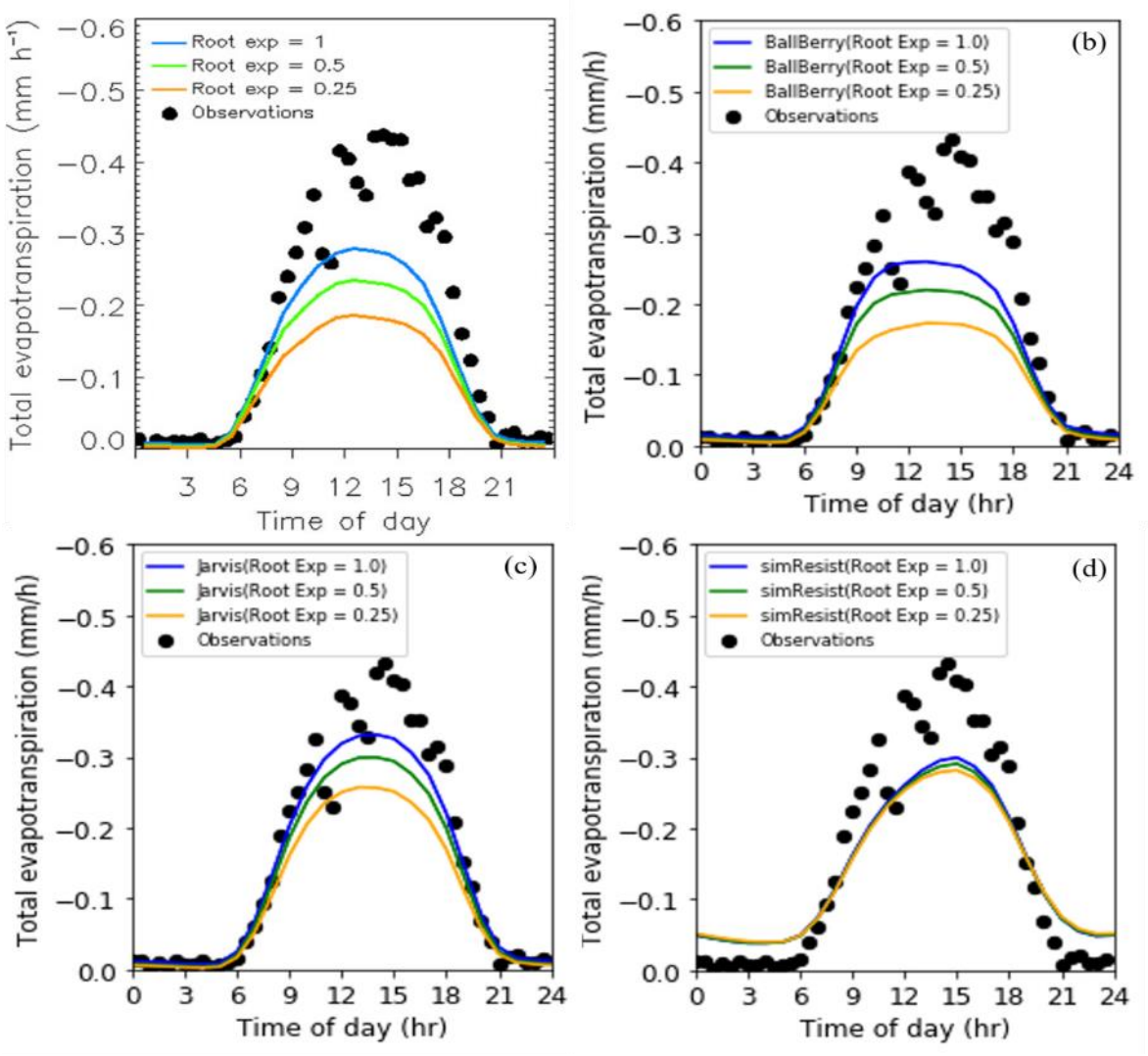

Figure 2.9. Reproducibility and reusability of Figure 8 (a) of Clark et al. (2015b) showing the impact of root distribution parameter with different stomatal resistance parameterization on total evapotranspiration (a): published output, (b): reproduced output, (c) and (d): output from reusability application extending the prior study

2.3.4 Evaluating Reproducibility

To evaluate if reproducibility was achieved, we considered definitions and concepts for evaluating reproducibility being put forward by others. For example, the National Academies of Science, Engineering, and Medicine (National Academies of Sciences, 2019) define reproducibility, focused on computational reproducibility, as "obtaining consistent results using the same input data; computational steps, methods, and code; and conditions of analysis." To guarantee reproducibility, the organization recommended delivering "clear, specific, and complete information about any computational methods and data products to repeat the previous study, and that information should include the data, methods, and computational environment." FAIR

principles (Wilkinson et al., 2016) include 15 metrics that should be met as a minimum for reproducibility. These metrics are a) Findable (4 metrics): (meta) data to find easy, b) Accessible (4 metrics): (meta) data to access with authentication and authorization; c) Interoperable (3 metrics): (meta) data to interoperate with applications; d) Reusable (4 metrics): (meta) data to achieve reusability, reproducibility, and replicability.

In the hydrology and water resources fields, Hutton et al. (2016) recommended reproducible studies have 1) readable and reusable code, 2) an unambiguous workflow, 3) a repository to easily find data and code with associated metadata, 4) use of unique persistent identifiers, 5) new procedures to reproduce large-extent studies using HPC. Additionally, Hut et al. (2017) suggested the use of containers and open interfaces to guarantee stronger reproducibility as a response to Hutton et al., (2016). Finally, Stagge et al. (2019) proposed a set of survey questions to assess the reproducibility of a journal article. The survey requires that eight elements be available for a study to be called reproducible: 1) directions to run or reproduce the study, 2) code/model/software files, 3) input data, 4) hardware/software requirements, 5) stated data persistence policy, 6) materials linked by unique and persistent identifiers, 7) metadata to describe the code, and 8) common file format and instructions to open these files.

With these criteria in mind, by simply using HydroShare as the data repository for all data and software used for the study, we can support many of the metrics associated with reproducibility. HydroShare supports FAIR principles (Tarboton et al., 2018) for each resource that includes model input, source code, metadata, and supplementary documents. Using JupyterHub as described in the paper provides a consistent computational environment and using Jupyter notebooks and containerized model execution environments provides a clear and easy workflow to assure users can reproduce a published study. Finally, using a model API makes it easier for a user to follow the logic and steps used to configure, run, and postprocess a modeling simulation, allowing for not only reproducibility but also reuse and extension of prior work. Therefore, if we compare these definitions and concepts for a validation of reproducibility to our approach and its example application, we can claim that it satisfies the criteria for reproducible computational modeling. Still, while the framework allows for satisfying the criteria, it is still up to the user to ensure care is taken with sharing and documenting resources with adequate metadata and instructions to achieve reproducibility.

2.3.5 Approach Limitations and Opportunities for Future Research

This research focuses on examples that assume model input files had already been processed and are available for use in the modeling analysis. The preprocessing steps required to generate model input files from raw geospatial and time series observational data are a necessary component of longer-term goals for creating so called “end-to-end” reproducible analysis workflows. For example, Slater et al. (2019) provided an “end-to-end” reproducible hydrology workflow using R for climate data retrieval, spatial analysis, modeling, statistical analysis, visualization, and data publishing. As another example of automated end-to-end workflows, HydroTerre (L. N. Leonard, 2015) includes 1) data workflows (L. Leonard & Duffy, 2013) to create watershed models using Essential Terrestrial Variables (ETV), 2) data-model workflows to

transform watershed data into model inputs, 3) model workflows (L. Leonard & Duffy, 2014) to execute models in HPC, especially The Penn State Integrated Hydrologic Modeling System (PIHM), and 4) visualization workflows to visualize the first three workflows to easily create and share model results for analysis.

Currently, pySUMMA has developed the functionalities of manipulating created model input, executing SUMMA, and plotting model output. To complete "end-to-end" workflows, data preprocessing is critical for improving reproducibility as the steps to create model input files are often nontrivial and require a significant time investment. Prior work to address this challenge includes the EcohydroLib Python library developed as a software framework for managing spatial data acquisition and preparation workflows for ecohydrology modeling (Miles & Band, 2015). EcohydroLib takes advantage of open source GRASS GIS libraries to automate data gathering and preparation for environmental models. It is a model agnostic approach for mapping a variety of data sources into input files required by environmental models. Alternative data processing workflows and pipelines such as HydroTerre could also be explored for bringing data preprocessing capabilities for environmental models into the general approach described through this work. However, just having new data processing pipelines alone will be insufficient. We also need more detailed modeling protocols and procedures to replicate (or even reproduce) a study (Ceola et al., 2015) because reproducibility is not just a technological problem, it is equally an educational problem (Grüning et al., 2018).

Post-processing for visualization and model analysis procedures is also essential to creating a powerful modeling environment, saving time when analyzing model output, and strengthening reproducibility. To grow use of model APIs, many analysis methods will be necessary such as plotting, calibration, optimization, and uncertainty analysis. While pySUMMA is still being developed toward these goals, other model APIs discussed in this paper and that could be used within the example modeling system do have more robust processing capabilities already. One question that remains is the extent to which environmental model APIs can reuse underlying software to support common model post-processing routines. General libraries in Python, such as Pandas and matplotlib, are universally applicable to environmental modeling post-processing tasks. However, is a plotting or data analysis library more tailored for environmental modeling but still sufficiently general to serve many environmental models possible? If so, it could further reduce the duplication of code across environmental model APIs and, ultimately, encourage more environmental model APIs that are robust, easier to maintain, and feature rich.

The ability to include data pre- and post-processing within the framework would be an important step for moving from reproducibility to replicability within the framework. Replicability is defined by the National Academies of Science, Engineering, and Medicine (National Academies of Sciences, 2019) as "obtaining consistent results across studies aimed at answering the same scientific question, each of which has obtained its own data." Replication, therefore, can be thought of as a next step beyond reproducibility where a study is repeated using new data, potentially from a new site or different time period, but similar methods. This work has focused on a general approach to support reproducibility of computational models. The framework could be extended for replication by extending a model API, like the pySUMMA API described in this paper, to

include not only functions for model configuration (e.g., settings and parameter values assuming model input files have already been generated), but also for model preprocessing where input files for the model are generated from raw data sources.

2.4 Conclusions

Computational irreproducibility is an important problem in many scientific fields. Recent research to improve computational reproducibility has focused on advancing the sharing of data used in studies, using computational notebooks and containers for encapsulating complete computational environments, and developing model APIs for programmatically interacting with simulation models. A contribution of this research is to present a general approach to integrate these three areas of past work into a general approach for supporting more open and reproducible environmental modeling. We present an example implementation of this approach by leveraging 1) HydroShare as a data sharing repository, 2) JupyterHub as a notebook-based, containerized, and cloud-based computational environment, and 3) pySUMMA as an example model API able to abstract lower-level details for model configuration, execution, and visualization from end users.

Using the example implementation, we demonstrate how modeling analyses can be completed in a more open and reproducible way. Building from a prior study presenting a series of modeling experiments applying SUMMA at the Reynolds Mountain East Area in the Reynolds Creek Experimental Watershed in Idaho, USA (Clark et al., 2015b), we first create and organize HydroShare resources to share data and model files. Next, we create Jupyter notebooks that leveraged the pySUMMA API, introduced in this paper, to reproduce and extend figures from the prior study. Each notebook (a) pulled required data from HydroShare into the computational environment, (b) provided a notebook using text, equations, code, and inline visualizations for documenting the experiment, and (c) allowed for online execution of the notebook and sharing of modifications to the notebook through HydroShare. Finally, we discuss how we evaluated that reproducibility was achieved and future steps that could be taken to further improve the proposed framework.

From this research, we conclude that cyberinfrastructure is reaching a point where it is possible to build open and transparent environmental modeling systems. Online repositories are sufficiently mature where they can be relied upon for storing key data and software resources for studies. Computational environments able to execute containerized environmental models can be interlinked with data repositories and the ability for these computational environments to serve as gateways to High Performance Computing (HPC) resources is improving. More models are being provided with APIs that allow for programmatic control of the model configuration, execution, and visualization. Jupyter notebooks provide an important orchestration and documentation glue across these components where users can leverage APIs to access and publish data from online repositories, submit jobs to HPC resources, and programmatically interact with state-of-the-art environmental models. Linking these capabilities in a way that can be built upon and expanded as new models become available, as demonstrated in this paper, will move environmental modeling in a direction where open, transparent, reproducible, reusable, and replicable studies become the rule rather than the exception.

Software and Data Availability

All software and data used in this study were published with persistent digital object identifiers (DOI's) on HydroShare. A collection resource in HydroShare (Choi et al., 2020) contains each of these resources. In addition to these resources published through HydroShare, the pySUMMA source code created though this study is available on GitHub as detailed below.

Product Title: pySUMMA v3.0.0

Lead Developers: Young-Don Choi and Andrew Bennett

Contact Email: yc5ef@virginia.edu, andrbenn@uw.edu

Tested Platform:

- HydroShare CUAHSI JupyterHub
- CyberGIS-Jupyter for water

Software Required: Python 3.5 or above

Availability: The pySUMMA source code is publicly available through GitHub

- https://github.com/UW-Hydro/pysumma/releases/tag/3.0.0

License: BSD 3-Clause License

List of Relevant URLs

CUAHSI JupyterHub: https://jupyterhub.cuahsi.org/

CUAHSI JupyterHub Legacy Environment: https://jupyter.cuahsi.org

CUAHSI JupyterHub GitHub: https://github.com/hydroshare/hydroshare-jupyterhub

CyberGIS-Jupyter (beta): https://hsjupyter.cigi.illinois.edu:8000

CyberGIS-Jupyter for Water: https://go.illinois.edu/cybergis-jupyter-water

DataOne: https://www.dataone.org

Docker: https://www.docker.com

DockerHub: https://hub.docker.com

DockerSpawner GitHub: https://github.com/jupyterhub/dockerspawner

EcohydroLib: https://github.com/selimnairb/EcohydroLib

Facebook API: https://developers.facebook.com/docs/apis-and-sdks

FigShare: https://figshare.com

Google API: https://developers.google.com/apis-explorer

Harvard Dataverse: https://dataverse.harvard.edu

HydroShare REST API: https://github.com/hydroshare/hydroshare/wiki/HydroShare-REST-API

NetCDF4 GitHub: https://github.com/Unidata/netcdf4-python

Numpy: https://www.numpy.org

Pandas: https://pandas.pydata.org

Seaborn: https://seaborn.pydata.org

Singularity: https://sylabs.io

SUMMA on the UCAR: https://ral.ucar.edu/projects/summa

xarray: http://xarray.pydata.org

XSEDE: https://www.xsede.org

Chapter 3

Comparing Containerization Approaches for Achieving Reproducible Environmental Modeling across Computing Environments

3.1 Introduction

The rapid advancement of computing has played an important role and offers both opportunities and challenges for reproducibility in computational research (de Lusignan & van Weel, 2006). On one hand, new tools and technologies have made possible complex modeling (Kerandi et al., 2018), deep learning (Shen, 2018), and interdisciplinary research (Laniak et al., 2013; Vogel et al., 2015). Additionally, with the possible exception of stochastic modeling, there is some level of certainty that if the same input data and model software or code are used on identical machines, it should result in the same output, even when the modeling software is very complicated (Sacks et al., 1989). On the other hand, creating "identical machines" can be very difficult in practice. When these computational models are moved to a new machine, scientists can often experience difficulties in reproducing the same model results (Monya; Baker, 2016; Essawy et al., 2020; Hothorn & Leisch, 2011; Wilson et al., 2017). This is because the way software is packaged, installed, and executed to reproduce complete computational environments and workflows is often very complicated and challenging, even when these steps are well documented (Garijo et al., 2013). To make matters worse, the rapid evolution and changing versions of software, especially open source software commonly used in many scientific communities, make computational reproducibility even more difficult (Epskamp, 2019). Just minor differences in computational approaches can result in fatal errors in re-executing computational environments and can have major influences on the analytical outputs. As a result, researchers have been highlighting the difference between what might be thought of as reproducible work, such as simply sharing data and workflow documents, and what is actually required for reproducible work: sharing computational environments and automated workflows (Beaulieu-Jones & Greene, 2017; Essawy et al., 2020; Kim et al., 2018).

To overcome this reproducibility gap, researchers have presented guidelines and principles (Choi et al., 2021; Essawy et al., 2020; Gil et al., 2016; Wilkinson et al., 2016) and developed various reproducible tools (e.g., Singularity, Kurtzer et al., 2017; Docker, Merkel, 2014; Sciunit, That et al., 2017) to increase the likelihood of reproducibility. For example, as online repositories that follow FAIR (Findable, Accessible, Interoperable, Reusable) guiding principles (Wilkinson et al., 2016) continue to mature, reproducibility research has led to a growing demand not only for data sharing with well-documented data, source code, software, and workflows, but also with tools for automatically encapsulating computational environments and workflows using containerization and literate programming (Kery et al., 2018; Knuth, 1984). For example, Bast (2019) suggested that source code management and containerization tools are needed to reproduce

computational environments while Goble et al. (2020) suggested the FAIR principles need end-to-end workflows to describe the execution of a computational process such as data collection, data preparation, data analysis, and modeling simulation. In hydrology, Hutton et al. (2016) recommended an online repository to easily find data and source code with unique persistent identifiers and computational workflows to describe the precise procedure among data and modeling processes. In addition, Hut et al. (2017) suggested the use of containerization tools and open interfaces to complement the preservation of computational environments suggested by Hutton et al. (2016).

Reproducibility of computational environments and automated workflows have been shown to be critical to filling the gaps of computational reproducibility in practice (Piccolo & Frampton, 2016; Rosenberg et al., 2020; Sandve et al., 2013). In order to reproduce computational environmental models, it is important to consider the fact that computational modeling software is actually comprised of multiple interdependent components: 1) the core model software, 2) other secondary software needed to support the modeling application, and 3) modeling workflows that capture the end-to-end modeling application. The core model software is the main computational engine for the environmental model and is most often developed using a compiled programming language such as Fortran, C, or C++. Other secondary software needed to support the modeling application can include a Graphical User Interface (GUI) or an Application Programming Interface (API) often programmed using an interpreted language like Python (Choi et al., 2021; Lampert & Wu, 2015; B. McDonnell et al., 2020; Volk & Turner, 2019). Finally, modeling workflows are an important component to capture the entire end-to-end process required to reproduce published modeling results. Literate programming is an increasingly popular means for creating modeling workflows as a narrative that combines code, documentation, and model output directly within a single narrative (Kery et al., 2018; Knuth, 1984; Pimentel et al., 2019). For example, Jupyter (Avila et al., 2020; Pérez & Granger, 2007) and RMarkdown (Baumer et al., 2014; Rstudio Team, 2020) are becoming increasingly used to conduct and document computational experiments.

Reproducing computational models is difficult in part because it requires a certain level of expertise in order to install and configure complete computational modeling setups. While most model developers may know the specific requirements to reproduce their environmental model on another computer, it is challenging to completely document this procedure for others to follow. To address this challenge, model developers have recently started using containerization tools such as Docker, Singularity, and Sciunit. Because these containerization tools can encapsulate complex software, developers and researchers can more easily and confidently create reproducible modeling studies that can be repeated across machines. While these containerization tools offer an important opportunity, it can be challenging for domain scientists to know how best to utilize these tools for different modeling use cases and computational experiments. Many containerization approaches exist, and these approaches can be executed in different computational environments that include both local compute resources (i.e., the researcher's personal computer) and a growing number of remote computing resources (i.e., high performance computing (HPC) clusters or cloud-computing environments) available for environmental modeling.

Thus, we focus in this paper on comparing different containerization approaches for advancing reproducible environmental modeling. In total, 11 approaches are considered, with each using a different combination of containerization software and computational environments. In the methodology section, we describe the characteristics and typical procedures for each approach. We also explain the methodology for evaluating and comparing the approaches. Then, in the results section we present the evaluation results generated from a hydrologic modeling case study that uses the Structure for Unifying Multiple Modeling Alternative (SUMMA) (Clark et al., 2015a) hydrological model. We then discuss the benefits and weaknesses of each of the 11 reproducible approaches and, in the conclusion section, summarize best practices for using the approaches to achieve different modeling objectives. Lastly, we list remaining knowledge gaps that require future research to develop more reliable and efficient containerization approaches for reproducible environmental modeling.

3.2 Methodology

3.2.1 Introducing the Reproducible Environmental Modeling Approaches

When discussing reproducible environmental modeling, it is first necessary to define terminology and components of typical environmental modeling workflows. First, we consider a computational model as consisting of three primary software components, as introduced earlier: 1) the core model software, 2) other secondary software needed to support the modeling application, and 3) a modeling workflow that links the core and secondary software. Reproducibility approaches may address one or more of these components and, therefore, may not necessarily address the entire end-to-end modeling workflow. This is described further as we introduce the 11 reproducible approaches compared in this study. Across these 11 reproducible approaches, we used different software tools to achieve reproducibility, namely GNU Make, Conda Virtual Environment (hereafter Conda VE), Docker, Singularity, Sciunit, and Jupyter notebooks. Through different combinations of these tools and using different computational environments, we arrived at the 11 representative reproducible approaches considered in this study.

The 11 reproducible approaches consist of five strategies using local computing resources and six strategies using remote computing resources (Table 3.1). The first local approach compiles the model software using GNU Make and encapsulates other secondary software using Conda VE to support the modeling application. This can be thought of as a standard approach commonly used now by model developers that does not adopt containerization. The second approach introduces containerization for only the core model software component of the workflow using Docker as the containerization tool. The third approach uses containerization for not only the core model software, but also the secondary software supporting the model, again using Docker as the containerization tool. The fourth approach builds from the third by keeping the same containerization strategy, but using Singularity as the containerization tool rather than Docker. The fifth approach is like the third and fourth approaches, except Sciunit is used as the containerization tool.

The six remote approaches leverage either HPC clusters or cloud computing environments as the computational resource (Li, 2020; Prasad et al., 2020; Shuler & Mariner, 2020). The sixth approach uses the CUAHSI JupyterHub (hereafter CJH), which is a cloud computing environment on the Google Cloud Platform specifically designed to support research and education in the water sciences. The seventh approach uses CyberGIS-Jupyter for Water (hereafter CJW), which is a tailored CyberGISX instance to support data-intensive and reproducible research in the environmental modeling community built on the XSEDE Jetstream computational resource. The eighth and ninth approaches use CJH and CJW, respectively, with Sciunit as the containerization tool for capturing the workflow. The tenth approach uses a different computational environment and containerization approach: Binder. Binder is an online JupyterHub for building and sharing reproducible and interactive computational environments from online repositories. It uses Docker as a containerization technology, although it attempts to hide the user from the details of Docker containerization to lower the barrier to entry. The last approach uses a HPC cluster with Singularity to support the use of multiple cores for parallel computation. A common tool across the 11 reproducible approaches is Jupyter, a literal programming approach for capturing modeling workflows. In the following subsection, we describe the specific procedures and characteristics of each approach in further detail.

Table 3.1. The 11 representative reproducible approaches using different combinations of software tools and computational environments

<table>
<tr><th rowspan="2">Approach No.</th><th colspan="2" rowspan="2">Local and Remote Computational Environments</th><th colspan="3">Combination of Software Tools and Modeling Workflows</th></tr>
<tr><th>1) Core Model Software</th><th>2) Secondary Software</th><th>3) Modeling Workflows</th></tr>
<tr><td>1</td><td rowspan="5">LOCAL</td><td rowspan="5">Virtual Box</td><td>GNU Make</td><td>Conda Virtual Environment</td><td rowspan="4">Jupyter Notebook</td></tr>
<tr><td>2</td><td>Docker</td></tr>
<tr><td>3</td><td colspan="2">Docker</td></tr>
<tr><td>4</td><td colspan="2">Singularity</td></tr>
<tr><td>5</td><td colspan="2">Sciunit</td></tr>
<tr><td>6</td><td rowspan="6">REMOTE</td><td>CUAHSI JupyterHub</td><td colspan="2">Docker</td><td rowspan="2">Jupyter Notebook</td></tr>
<tr><td>7</td><td>CyberGIS-Jupyter for water</td><td colspan="2">Docker</td></tr>
<tr><td>8</td><td>CUAHSI JupyterHub</td><td colspan="2">Sciunit</td><td></td></tr>
<tr><td>9</td><td>CyberGIS-Jupyter for water</td><td colspan="2">Sciunit</td><td></td></tr>
<tr><td>10</td><td>Binder</td><td colspan="2">Docker</td><td>Jupyter Notebook</td></tr>
<tr><td>11</td><td>HPC</td><td colspan="2">Singularity</td><td></td></tr>
</table>

3.2.2 Local Reproducible Approaches

In this subsection, we describe the five local reproducible approaches for sharing modeling environments and workflows. For all five local approaches, we used Virtual Box to create a Linux virtual environment (Ubuntu 20.04 LDT) on a Windows operating system (Table 3.2) with a single-core processor. Using this computer setup, we reproduced approaches 1-5 as shown in Table 3.1.

Table 3.2. Specification of the base local computational environment

Specification	Descriptions
Processor	Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz
RAM	15.6GB
Base Operating System	Window 10
Linux Emulator	VirtualBox 5.2.12
Linux Operating System	Ubuntu 20.04 LDT
Number of CPU Core	1

3.2.2.1 Approach 1: Compiling the Core Model Software

The first approach compiles the core model software using GNU Make and encapsulates the other secondary software using Conda VE. This approach is a very common approach for reproducing hydrologic models (Peckham et al., 2013). GNU Make is used to create a core model software executable with a configuration file called Makefile. The Makefile provides the procedure for how to build the source code on a particular operating system. According to the technical specification of hydrologic models in Community Surface Dynamic Modeling System (CSDMS), most hydrologic models were developed by compiled programming languages (Fortran, C, and C++) (Table 3.3), perhaps because of computing speed and the use of legacy source codes. Therefore, we used GNU Make as the approach to reproducing hydrologic model software. Creating model executables using GNU Make requires a substantial time investment at first because it requires an understanding of the necessary dependencies, file paths, and environmental variables.

Table 3.3. The programming languages in the popular hydrologic models (https://csdms.colorado.edu/wiki/Hydrological_Models)

Models	Programming Languages	Models	Programming Languages
DHSVM	C	PRMS	Fortran77, C
Delft3D	Fortran77, Fortran90, C, C++	ParFlow	Fortran90, C
GSFLOW	Fortran90, C	RHESSys	C
PIHM	C, C++	SWAT	Fortran
VIC	C	SWMM	C

In addition, Conda VEs are used to encapsulate other secondary software needed to support the modeling application. While Python has a lot of advantages in that we are able to use a vast number of Python packages, it requires package and environment management to avoid version conflicts between each package. There are several package and environment managers to overcome version conflicts in Python. The most commonly used ones are pip, virtualenv, and Conda. Pip is a package manager and most Python users use pip to install Python packages, but pip does not support environment management. Virtualenv is an environment manager to create isolated virtual environments, but virtualenv does not support package management. Conda is a Python environment and package manager capable of providing isolated virtual environments,

installation of software packages, and a record of the exact versions of open source libraries used within a virtual environment. Therefore, we used Conda to encapsulate other software needed to support modeling applications, especially Python APIs.

Figure 3.1 shows the general procedure of "*Approach 1: Compiling the Core Model Software*" using the GNU Make tool and Conda VE. To describe the approach, we divided Figure 3.1 into two parts: developer work and user work. For the developer work, the first reproducible step for environmental model software has three substeps: 1) creating a Makefile, 2) compiling and building a core software executable, and 3) sharing the source code and Makefile on an online repository such as GitHub or HydroShare. The second step is for Python-based model APIs that require an environment.yml file, which is a configuration file containing a list of Python packages needed to create the Conda VE. These two steps encapsulate the computational environment for the environmental model. The final step encapsulates the modeling workflows by using Jupyter notebooks that document the end-to-end modeling steps. Once the model has been captured, developers need to share the model input files needed to reproduce the original study.

After the developer's work is complete, users can reproduce the modeling environment and workflows. The steps in this process are as follows: 1) download the source code and Makefile for building the environmental model, 2) edit the Makefile to set the paths to the configuration files and software dependencies for the environmental model software on the user's computer, and 3) compile and build the executable of core model software. Next, users need to download the Jupyter notebooks that document every step in the workflow including installing Python-based model APIs, downloading model input data, and executing the environmental model. Compared to the developer work, the user work becomes simpler because Jupyter notebooks are able to document most of the workflow, except for the step of compiling the core model software.

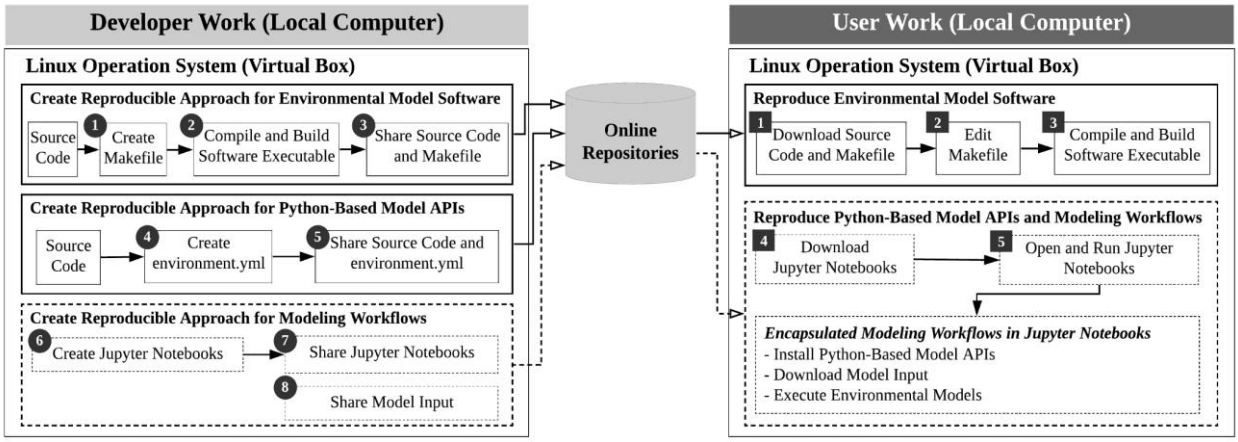

Figure 3.1. A general procedure of "Approach 1: Compiling the Core Model Software"

3.2.2.2 Approach 2: Containerizing the Core Model Software only with Docker

The second approach containerizes only the core model software using Docker. While Docker may be somewhat difficult for those without software development expertise, it is a revolutionary tool in that it allows for 1) ease of use to set up and apply computational

environments, 2) speed to execute applications, 3) ease of sharing via DockerHub, and 4) modularity and scalability to break, scale, and update Docker images (Boettiger, 2015). In environmental modeling, we can use Docker to containerize computational environments, software packages, libraries, model input, and model output into a Docker image. Python has become a popular language for environmental modelers and many modelers will be familiar with Conda VE to encapsulate Python-based environments without using containerization tools like Docker. In addition, Conda VE is more compatible than Docker to install new software to a VE, provided that the software is available through Conda. Thus, we use Docker to containerize the core model software to easily reproduce it but use Conda to encapsulate secondary software, which is Python-based and available through Conda.

Figure 3.2 shows the general procedure of "*Approach 2: Containerizing the Core Model Software only with Docker.*" In developer work, the first reproducible step for the environmental model software has three substeps: 1) creating a Dockerfile, 2) creating a Docker image, and 3) sharing a Docker image on online repositories such as DockerHub. The next two steps for the Python-based model APIs and modeling workflows are the same as "*Approach 1: Compiling the Core Model Software*." However, the user work is simpler than "*Approach 1*" because we used Docker to containerize the environmental model software. Therefore, users only need to install Docker using the simple command "*sudo apt install docker.io*," then open and run the Jupyter notebooks including installing Python-based model APIs, downloading the model input and the Docker image, and executing the environmental model.

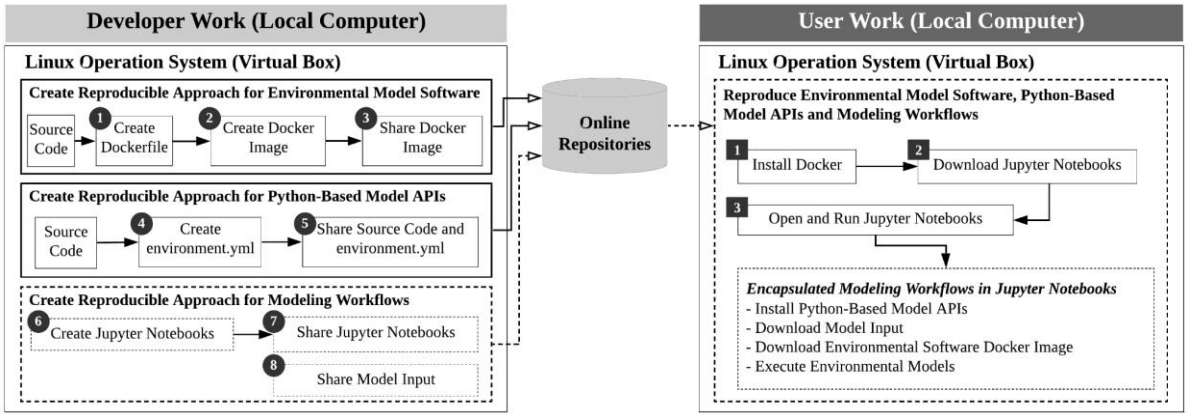

Figure 3.2. A general procedure of "Approach 2: Containerizing the Core Model Software only with Docker"

3.2.2.3 Approach 3: Containerizing All Software with Docker

The third approach containerizes all software in the end-to-end workflow in Docker: the core model software, other secondary software, and workflows for running the experiment. After creating a Dockerfile, it is easy to share and recreate a Docker image from it on different machines. While there are some benefits in "*Approach 3*" compared to earlier approaches, there are some inconveniences as well. First, to install additional software or Python packages permanently to the secondary software requires changes to the Docker image, which can be time consuming to rebuild. Also, if users install new Python packages in the Docker image without rebuilding the

Docker image, Docker requires Python packages to be reinstalled when the container is started, which can also be time consuming. Finally, researchers need to save their modeling results outside of the Docker image before stopping the Docker container, or else the results will be lost as the Docker container is temporary.

Figure 3.3 shows the general procedure of "*Approach 3: Containerizing All Software with Docker*." In the developer's work, the first step is creating Jupyter notebooks to containerize workflows into the Docker container. Next, developers must create a Dockerfile that includes each command needed to containerize the environmental model software, Python-based model APIs, and modeling workflows. In this approach, users only need to install the Docker tool and run the Docker image because the Docker image has all of the required dependencies. Then users can open and run Jupyter notebooks to reproduce the environmental model software, Python-based model APIs, and the modeling workflows.

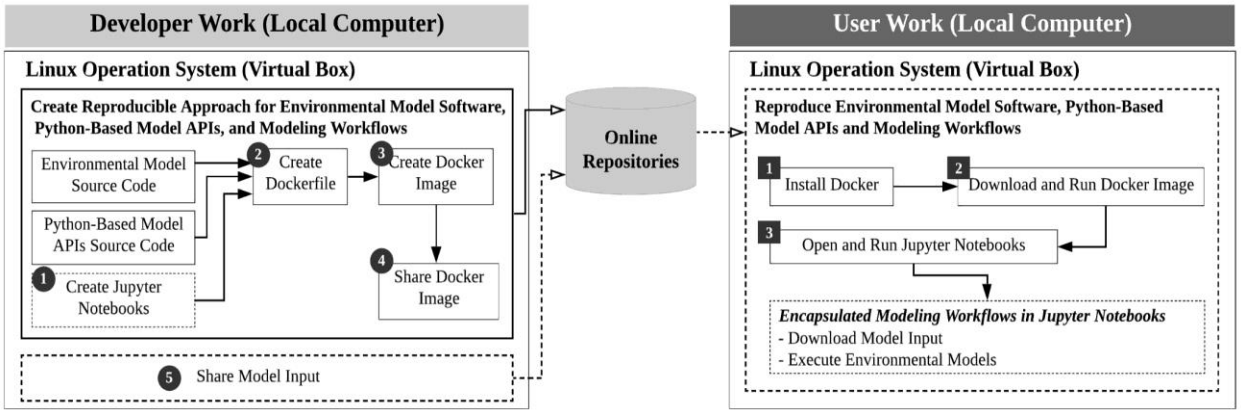

Figure 3.3. A general procedure of "Approach 3: Containerizing All Software with Docker"

3.2.2.4 Approach 4: Containerizing All Software with Singularity

The fourth approach containerizes all software using Singularity, which is another containerization tool to support computational environments. Singularity is more popular to use in HPC environments over Docker for avoiding security concerns, and we will describe the characteristics of Singularity in detail later in "*Approach 11: using a HPC Cluster*." A Singularity image can be created using a Definition file, similar to Dockerfile, which defines an operating system and various software requirements. Recently, many HPC environments, such as XSEDE (an HPC resource on the Extreme Science and Engineering Discovery Environment), Rivanna (HPC at University of Virginia), and NCAR (National Center for Atmospheric Research) HPC support Jupyter and Singularity. Thus, it is possible to utilize a Singularity image through the Jupyter user interface by linking the Singularity image and Jupyter notebooks through the Jupyter kernel. In addition, the docker2singularity library can support automatically converting a Docker image to a Singularity image.

Figure 3.4 shows the general procedure of "*Approach 4: Containerizing All Software with Singularity*." In the developer work, the first step is creating a Definition file to create a Singularity

image that includes a dependency list. Next, developers need to make a "kernel.json" file to link a Jupyter kernel with the Singularity image and Jupyter notebooks. Next, developers can share the Singularity image through online repositories or Singularity Hub. Also, developers have to create and share Jupyter notebooks and the model input for the modeling workflows. After the developer's work is complete, users need to first download the Jupyter notebooks, then open and run the Jupyter notebooks. For automated workflows, Jupyter notebooks handle the rest of the workflows such as downloading the Singularity image of the core model software, creating the Jupyter kernel to create a link between the Singularity image and Jupyter notebooks, downloading the model input data, and executing the environmental model.

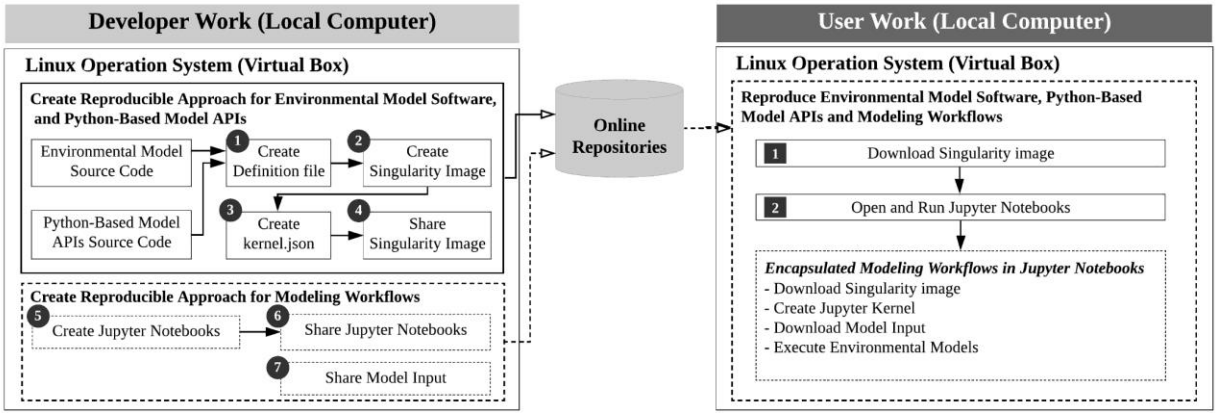

Figure 3.4. A general procedure of "Approach 4: Containerizing All Software with Singularity"

3.2.2.5 Approach 5: Containerizing All Software and Modeling Workflows with Sciunit

The fifth approach is similar to the third and fourth approaches, except Sciunit is used as the containerization tool. Sciunit was developed by the Geotrust project funded through the US National Science Foundation (NSF) EarthCube program and is a tool to ease the process of containerizing, sharing and tracking scientific applications, lowering the barrier to entry for researchers. While Docker and Singularity require document scripts for creating container images, Sciunit (That et al., 2017) can capture every dependency by monitoring software usage during execution of the analysis workflow. Sciunit then generates container-like packages called sciunits that contain all software used during the analysis workflow and that are portable across machines and different computational environments. These sciunit packages are also more lightweight than other containerization tools because Sciunit is able to trace the program execution and captures only those software dependencies that the model run used. Therefore, Sciunit is helpful for researchers who are non-experts to reproduce model run, but also experts can benefit from Sciunit by creating minimal containers to reproduce computational experiments.

Figure 3.5 shows the general procedure of "*Approach 5: Containerizing All Software and Modeling Workflows with Sciunit.*" Developers first need to create a Jupyter notebook to encapsulate Sciunit workflows. Next, developers need to create a Sciunit image using the created programming code and the Jupyter notebook. After that, developers can share the Sciunit image and the Jupyter notebook. Users can then download a Sciunit image and a Jupyter notebook and

only need to open and run a Jupyter notebook. Unlike other approaches, users do not need to download the model input as the Sciunit image includes the model input and all the software dependencies.

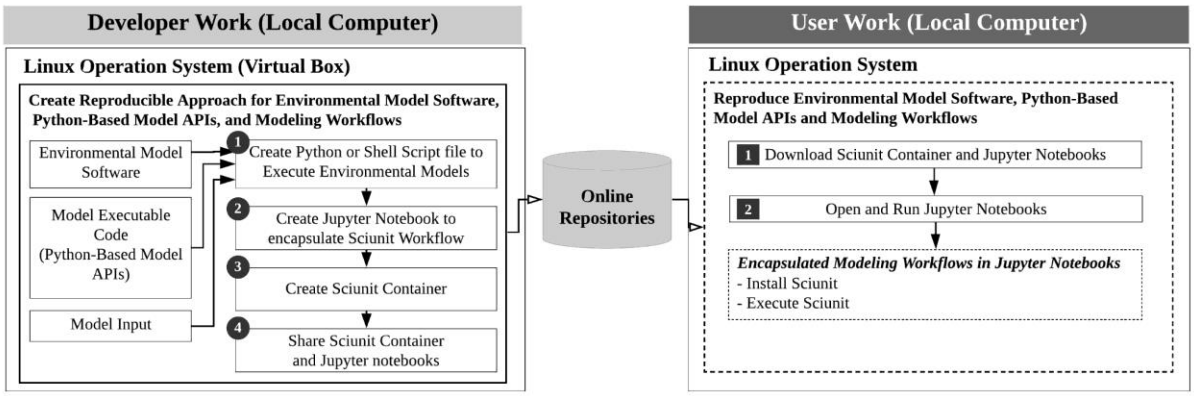

Figure 3.5. A general procedure of "Approach 5: Containerizing All Software and Modeling Workflows with Sciunit"

3.2.3 Remote Reproducible Approaches

Many science gateways and virtual research environments (Prasad et al., 2020) and many large companies (Google: Google Colab, Microsoft: Azure Notebooks, and Amazon: SageMaker) have created advanced remote computing environments to support computational research and education (Prasad et al., 2020). In this study, we considered four remote environments (CJH, CJW, HPC, and Binder) and six approaches for reproducible modeling across these remote environments using different containerization tools.

3.2.3.1 Approaches 6 and 7: Using CUAHSI JupyterHub and CyberGIS Jupyter for water

The sixth and seventh approaches use CJH and CJW, respectively, which are Docker-based remote environments. Docker-based remote environments can provide consistent preconfigured modeling environments and there has been a rapid adoption of Docker-based remote environments in industry and academic fields (Prasad et al., 2020). CJH and CJW are two examples of such remote environments. To add new Docker images into CJH and CJW, model developers need to share Dockerfiles that have lists of the core model software and other secondary software with CJH and CJW development teams. Then the CJH and CJW development team can review and deploy the new Docker images into CJH and CJW. After deploying the new Docker images, users can easily use the preconfigured environments for their modeling work. Also, CJH supports the installation of a custom Conda VE, so users can easily apply new Python or other packages in a consistent environment (Choi et al., 2021).

Figure 3.6 shows the general procedure of deploying environmental models and Python-based APIs on CJH and CJW from the developer's perspective and how to execute model software using preconfigured modeling environments from the user's perspective. The developer must create a Dockerfile similar to "*Approach 2*" or "*Approach 3*" and may use GitHub to add a new

Dockerfile as a pull request to the CJH or CJW GitHub. After sending a pull request to the GitHub repository of CJH or CJW, the Dockerfile needs to be reviewed by CJH or CJW development team to deploy a new Docker image. After finishing the developer's work, users only need to log into CJH or CJW and run Jupyter notebooks because modeling environments are preconfigured and shared through environmental profiles of CJH or Jupyter kernels of CJW.

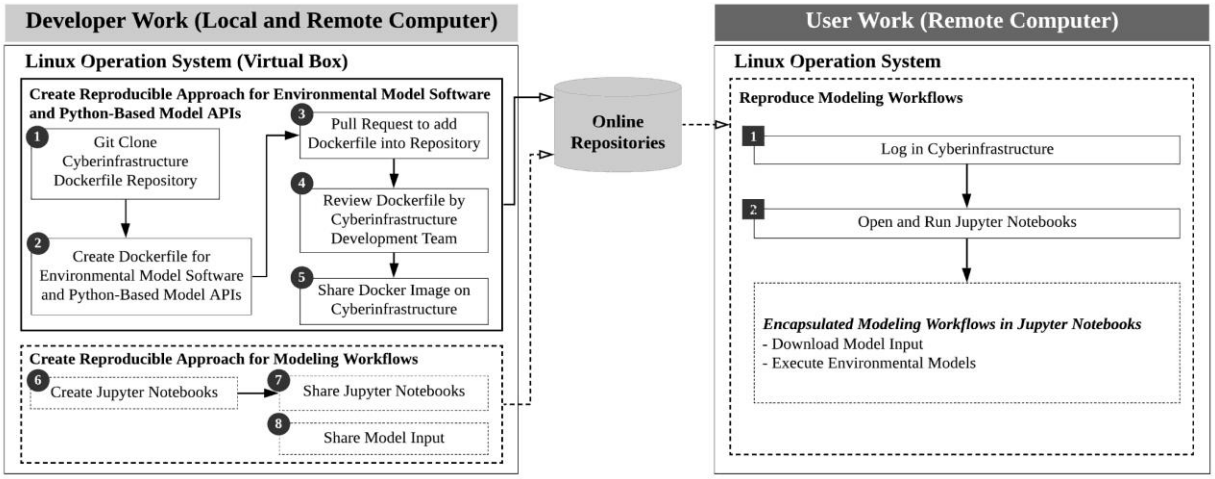

Figure 3.6. A general procedure to create "Approach 6 and 7: Using CUAHSI JupyterHub and CyberGIS Jupyter for water"

3.2.3.2 Approach 8 and 9: Using Sciunit in CJH and CJW

The eighth and ninth approaches use Sciunit in CJH and CJW, respectively. Unlike Docker and Singularity, Sciunit is an installable package in both local and remote Linux operating systems without administrative privileges. Therefore, even without deploying preconfigured computational environments in remote environments, researchers can containerize the core model software, other secondary software, and modeling workflows after simply installing Sciunit. The general procedure of Sciunit is the same with "*Approach 5: Containerizing All Software and Modeling Workflows with Sciunit.*" Therefore, we have omitted additional explanation of the approach including a general procedure figure since it repeats what is shown in Figure 3.5.

3.2.3.3 Approach 10: Using Binder

The tenth approach uses Binder, which is an open source web service to allow creating sharable, interactive, and reproducible environments (Jupyter Project et al., 2018). Binder was developed by integrating three tools: 1) JupyterHub deployed using Kubernetes, 2) Online open source repositories (GitHub, Figshare, Zenodo, and HydroShare) combined with a Repo2Docker library (Forde et al., 2018), and 3) BinderHub, a web-interface to bind JupyterHub and Repo2docker using user-defined kernels and interactive sessions. JupyterHub and Kubernetes support scalable interactive user sessions to handle many user sessions and sustainable user work. Repo2Docker is a lightweight tool to convert online open-source repositories into a Docker image that can be run with JupyterHub using various configuration files such as enivironment.yml,

setup.py, install.R, postBuild, and Dockerfile. By combining these tools, Binder can create containers to encapsulate the core model software and other secondary software, and generate user sessions to run the computational workflows expressed as Jupyter notebooks. Also, Binder can provide a URL to share with others that allows them to interact with the remote Binder environment. However, currently Binder implementations like MyBinder.org do not support persistent user sessions because sessions are ephemeral.

Figure 3.7 shows the general procedure of "*Approach 10: Using Binder*." First, the developer must create a configuration file that is supported by Binder to encapsulate the environmental model software and Python-based model APIs used by the model. Next, the developer must create Jupyter notebooks to document the modeling workflows. Then, the developer needs to share configuration files and the Jupyter notebooks through an online repository such as GitHub, Figshare, Zenodo, or HydroShare. After that, the developer uses the MyBinder website (https://mybinder.org) to create a remote modeling environment for the modeling setup. Finally, the developer can share the Binder URL pointing to the remote modeling environment with end users.

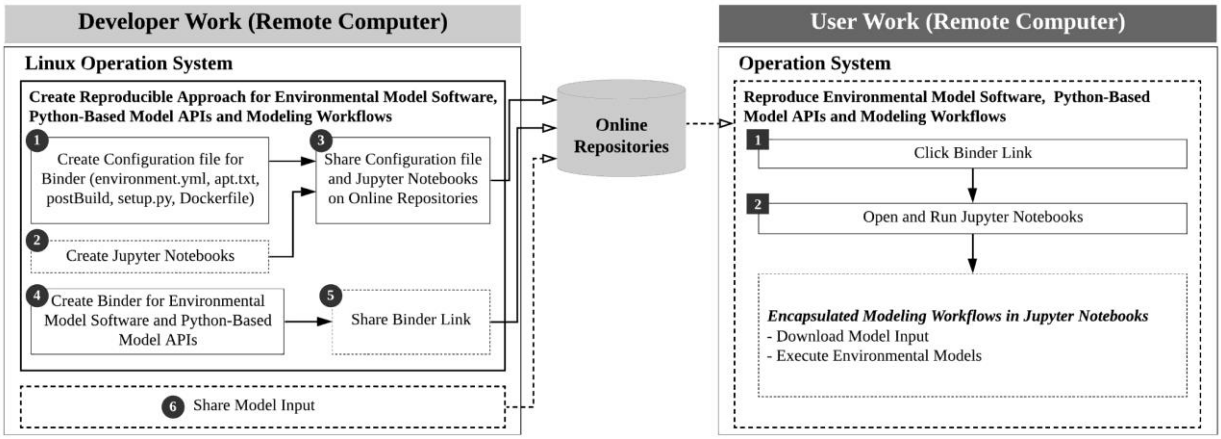

Figure 3.7. A general procedure to create "Approach 10: Using Binder"

3.2.3.4 Approach 11: Using a HPC Cluster

The final approach uses a HPC cluster with Singularity to perform the model computation in parallel. While Docker is the standard containerization tool, it does not meet the needs for scientific computing in HPC environments (Kurtzer et al., 2017). A major limitation is security concerns with using Docker in HPC environments because Docker requires root access to create and execute Docker containers. To overcome these problems, Singularity was developed to support portable and flexible computational environments without security risks, particularly for the use case of HPC modeling (Kurtzer et al., 2017).

Figure 3.8 shows the general procedure of "*Approach 11: Using a HPC Cluster*." Most steps are the same as Approach 4 but there are two important differences. First, developers need to upload a Singularity image into the HPC environment to use the containerized modeling environment. Second, users only need to create a Jupyter kernel to establish a link between Jupyter

notebooks of the modeling workflow and the Singularity image that the developer uploaded into the HPC that includes the model code.

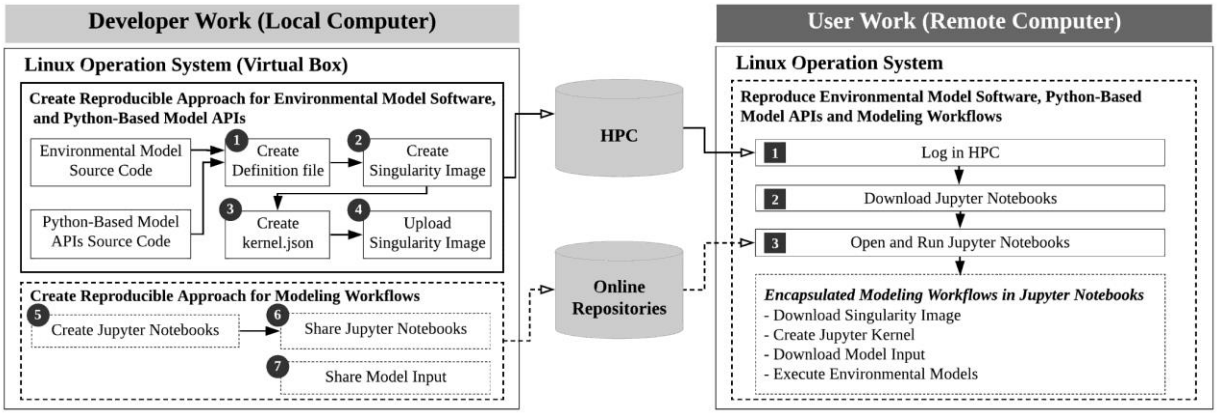

Figure 3.8. A general procedure to create "Approach 11: Using a HPC Cluster"

3.2.4 Evaluation

To evaluate each of the 11 approaches, we first selected an example modeling case study including three components: SUMMA (Clark et al., 2015a) as the core model software, pySUMMA (Choi et al., 2021) and other Python packages as other secondary software, and Jupyter notebooks as modeling workflows. Each of these three components is described in further detail in the coming subsections. Then, we defined quantitative and qualitative criteria to evaluate the 11 reproducible approaches. These evaluation criteria are also described in the subsections below.

3.2.4.1 The Example Modeling Case Study

SUMMA was selected as the case study model for this study because it is a general environmental model that enables the controlled and systematic evaluation of multiple model representations of hydrologic processes and scaling behavior through a flexible hierarchical spatial structure. SUMMA was developed in the Fortran programming language; therefore, we need a Fortran compiler (i.e., gfortran) to compile SUMMA source code. Also, SUMMA requires the NetCDF (Network Common Data Form) and LAPACK (Linear Algebra PACKage) libraries. The NetCDF library (libnetcdff.*) supports creating, accessing, and sharing data stored in a NetCDF format. The LAPACK library provides a series of routines for linear algebra operations, including matrix solvers. These libraries are considered core software for the model because they are required for the model to run. SUMMA Makefile and Dockerfiles are shared through the SUMMA GitHub repository (SUMMA GitHub, 2021) to support compiling SUMMA source code and creating a SUMMA Docker image. Also, the created SUMMA Docker image is shared via DockerHub (SUMMA DockerHub, 2021).

Other secondary software, not required to run SUMMA but convenient for working with SUMMA models, includes pySUMMA: a Python-based SUMMA model API. pySUMMA allows for programmatic control of the model configuration, execution, and visualization of SUMMA models. Currently, pySUMMA can be installed from either a conda command (e.g., "*conda install*

–c conda-forge pysumma") or a pip command (e.g., "*pip install pysumma*"). Users can also download the pySUMMA source code from the pySUMMA GitHub and install it manually using "*environment.yml*" for conda install or "*setup.py*" for pip install. The "*environment.yml*" and "*setup.py*" files have the lists of pySUMMA dependencies for each installation method, thus making it possible to reproduce the pySUMMA environment with dependencies on a new machine.

Finally, for modeling workflows we used Jupyter notebooks to document SUMMA and pySUMMA modeling workflows through a mix of formatted text, mathematical equations, and executable code with in-line visualizations. We created Jupyter notebooks for each reproducible approach to encapsulate reproducible artifacts and modeling workflows for SUMMA and pySUMMA. These notebooks are available as described in the Data and Software Availability section.

3.2.4.2 Quantitative Performance

The following quantitative measures were used to evaluate each of the 11 approaches. (1) Complexity considers the total number of steps weighted by the difficulty in reproducing each step and is an important metric for lessening the burden of reproducibility work for researchers (Atmanspacher et al., 2014). (2) The size of computational artifacts takes into account storage requirements for storing and sharing each approach (Craig & Victoria., 2020; Kovács, 2017). (3) The computational time measures the wall time required to execute the model and can vary significantly across approaches (Kozhirbayev & Sinnott, 2017).

In the complexity metric, we measured the number of steps and the level of difficulty to evaluate the complexity of each approach considering both developer work and user work. We used the general procedures for the 11 reproducible approaches shown in Figures 3.1 through 3.8 to count the steps. We defined the level of difficulty for each step using three levels: Easy, Medium, and Difficult. Considering the time and expertise required for completing a medium task, we set the score to be five times that of an easy task. Likewise, we estimated that a difficult task was twice the score of a medium task. Therefore, we set an Easy task to a score of 1, Medium to a score of 5, and Difficult to a score of 10.

In the size metric, we measured how much space is used to store all components of the reproducible environment. We only considered the size metric for the five local approaches because the size of the preconfigured computational artifacts in remote environment will be determined by the specific technical implementation in that remote environment.

Finally, in the computational time metric, we measured the execution time across all 11 reproducible approaches. In this performance metric, we measure time considering the capabilities of the reproducible tools themselves in a local environment and extensible capabilities to connect with remote environments such as using Dask (Rocklin, 2015) for parallel computing. As model software is rapidly becoming complicated with the use of large datasets (Addor et al., 2017; Boulmaiz et al., 2020; Kerandi et al., 2018), this computational time performance is critical. For

example, many environmental models require multiple runs for calibration, sensitivity and uncertainty analysis.

We measured these three metrics using a SUMMA modeling case study. Clark et al. (2015b) created a set of thirteen SUMMA modeling datasets exploring various hydrologic modeling scenarios. Based on these datasets, we created four scenarios (Table 3.4) using two datasets to reproduce Figures 7, 8, and 9 in Clark et al. (2015b) and to measure computational time across our eleven approaches. These four scenarios include additional model simulations based on Clark et al. (2015b) to compare computational time across different simulation periods (15 vs 75 months) and intensity (3 vs 9 ensemble simulations). The first scenario is a single simulation during 15 months for analyzing the impact on ET using the S*imple Resistance* method, which is one of stomatal resistance parameterizations in SUMMA. The second scenario is nine ensemble simulations for analyzing the impact on ET using 1) three different stomatal resistance parameterizations which are *Simple Resistance*, *Ball-Berry* (Ball et al., 1987), and *Jarvis* (Jarvis, 1976) and 2) three different values (1.0, 0.5, 0.25) of the *root exponential distribution* parameter. From the first and second scenarios, we reproduced Appendix Figure A.1 from Figures 7 and 8 in Clark et al. (2015b) to verify the reproducible approach in this study. The third scenario is a single simulation for a 75 month period for analyzing the impact on runoff using the *1d Richards* method (Celia et al., 1990), which is one of the lateral flow parameterizations in SUMMA. The last scenario is three ensemble simulations for analyzing the impact on runoff using three different lateral flow parameterizations: *1d Richards, Lumped Topmodel,* and *Distributed Topmodel* (J. Duan & Miller, 1997). From the third and fourth scenarios, we reproduced Appendix Figure A.2 from Figure 9 in Clark et al. (2015b).

Table 3.4. SUMMA simulation scenarios for comparison of computational time on eleven reproducible approaches

Scenario	Descriptions
(d) Scenario -1	☐ A single simulation (simple resistance method) ☐ Simulation periods: 2006-07-01 ~ 2007-09-30 (15 months)
(e) Scenario -2	☐ Ensemble simulations (9 simulations) - 3 different parameterizations (Simple Resistance, Ball-Berry, and Jarvis) × 3 different parameters (Root Exponential values 1.0, 0.5, 0.25) ☐ Simulation periods: 2006-07-01 ~ 2007-09-30 (15 months)
(f) Scenario -3	☐ A single simulation (1d Richards) ☐ Simulation periods: 2002-07-01 ~ 2008-09-30 (75 months)
(d) Scenario -4	☐ Ensemble simulations (3 simulations) - 3 different parameterizations (1d Richards, Lumped Topmodel, and Distributed Topmodel) ☐ Simulation periods: 2002-07-01 ~ 2008-09-30 (75 months)

3.2.4.3 Qualitative Performance

Qualitative performance was analyzed to complement limitations of using quantitative performance measures alone. In evaluating qualitative performance, we describe the strengths and weaknesses of each approach through experiences learned from applications of the 11 different reproducible approaches from both the developer and user perspectives. We then classified various

environmental modeling objectives into two broad categories: 1) education and 2) research because the purpose of reproducibility is to practice and extend previous knowledge based on confirmation of published results (Craig & Victoria., 2020; Prasad et al., 2020). Finally, based on these strengths and weaknesses and the two broad categories, we present recommendations for best practices when using the containerization approaches.

3.3 Results

3.3.1 Quantitative Performance

3.3.1.1 Complexity

Table 3.5 is an example of the complexity metric output for "*Approach 2: Containerizing the Core Model Software only with Docker*." In the developer work, we divided three categories considering three primary components for this study. In the "1. Create a Reproducible Approach for SUMMA" step for the core model software, we measured "1.1 Create SUMMA Dockerfile" as a "Difficult" because creating the Dockerfile requires understanding the required dependencies, executable locations of software, and Docker commands. "1.2 Create SUMMA Docker Image" and "1.3 Share SUMMA Docker Image" were measured as "Easy" because this procedure can be completed using simple Docker commands. In the "2 Create a Reproducible Approach for pySUMMA" step for the secondary software, we measured "1.2 Create pySUMMA environment.yml" as "Medium" because often users can simply install pySUMMA using pip or Conda, but other times users have to install the software manually because of version conflicts with Python or other Python packages. In the final step of the developer work, we measured "3.1 Create Jupyter Notebooks" as a "Difficult" because this process requires significant effort to encapsulate SUMMA modeling workflows considering the interaction with online repositories and software. In the user work, every step was measured as "Easy" because every reproducible component is shared online, allowing users to complete each step using simple commands and Jupyter notebooks. Similar tables to Table 3.5 for the other approaches are included in the Appendix.

Table 3.5. An example of reproducible approaches for "Approach 2: Containerizing the Core Model Software only with Docker" (Tables for other approaches are in Appendix)

Developer Work

Steps	Level of Difficulty	Scores
1. Create a Reproducible Approach for SUMMA		**12**
1.1 Create SUMMA Dockerfile	Difficult	10
1.2 Create SUMMA Docker Image	Easy	1
1.3 Share SUMMA Docker Image	Easy	1
2. Create a Reproducible Approach for pySUMMA		**6**
2.1 Create pySUMMA environment.yml	Medium	5
2.2 Share Source Code and environment.yml	Easy	1
3.Create a Reproducible Approach of Modeling Workflows		**12**
3.1 Create Jupyter Notebooks	Difficult	10
3.2 Share Jupyter Notebooks	Easy	1
3.3 Share SUMMA Input	Easy	1
Total Score		**30**

User Work

Steps	Level of Difficulty	Scores
1.Reproduce SUMMA, pySUMMA, and Modeling Workflows		**3**
1.1 Install Docker	Easy	1
1.2 Download Jupyter Notebooks	Easy	1
1.3 Open and Run Jupyter Notebooks - Install pySUMMA - Download SUMMA input - Download SUMMA Docker Image - Execute SUMMA	Easy	1
Total Score		**3**

Figure 3.9 compares the total scores of complexity for the nine reproducible approaches considered in this metric except for two Sciunit approaches "*Approach-8 and 9*" because of the same complexity with "*Approach-5*." Overall, user work is much simpler than developer work, as expected, and we can see that creating appropriate reproducible approaches is important and helpful for the reproducibility of other researchers (Piccolo & Frampton, 2016). In terms of developer work, "*Approach-1: Compiling the Core Model Software"* is the most complicated approach because it does not use containerization tools, so in this approach the developer needs to reproduce every step individually. Compared to this, "*Approach-5, 8 and 9: Using Sciunit*" are the simplest approaches. In terms of user work, most approaches are simple except for *"Approach-1: Compiling the Core Model Software."* "*Approach-6 and 7: CJH and CJW*", "*Approach 10: Using Binder*", and "*Approach-5, 8 and 9: Using Sciunit*" are the simplest approaches because every dependency for environmental modeling is preconfigured into containers. Sciunit is the simplest and most straightforward from both the developer and user perspectives, because Sciunit can containerize every modeling environment and workflow into a Sciunit container using model execution code that was created for the original study, with no additional work. Users can reproduce published results using Sciunit containers easily using simple Sciunit commands in a Jupyter notebook.

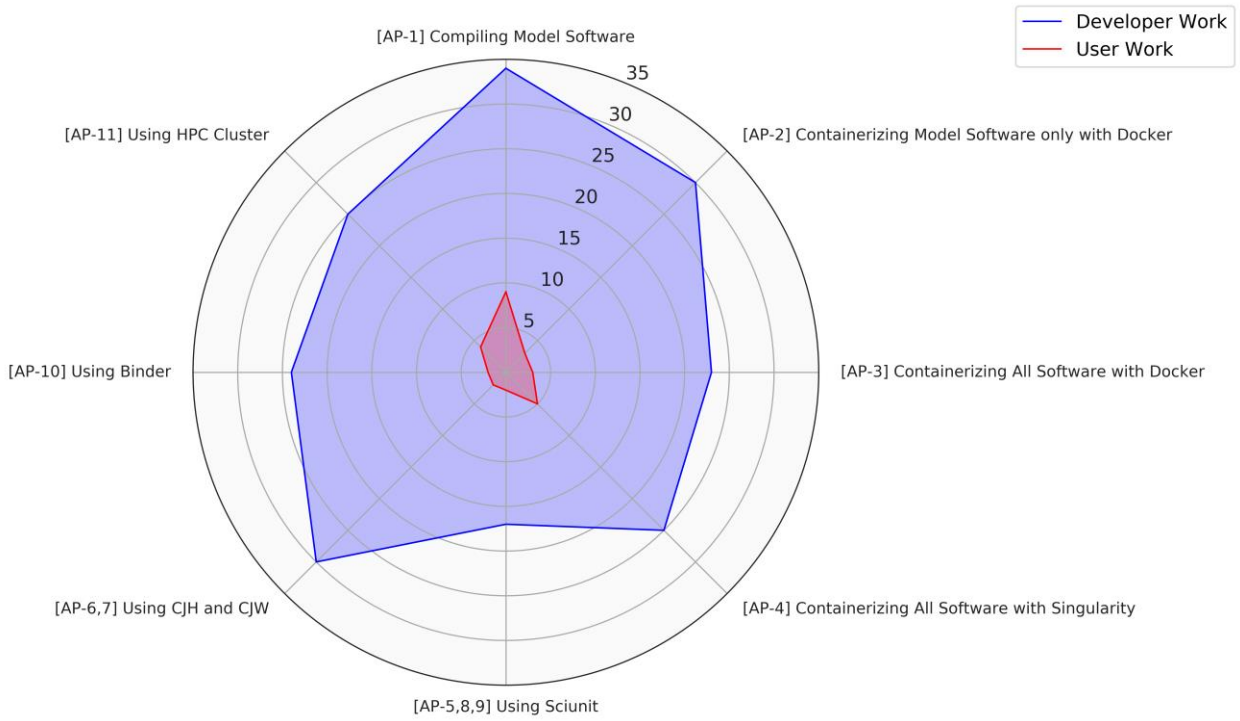

Figure 3.9. The total scores of complexity on reproducible approaches for developer and user work

3.3.1.2 Size of Reproducible Artifacts

Figure 3.10 shows the comparison of sizes for the five local reproducible approaches. "*Approach-5: Containerizing All Software and Modeling Workflows with Sciunit*" is the most lightweight and it is ten times smaller than "*Approach-4: Containerizing All Software with Singularity*," which is the second most lightweight. The reason for this is that Sciunit only encapsulates dependencies when dependencies are used during modeling workflows, compared to other containerization tools that containerize additional software and Python libraries perhaps not directly used in the workflow. In addition, "*Approach-4: Containerizing All Software with Singularity*" is more lightweight than approaches 1-3 ("*Approach-1: Compiling the Core Model Software*", "*Approach 2: Containerizing the Core Model Software only with Docker*", and "*Approach 3: Containerizing All Software with Docker*") because Singularity utilizes a flatter structure, meaning every dependency is included in only one image. In contrast, Docker has a layer structure concept for multiple images; therefore, dependencies in each image can separately be used. This concept used in Docker is not helpful for a single model software run, but it is important when researchers want to use multiple commands with layered images such as web development and operation. Finally, container tools have a compression function, so "*Approach 3: Containerizing All Software with Docker*" is more lightweight than "*Approach-1: Compiling the Core Model Software*" and "*Approach 2: Containerizing the Core Model Software only with Docker*."

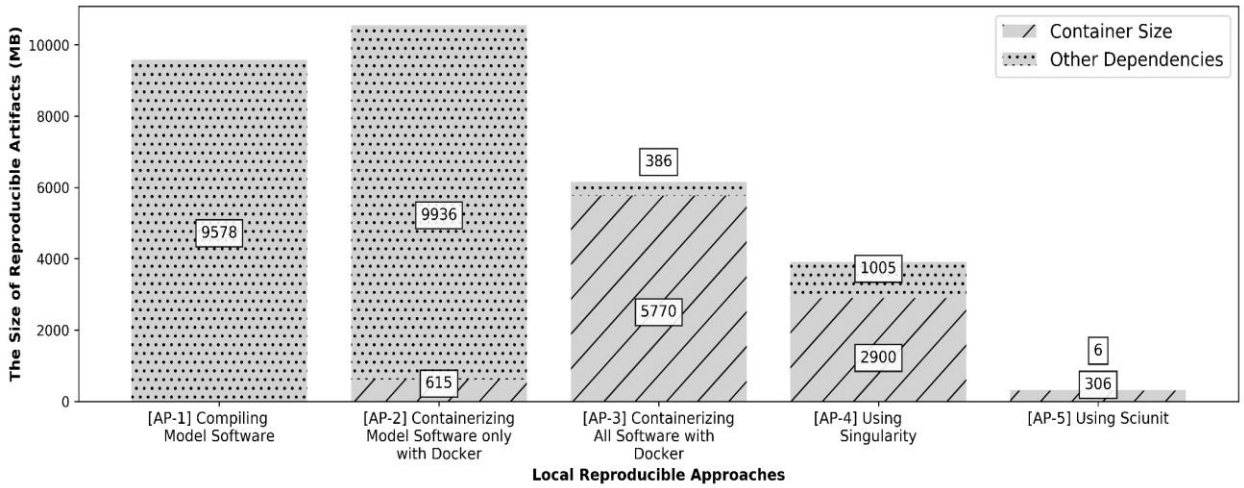

Figure 3.10. Comparison of the size for reproducible artifacts in five local reproducible approaches

3.3.1.3 Computational Time

Figure 3.11 shows the results of computational time for the five approaches using a local computer. It shows that "*Approach 2: Containerizing the Core Model Software only with Docker*" and "*Approach-5: Containerizing All Software and Modeling Workflows with Sciunit"* are slightly slower than the other approaches. However, overall computational time is similar across the five local reproducible approaches. For the remote approaches (Figure 3.12), "*Approach 11: Using a HPC Cluster*" was the fastest approach, followed by "*Approach-7: Using CJW.*" Because CJW and XSEDE are on separated machines connected by the Internet, unlike "*Approach 11: Using a HPC Cluster*", "*Approach-7: Using CJW*" requires additional time to submit jobs between CJW to XSEDE and retrieve model output from XSEDE to CJW. Although there are variations according to the status of memory use, the rest of the remote reproducible approaches are similar to the local reproducible approaches. Also, Sciunit can currently use only one core. Because Dask (Rocklin, 2015) automatically allocates multiple cores for ensemble simulations, Sciunit cannot encapsulate ensemble simulations (Scenario 2 and 4). Therefore, when Sciunit reproduced modeling workflows, it could not find cores that were used when a Sciunit container was created. Hence, we could not measure its computational time for scenarios 2 and 4. From the performance test of computational time, for data intensive modeling such as the simulation of fully distributed models and Contiguous United States (CONUS) scale models, we can see that it is invaluable to use remote environments, especially HPC clusters.

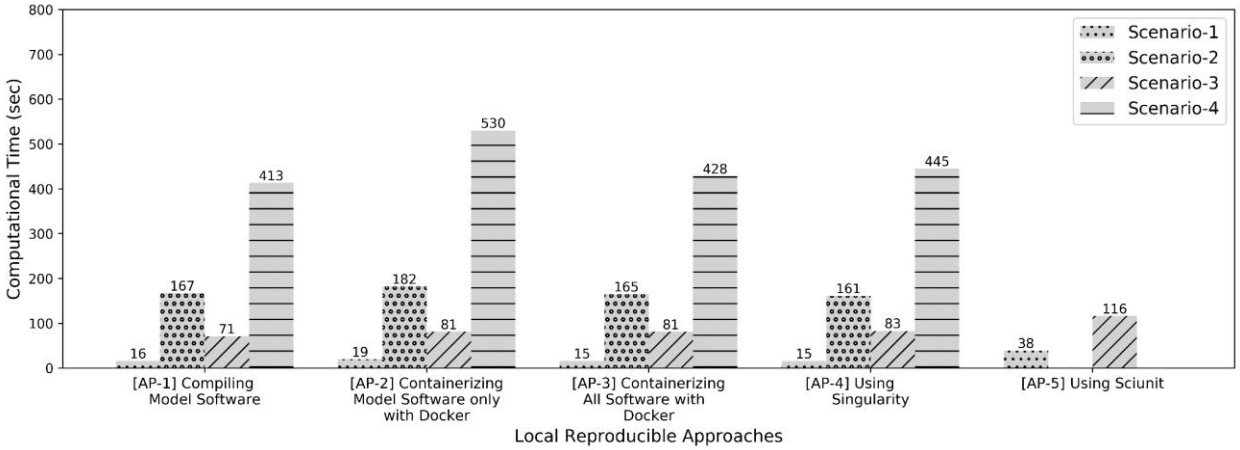

Figure 3.11. Comparison of computational time in five local reproducible approaches

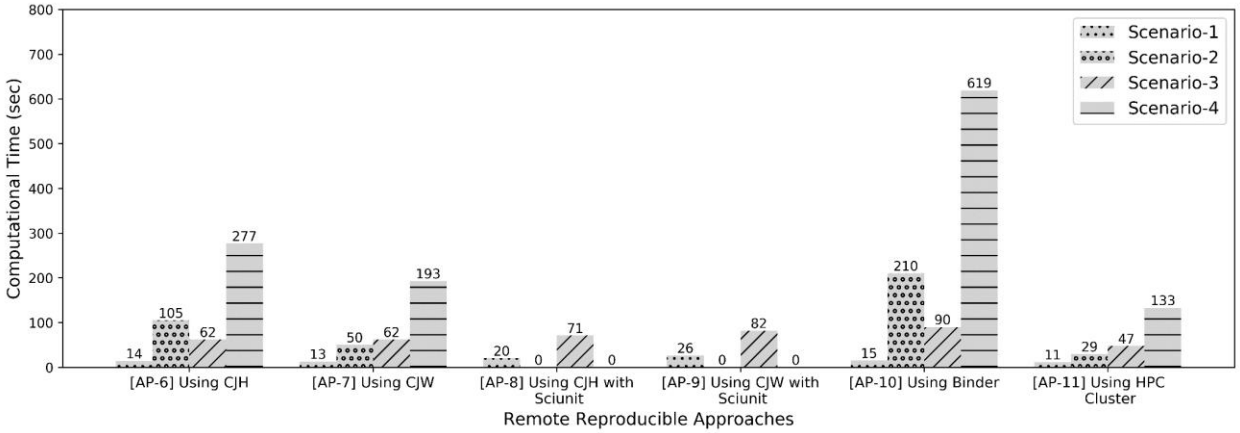

Figure 3.12. Comparison of computational time in six remote reproducible approaches

3.3.2 Qualitative Performance

Table 3.6 presents results from the qualitative performance tests for "*Approach 1: Compiling the Core Model Software*." As the GNU Make tool is a traditional method to reproduce model software, GNU Make itself is important because, even though we use containerization tools, this tool has to be used across the 11 reproducible approaches by the developer to build the SUMMA executable. However, this approach is still difficult for model users. Therefore, programming experts or model developers should use GNU Make to efficiently review and apply their new and modified source code in model software. Consequently, we recommend this approach to researchers who want to participate in model software development and management.

Table 3.6. Qualitative evaluation and recommended uses for "Approach-1: Compiling the Core Model Software"

Scenario	Descriptions [D: Developer, U: User]
Strengths	☐ [D, U] GNU Make itself is important because this tool has to use in 11 reproducible approaches
	☐ [D] Efficient for model software developers to review and apply their new and modified source code
Weaknesses	☐ [U] Difficult to apply Makefile configuration setting for compiling model software
Recommended usages	☐ [Research] model software development and management

Table 3.7 presents the results of the qualitative performance evaluation for "*Approach 2: Containerizing the Core Model Software only with Docker*." This approach uses Docker to containerize only the core model software; therefore, users can easily reproduce SUMMA using Docker from DockerHub. In addition, because users can install and apply new Python libraries as model APIs without any limitations, this approach has become popular. After downloading the SUMMA Docker image and installing pySUMMA within a Conda VE, users can execute SUMMA using the "*docker*" option in pySUMMA "*run*" method. Even if users have not downloaded the SUMMA Docker image in local computer, pySUMMA can automatically download it from DockerHub. However, sometimes when users create the Conda VE, unexpected errors may occur causing the user to have to create the Conda VE manually. Therefore, we recommend this approach for model applications where the user requires flexibility in what Python packages and other libraries are needed to complete the application.

Table 3.7. Qualitative evaluation and recommended uses for "Approach 2: Containerizing the Core Model Software only with Docker"

Scenario	Descriptions [D: Developer, U: User]
Strengths	☐ [U] Easy to download and use Docker images for model software via DockerHub
	☐ [U] Efficient to install new Python packages or other libraries for various application research
Weaknesses	☐ [U] Unexpected errors may occur when users create Conda VE manually
Recommended usages	☐ [Research] Model application with flexible application of various Python packages and other libraries

Table 3.8 presents the results for the qualitative performance evaluation for "*Approach 3: Containerizing All Software with Docker*." This approach containerizes every dependency into a Docker image; therefore, the procedure is stable and consistent in that it is very unlikely that errors will occur across users. However, there is a limitation for installing new software or dependencies. Because users have to work inside a Docker image, even if users can install new dependencies, they are temporary. Therefore, this approach is helpful for the purpose of offline education for practicing and reproducing published results in local computers (public or personal computers), but is less well suited for use cases that require extension of past work.

Table 3.8. Qualitative evaluation and recommended uses for "Approach 3: Containerizing All Software with Docker"

Scenario	Descriptions [D: Developer, U: User]
Strengths	☐ [U] Easy to download and use Docker images for environmental modeling via DockerHub
	☐ [U] Possible to use all required model software and other software from a Docker image
	☐ [U] Stable steps to use environmental models
Weaknesses	☐ [U] Limitation to install new model software or other software
Recommended usages	☐ [Education] Offline education requiring stable and consistent reproducibility

Table 3.9 presents the qualitative performance evaluation results for "*Approach-5, 8 and 9: Using Sciunit.*" Sciunit has many advantages such as being the most simple and lightweight of the 11 reproducible approaches considered in this study. However, Sciunit sometimes struggles to encapsulate not only every dependency, but also all workflows for complicated applications. Due to the lightweight containers and easy installation of the Sciunit tool, this approach is helpful for education use cases where instructors can share educational reproducible computational materials and students are asked to containerize their own analyses. In addition, Sciunit is efficient in terms of memory use for encapsulating all modeling environments, workflows, and data into one container. Thus, it is also a powerful tool for reliable reproducible research without continuous version control.

Table 3.9. Qualitative evaluation and recommended uses for Approach-5, 8 and 9: Using Sciunit

Scenario	Descriptions [D: Developer, U: User]
Strengths	☐ [D, U] The simplest complexity for reproducibility in both developer and user perspective
	☐ [U] The most lightweight in 11 reproducible approaches
	☐ [D, U] Easy to share Sciunit containers as a file format
	☐ [D, U] Possible to use Sciunit on local and remote environments after installing it using pip install
Weaknesses	☐ [U] Niche usage comparing to Docker and Singularity, sometimes unstable to create containers
	☐ [U] Impossible to encapsulate automatic allocation of parallel computing such as Dask
Recommended usages	☐ [Education] Offline education
	☐ [Research] Reliable reproducibility because Sciunit can containerize all reproducible artifacts into a Container without significant memory use

Table 3.10 presents the qualitative performance evaluation results for "*Approach-6 and 7: Using CJH and CJW.*" This approach allows for users to use preconfigured modeling environments; therefore, users can use environmental models in a straightforward way without any additional software installation. In addition, CJW supports HPC (XSEDE) use for parallel computing. Also, CJH supports a custom Conda VE to install Python or other libraries permanently, like the "*Approach 2: Containerizing the Core Model Software only with Docker.*" However, there is a limitation to install a new model software into CJH and CJW by users because both are developed by Docker, so there is a security issue when users install the new software. Therefore, it takes time to deploy new software into CJH and CJW because the CJH and CJW

development teams need a certain amount of time to review and deploy the new software on CJH and CJW. Consequently, we recommend this approach for online education and data-intensive computing research (CJW).

Table 3.10. Qualitative evaluation and recommended uses for "Approach-6 and 7: Using CJH and CJW"

Scenario	Descriptions [D: Developer, U: User]
Strengths	☐ [U] The simplest complexity for users, possible to use preconfigured modeling environments
	☐ [U] Possible to use HPC (XSEDE) for parallel computing (CyberGIS-Jupyter for water)
	☐ [U] Possible to install custom Conda VE (CUAHSI JupyterHub)
Weaknesses	☐ [U] Impossible to install particular model software or packages that uses 'sudo' command
	☐ [D] Requires a certain amount of time to review and deploy a new software by CJH and CJW development team
Recommended usages	☐ [Education] Online education (CJH and CJW)
	☐ [Research] Data-intensive computing (CJW)

Table 3.11 presents the qualitative performance test results for "*Approach 10: Using Binder*." This approach allows developers to share modeling environments online with users through a single click. Also, users can add new software or libraries, though users need to understand how to edit Binder configuration files to do so. Despite these conveniences, MyBinder has a limitation in persistent sessions because it supports these online modeling environments for free. Therefore, if there is no activity by users for 10 minutes, the Jupyter modeling environment is automatically shut down without saving into a persistent data storage. Therefore, we recommend this approach for online education use cases, but not for more sophisticated research applications unless Binder can be implemented with persistent data storage. That said, this approach is useful as a preliminary auditing procedure for research applications to deploy new software or libraries into Docker-based virtual research environments (Prasad et al., 2020) such as CJH and CJW because both Binder and these cyberinfrastructure are developed by Docker.

Table 3.11. Qualitative evaluation and recommended uses for "Approach 10: Using Binder"

Scenario	Descriptions [D: Developer, U: User]
Strengths	☐ [U] Easy to share modeling environments online
Weaknesses	☐ [U] Non-persistent sessions (automatically shut down if there is no activity for 10 min)
Recommended usages	☐ [Education] Online education

Table 3.12 presents the qualitative performance evaluation results for "*Approach 11: Using a HPC Cluster*." Comparing to "*Approach-6 and 7: Using CJH and CJW*", "*Approach 11: Using a HPC Cluster*" can add new model software or libraries by users without security issues. Therefore, if researchers want to add new software by themselves and use HPC clusters for parallel computing, "*Approach 11"* is the most appropriate approach. Therefore, we recommend this approach for data intensive computing research. Currently, Singularity is less popular than Docker, so sometimes researchers need to create Singularity definition files by themselves. In this scenario,

we recommend researchers try to find a Dockerfile first and then use the docker2singularity library to automatically convert the Dockerfile into a Singularity definition file.

Table 3.12. Qualitative evaluation and recommended use for "Approach 11: Using a HPC Cluster"

Scenario	Descriptions [D: Developer, U: User]
Strengths	☐ [D] Possible to add new software or libraries in HPC without other's help
	☐ [D] Easy to convert Docker images to Singularity images using docker2singularity library
	☐ [U] The fastest computational time (Possible to use multiple cores for parallel computing in HPC)
	☐ [U] Lightweight than other reproducible approaches except Sciunit
Weaknesses	☐ [U] Niche usage comparing to Docker
	☐ [U] Requires preinstalled JupyterHub environment in HPC for a user-friendly interface
Recommended usages	☐ [Research] Data and compute intensive modeling

3.4 Discussion

3.4.1 Guidance and Recommended Uses

Across the quantitative and qualitative results presented in this study, it is possible to draw out best practices for leveraging containerization and computing environments to achieve reproducible environmental modeling objectives. We can classify various environmental modeling objectives into two broad categories: 1) education and 2) research. Traditionally, we practice environmental modeling through classes and workshops in an "offline" manner that requires installing software on local computers. However, recently many educational institutions are transitioning to remote or "online" compute environments (Prasad et al., 2020). Therefore, we divide the objectives of education into 1) online and 2) offline. For environmental modeling research, we can generally divide the objectives of research into 1) model installation as developers, 2) model application as users, and 3) data-intensive computing for complex modeling using large datasets (Addor et al., 2020).

For education purposes, especially online education, "*Approach-6 and 7: Using CJH and CJW"* and *"Approach 10: Using Binder"* are the best approaches because they offer the lowest complexity for users (complexity score of users: 2, Figure 3.9). In addition, these remote reproducible approaches offer more flexibly than HPC because, in general, HPC requires more rigid account permissions (like YubiKey for NCAR HPC) than CJH and CJW. Moreover, these environments support easy sharing via HydroShare and preconfigured modeling environments. Sciunit also has the lowest complexity; however, because Sciunit needs to encapsulate dependencies and workflows together, sometimes creating Sciunit containers can be more unstable than other approaches. Next, for offline education, "*Approach 3: Containerizing All Software with Docker*" and "*Approach-5: Containerizing All Software and Modeling Workflows with Sciunit*" are the best approaches because of the first and second lowest complexity scores for users (2 for *Approach-5* and 3 for *Approach 3*). Among these reproducible approaches, if users want a more reliable approach, "*Approach 3: Containerizing All Software with Docker*" is better because Docker containerizes every dependency into Docker images. In addition, if users want a more lightweight approach (Figure 3.10) to distribute containerized images and a reliable approach

without considering version control for offline education, "*Approach-5: Containerizing All Software and Modeling Workflows with Sciunit*" is better.

For the purpose of model development in research, "*Approach-1: Compiling the Core Model Software*" is the only approach that can build new or modified model software source code efficiently. Other approaches can only create a container image using existing model software source code for reproducibility. For the purpose of model application in research, "*Approach 2: Containerizing the Core Model Software only with Docker*" and "*Approach-6: Using CJH" are* the best because these approaches have the flexibility to install and apply new Python libraries for various analyses and visualizations. For the research purpose of data-intensive computing, "*Approach- 7: Using CJW*" and "*Approach 11: Using a HPC Cluster*" are the best approaches because both remote approaches are able to use multiple cores for parallel computing; therefore, these two approaches have the first and second fastest computational time (Figure 3.12).

Table 3.13. Best practices for reproducible approaches on local and remote environments to achieve environmental modeling objectives

Objectives		Best Practices
(a) Education	(1) Online (Class or Workshop)	☐ CUAHSI JupyterHub and CyberGIS-Jupyter for water (AP-6 and 7) and Binder (AP-10) → The lowest complexity for users (score:2), a flexible approach, and easy sharing
	(2) Offline (Class or Workshop)	☐ Containerizing Model Software and Other Software (AP-3) and Sciunit (AP-5) → The first (AP-5, score:2) and second (AP-3, score:3) lowest complexity for users, a more stable approach (AP-3), and the most lightweight artifacts (AP-5)
(b) Research	(3) Model Development	☐ Compiling Model Software (AP-1) → The only approach to build new or modified model software source code
	(4) Model Application	☐ Containerizing Model Software (AP-2) and CUAHSI JupyterHub (AP-6) → Lower complexity than others (score:3), flexibility to install and apply new Python libraries for various analysis and visualization
	(5) Data-Intensive Computing	☐ CyberGIS-Jupyter for water (AP-7) and HPC Cluster (AP-11) → The first and second fastest computational time, possible to use multiple cores for parallel computing

3.4.2 Limitations of Current Sciunit Software

"*Approach-5, 8 and 9: Using Sciunit"* is the simplest and most lightweight reproducible approach. However, because Sciunit is still in active development, there are limitations to reproduce modeling environments and workflows together in Sciunit. As Sciunit containerizes both modeling environments and workflows, Sciunit has to interact with the same workflow that was applied to create the Sciunit container. Other approaches such as Docker and Singularity separate the computational modeling environments and workflows, so they are more flexible than Sciunit and can apply different workflows based on containerized computational environments. Therefore, Sciunit is developing a functionality to convert a Sciunit container to a Docker image for more flexible workflow applications. In addition, Sciunit only containerizes dependencies that were used in workflows; therefore, users cannot employ functions that were not originally part of workflows that created the Sciunit container even though these functions exist in model software or Python-based model APIs. Therefore, Sciunit is in active development to develop a way to

import new libraries into Sciunit containers. Moreover, as Sciunit encapsulates modeling environments and workflows simultaneously, Sciunit can cause unpredictable errors during tracking and self-containerizing. Therefore, Sciunit needs more experiments and evaluations in various modeling environments and workflows.

3.4.3 Limitations of Currently Available Virtual Environments for Environmental Modeling

"*Approach-6 and 7: Using CJH and CJW*" use a Jupyter interface and have been widely used because of easy access and preconfigured modeling environments (Prasad et al., 2020). However, these virtual research environments still have limitations for users installing new software. Therefore, to foster remote environmental modeling, we need more compatible computational modeling environments to allow users to install new software on virtual research environments. The "udocker" tool, which is a tool for using Docker without privileges (Gomes et al., 2018), would allow users to add new model software to a Docker image. In addition, we need official and standard procedures for adding new software on CJH and CJW because the CJH and CJW development team cannot control every deployment of new software. Using compatible capabilities to install new software, users could verify modeling environments in their own user sessions. Then, once tested, they could request their successful modeling environments be made public for other researchers to use on the CJH and CJW.

"*Approach 10: Using Binder*" is also a powerful remote modeling environment. But there are limitations such as if users have no activity for 10 min, the MyBinder user session is automatically shut down. This is because creating MyBinder sessions on BinderHub is open to anybody, anywhere, and anytime for free. Therefore, some time limits for BinderHub user session resources are inevitable. As a short term solution, the current Binder supports an automatic save function when the user session is shut down if users are setting the local directory to save files in the user session. A potential solution would be creating online JupyterHub environments with a Binder-ready repository such as GESIS Notebook (https://notebooks.gesis.org).

3.5 Conclusions

Reproducibility is the cornerstone of science because it allows for the accumulation of knowledge by building on prior work (Pauliuk, 2020). While recent research has highlighted the difficulties in achieving reproducible computational work (Monya Baker, 2016) such as sharing modeling environments and automated workflows, gaps remain in understanding how to effectively use modern software tools and practices to achieve more reproducible computational analyses (Kim et al., 2018). To this aim, we explored 11 approaches for achieving reproducible modeling goals using a combination of different containerization tools and virtual environments. We assessed the approaches using both quantitative (complexity, size of reproducible environments, and computational time) and qualitative (strengths, weaknesses, and recommended usages) measures in order to offer perspectives on the best practices for different use cases commonly in the environmental modeling community. We used SUMMA, pySUMMA, and Jupyter notebooks to represent a common environmental modeling use case to assess the 11 methods.

From this study, we showed how no single approach is the best for all reproducibility use cases. We need to understand the specific modeling reproducibility objectives and find the best approach for those objectives. For educational use cases, we considered low complexity for users as the most important factor. Thus, the best methods for online education are CUAHSI JupyterHub, CyberGIS Jupyter for Water, and Binder (Approaches 6, 7, and 10) and the best method for offline education is Sciunit (Approach 5). For research use cases, we considered three possible objectives: model development, model application, and data-intensive modeling. For model development, it is important to be able to recompile after editing and updating model source code; therefore, the best method for model development is compiling the core model software (Approach 1). For model application, flexibility is important to deploy new software and to apply new methods; therefore, the best methods for model application are containerizing the core model software only with Docker and using an online JuypyterHub environment like CUAHSI JupyterHub (Approaches 2 and 6) for deploying the software. For data-intensive computing, it is important to be able to use multiple cores to improve computational time; therefore, the best methods for data-intensive computing are using Singularity for containerization with either the CyberGIS Jupyter for Water environment, which interfaces with XSEDE, or a HPC cluster with a Jupyter instance as the computational environment (Approaches 7 and 11).

Future research to further advance reliable and efficient reproducible approaches for environmental modeling should improve weaknesses in reproducible approaches we identified in this study. For education purposes, the trend of environmental modeling is moving to remote or online computational environments, so we need to focus on deployment flexibility for virtual research environments and persistent sessions and storage for solutions like Binder. While Sciunit has the strongest capabilities compared to other approaches for environmental modeling, it can benefit from adding flexibility to apply different modeling workflows and new software dependencies. This continued research and virtual environment enhancement will improve the software ecosystem needed to make computational research more reproducible, open, transparent, ultimately fostering a “culture of reproducibility” (Rosenberg et al., 2020) within environmental modeling.

Data and Software Availability

All data and computational environments used in this study are ten HydroShare resources and three GitHub repositories. We published all data and computational environments with persistent digital object identifiers (DOI’s) on HydroShare and shared them by a collection resource in HydroShare (Choi Y. J., 2021). This collection resource provides the links for all HydroShare resources as “Collection Contents” and three GitHub repositories as “Related Resource Reference.” Ten HydroShare resources consist of one collection resource, two model instance resources for SUMMA model input, one model program resource for Singularity image, one composite resource for Virtual Box image of five local approaches, four composite resources for Jupyter notebooks of four remote approaches (AP-6, 7, 8, and 9), and one composite resource for a Jupyter notebook to create Figure 3.9-3.12 using performance results. Three GitHub repositories created to share Jupyter notebooks and configuration files for AP-3, AP-4, AP-10, and AP-11.

List of Relevant URLs

Binder: https://mybinder.org

Binder Configuration: https://mybinder.readthedocs.io/en/latest/using/config_files.html

CSDMS: https://csdms.colorado.edu/wiki/Hydrological_Models

CUAHSI JupyterHub: https://jupyterhub.cuahsi.org

Docker recipes of CUAHSI JupyterHub: https://github.com/CUAHSI/cuahsi-stacks

CyberGISX: https://cybergisxhub.cigi.illinois.edu

CyberGIS-Jupyter for water: http://go.illinois.edu/cybergis-jupyter-water

Docker recipes of CyberGIS-Jupyter for water: https://github.com/cybergis/Jupyter-xsede/tree/master/singularity_def

docker2singularity: https://github.com/singularityhub/docker2singularity

Figshare: https://figshare.com

GESIS Notebook: https://notebooks.gesis.org/binder

GitHub: https://github.com

Google Colab: https://colab.research.google.com

GNU compilers (gfortran): https://gcc.gnu.org/fortran

GNU compilers (GCC): https://gcc.gnu.org

GNU builders (Make): https://www. gnu.org/software/make

HydroShare: https://www.hydroshare.org

Jupyter notebooks for pySUMMA tutorial: https://github.com/arbennett/pysumma-tutorial

Microsoft Azure: https://note books.azure.com

NCAR, National Center for Atmospheric Research, HPC: https://jupyterhub.ucar.edu

Pip: https://pip.pypa.io

pySUMMA: https://github.com/UW-Hydro/pysumma

Python: https://www.python.org

R: https://www.r-project.org

Rivanna, HPC at University of Virginia HPC: https://www.rc.virginia.edu

SUMMA GitHub: https://github.com/NCAR/summa

SUMMA DockerHub: https://hub.docker.com/r/uwhydro/summa

Virtual Box: https://www.virtualbox.org

Virtualenv: https://virtualenv.pypa.io

XSEDE, an HPC resource on the Extreme Science and Engineering Discovery Environment, https://www.xsede.org

Zenodo: https://zenodo.org

Chapter 4

Toward Seamless Environmental Modeling: Integration of HydroShare with Server-side Methods for Exposing Large Datasets to Models

4.1 Introduction

Reproducibility is a fundamental requirement to accumulate knowledge and advance science (Monya; Baker, 2016; National Academies of Sciences, 2019; Stagge et al., 2019; Wilkinson et al., 2016). In Nature's survey of reproducibility, however, about 70% of researchers had failed to reproduce another researcher's results and 50% of researchers failed to reproduce their own research results (Monya; Baker, 2016). To improve reproducibility, data management and stewardship are the basic elements (Wilkinson et al., 2016). However, a data science survey reported that data scientists spent 19% of their time collecting data (finding and accessing) and 60% of their time cleaning and organizing data (CrowdFlower, 2016). That leaves 21% of their time for core analysis. To overcome these problems, the FAIR principles have been presented as high level guidelines to improve scientific data management and access by making them Findable, Accessible, Interoperable, and Reusable (Wilkinson et al., 2016). The FAIR principles emphasize the necessities of both human and machine applicable data management environments. Ongoing efforts on FAIR guiding principles have advanced data repositories with identifier mechanisms, data management plans, policies and standards (Hodson et al., 2018). Based on the use of unique identifiers, such as the Digital Object Identifier (DOI) or other persistent identifiers, data can become "Findable." Public machine-accessible interfaces allow datasets and metadata to become "Accessible," and the use of standard terms, metadata, and wide range of datatypes allows data to become "Interoperable." Finally, detailed documents together with metadata can allow data to become "Reusable."

Recently, numerous online repositories have accepted FAIR principles and enhanced their functionalities to be more Findable, Accessible, Interoperable, and Reusable (Crosas, 2020; Wilkinson et al., 2017). For example, Dataverse (https://dataverse.org) provides the functionalities to create Digital Object Identifiers (DOI), share metadata and data files, and access data with public licenses on their data landing pages. Similar to Dataverse, FigShare (https://figshare.com), Mendeley (https://mendeley.com), and Zenodo (https://zenodo.org/) are other online repository examples that support these capabilities to follow FAIR principles. However, these online repositories have been developed for general purposes and have focused on data publication using metadata at the file level. This means the data are preconfigured for particular purposes, such as understanding published research and practicing data analysis in workshops. Therefore, there are limitations regarding reusability for multiple applications across different case studies and interoperability for programmatic access to multiple data collections using complementary tools.

In the hydrologic science community, the types of data and models, and thus the data and file sharing needs of the modeling community, are diverse (Horsburgh et al., 2016). HydroShare (https://www.hydroshare.org) helps to meet these needs by providing an online repository to support sharing these multiple types of data and models (Morsy et al., 2017). Data types include time series, geographic features (Shapefile), geographic rasters (GeoTIFF), and multidimensional space-time datasets (NetCDF). In addition, HydroShare supports model sharing using model programming (source code or compiled software with related metadata such as version, programming language, and release date) and model instance resources (model input and output with related metadata such as application methods and a relationship to a model program resource). After creating a HydroShare resource, users can share it using a unique URL (public sharing status) or DOI (published sharing status). Moreover, HydroShare supports RDF (Resource Description Framework) HydroShare Python Client, hsclient (https://github.com/hydroshare/hsclient) to interact with HydroShare resources and JupyterHub computational environments (CUAHSI JupyterHub and CyberGIS-Jupyter for Water) for various analyses such as modeling and big data analysis using Jupyter notebooks (Choi et al., 2021). Therefore, we can say HydroShare supports FAIR principles, using unique identifiers (Findable), metadata of multiple resource types (Accessible), hsclient (Interoperable), and JupyterHub (Reusable). HydroShare is mainly utilized for spatial data publication at the file level and is not commonly used to distributed large, national-scale datasets. However, HydroShare does serve as a shortcut to the CUAHSI HIS HydroClient (https://data.cuahsi.org), an external web application to support national-scale time series data distribution. However, there is no similar support for national-scale spatial data distribution within HydroShare.

For national-scale spatial data sharing, there are a number of government-sponsored organizations and research centers with open-web data distribution systems. For example, USDA (United States Department of Agriculture) NRCS (National Resources Conservation Service) provides over 100 high resolution raster and vector data such as Census, Digital Elevation Model (DEM), Hydrography, Land Cover, Soil, and Transportation in the Geospatial Data Gateway (https://datagateway.nrcs.usda.gov). USGS 3D Elevation Program (3DEP, https://www.usgs.gov/core-science-systems/ngp/3dep) provides various elevation maps such as DEMs and Lidar point clouds. MRLC (Multi-Resolution Land Characteristics Consortium, https://www.mrlc.gov) currently provides land cover, tree canopy, urban imperviousness and other related data from 2001 to 2016. In addition, some open-web distributed systems support application programming interfaces (APIs) to programmatically interoperate between users and open-web distributed system. However, the use of spatial data APIs is difficult and has a steep learning curve. Therefore, usually researchers collect spatial data manually, meaning downloaded needed data from these open-web distributed systems, for their modeling needs.

Service-Oriented Architecture (SOA) and open web-based data sharing technologies have been put forth as more convenient data access approaches, as well as approaches for integration of certain environmental models (Chen et al., 2020). However, these advanced approaches care difficult to design, build, and sustain, especially for the complex and heterogeneous data required in environmental modeling. Miles and Band (2015) put further one solution to the problem: the Ecohydrolib Python library for managing spatial data acquisition and preparation workflows for

ecohydrology modeling. The idea was to access data from data providers and to use data processing software to map these data into specific environmental model needs. HydroTerre (L. N. Leonard, 2015) offered a different approach to solving the problem by created a database of essential terrestrial variables built from multiple national-scale spatial datasets and processed to create input to the PIHM (The Penn State Integrated Hydrologic Modeling System) (M. Kumar et al., 2010) model. The EcoLib approach relied on having consistent and reliable APIs from data providers, which can change and create vulnerabilities within the system. The HydroTerre approach removed the reliance on data providers by creating copies of the data, but doing so for the national creates a major data management and storage problem due to the scale of the data required.

There are a growing number of scientific datasets that are national and international in scope, and that could benefit from ways to easily share them online in a machine readable way. Large sample hydrology studies (Addor et al., 2020) have become popular to cover large areas with consistent and robust high-quality datasets to "balance depth with breadth" (Gupta et al., 2014). For example, Model Parameter Estimation Experiment project (MOPEX) provided hydrometeorological observation and attribute data for 438 catchments across the USA (Q. Duan et al., 2006). Another example is the European catchments of Hydrological Predictions for the Environment model (E-HYPE), providing streamflow data and catchment attributes in 35,215 catchments and 1,366 river gauges in Europe (Kuentz et al., 2017). In an recent effort, Catchment Attributes and Meteorology for Large sample Studies (CAMELS, 671 catchments in the USA) (Addor et al., 2017; Newman et al., 2015) and CAMELS-Chile (516 catchments in the Chile) (Alvarez-Garreton et al., 2018) were created to provide climate data and catchment attribute data. Computational platforms are being developed to support big data in geosciences. For example, PANGEO (https://pangeo.io) supports a community platform with big data in climate, hydrologic, and ocean field (Hamman et al., 2018). However, outside of such systems, MOPEX, E-HYPE, CAMELS, and CAMELS-Chile data need to be downloaded manually and processed to be used in particular environmental models.

Recent research has made strides to overcome the limitations of spatial data sharing. In HydroShare, for example, two server-side methods are used to distribute spatial data in a machine-readable form: GeoServer (http://geoserver.org) (Crawley et al., 2017) and Thematic Real-time Environmental Distributed Data Services (THREDDS) Data Server (TDS) (Gan et al., 2020). GeoServer supports data access, display, and processing of geographic raster and feature data using the Open Geospatial Consortium (OGC) web service (OWS) (Wenjue et al., 2004). TDS is an advanced client/server software that provides remote access to data and metadata stored in various geo-temporal datasets. Using these services, HydroShare users can easily share, access, retrieve, and subset geographical data via GeoServer and various types of scientific data such as NetCDF via TDS. Automatic metadata harvesting and data transfer functionalities in HydroShare enable user uploaded NetCDF, geographic rasters, and feature data to be available through its connected GeoServer and TDS instances. This allows functionalities provided by GeoServer and TDS to be leveraged to not only visualize but also analyze spatial data stored in HydroShare. Despite the availability of these capabilities of GeoServer and TDS, both are being underutlized in HydroShare. For example, GeoServer is mainly being used for visualizing geographic raster and feature data online. TDS is mainly being used for sharing and visualizing grid-based

multidimensional climate data (Gan et al., 2020). The goal of this research is, therefore, to explore how these services can be used to support more complex use cases required in environmental modeling.

With this goal in mind, we aim to answer the following research questions. 1) Can the GeoServer and TDS implementations with HydroShare be used to enable more seamless environmental modeling? 2) Can HydroShare along with GeoServer and TDS provide a more sustainable and scalable solution for sharing machine-readable large-extent spatial datasets? The remainder of the paper is organized as follows. In the methodology section, we first present the procedures for creating large-extent spatial (LES) datasets and sharing them on GeoServer and TDS in HydroShare. Second, we describe how to subset LES datasets from GeoServer and TDS. Third, we present application workflows of these datasets for seamless environmental modeling using the Regional Hydro-Ecologic Simulation System (RHESSys) (Tague & Band, 2004a) as an example modeling system. In the results section, we present examples for three different watersheds: 1) Coweeta Subbasin18 in North Carolina, 2) Scotts Level Branch in Maryland, and 3) Spout Run in Virginia. In the discussion section, we review the advantages and limitations of using LES datasets on GeoServer and TDS. Finally, we conclude with a summary of the contributions of this research and suggest pathways for future research to further advance spatial data analysis in end-to-end environmental modeling.

4.2 Background

4.2.1 GeoServer

GeoServer is a Java-based open-source software that has been developed for publishing and visualizing spatial data online. Open Geospatial Consortium (OGC), a non-profit organization, has released standards for sharing spatial data online including the Web Map Service (WMS), Web Feature Service (WFS), and Web Coverage Service (WCS). WMS provides geo-registered spatial images either as a jpeg or png using a simple HTTP interface. WFS provides the direct interoperability to discover, retrieve, and subset feature geographic data rather than sharing geographic data at the file level. WCS is similar to WFS except that WCS provides direct interoperability to the raster geographic data. In addition, there are many client libraries that use OGC web service interface standards. In this study, we used OWSLib to visualize, retrieve, and subset spatial data using various formats such as through OGC web services, Shapefiles, ArcGRID, and GeoTIFF. In the past, GeoServer has been used within Hydroshare to support spatial data visualization. Initially it served as the spatial backend for the Hydroshare GIS Web App that was based on the Tethys framework (Crawley et al., 2017). This app has been deprecated in favor of an implementation that uses GeoServer natively to provide direct access to the data through built-in standardized OGC compliant web services. This feature allows sharing and visualizing public resources in HydroShare that contain spatial data. Every resource that becomes public and contains geographic raster or feature content is automatically registered with GeoServer using a customized middleware.

4.2.2 THREDDS Data Server (TDS)

The Thematic Real-time Environmental Distributed Data Services (THREDDS) Data Server (TDS) is open-source software distributed by the Unidata community program of the University Corporation for Atmospheric Research (UCAR). TDS provides web services that publish remote access to data and metadata stored in a variety of well-known geo-temporal dataset formats used for environmental research such as GRIB (Gridded Binary), HDF5 (Hierarchical Data Format version 5), and, most commonly, NetCDF (Network Common Data Form) as viewed through a Common Data Model (CDM) (Nativi et al., 2008). TDS presents gridded, point, and time series datasets organized into thematic catalogs and provides access to data through suites of web services such as TDS and OGC.

The Open-source Project for Network Data Access Protocols (OPeNDAP) specifies two suites of web service requests and responses, DAP2 and DAP4, for remotely accessing CDM datasets. Remote access of TDS-hosted datasets through DAP2 client software allows the advantage of placing dimensional constraints on the CDM variable arrays transported in a response. The DAP2 request contains the constraints and effectively creates a subset of the TDS-hosted dataset for transport. The requesting client, therefore, need not concern itself with the size of the TDS-hosted dataset, but only with the size of the data response.

As an additional advantage, DAP2 clients, such as Unidata's NetCDF libraries or higher-level software utilizing those libraries such as xarray, initially make requests for only the metadata contained in the CDM header via a DAP2 Data Descriptor Structure (DDS) request and do not further request data until the variable array data is instantiated in the client via a DAP2 Distributed Oceanographic Data Systems (DODS) request. This "lazy-loading" behavior allows remotely opening the entire dataset but only transports portions of the dataset as needed programmatically, thereby reducing network load, transmission time, and memory consumption.

In contrast, due to dimensional constraints in DAP2 requests and DAP2 client lazy-loading, tools which assemble and manage Network Common Data Form (NetCDF) datasets have more influence on the size of TDS-hosted datasets. When only subsets of the datasets require transport, practicalities concerning dataset construction and server-side dataset management become the principles determining how to best partition large collections of data. Often the tools and computer stations which remotely access CDM datasets through TDS are the same or similar tools and stations constructing and managing the datasets prior to hosting. When collection-wide views of data are desirable, TDS supports NetCDF Markup Language (NCML) facilities to aggregate many datasets into one virtual dataset.

CDM is a general model for dataset, dimensional, and variable attribute instantiation. Many TDS services depend on the recognition of dataset feature types (i.e., point, trajectory, station, profile, radial, grid, swath, etc.) for optimal operation. NetCDF, which configures metadata in compliance with Climate Forecast (CF) conventions, also enables both TDS and DAP2 clients to intelligently recognize feature types beyond what the general model would otherwise convey, particularly for geo-referencing and projection.

4.3 Methodology

Figure 4.1 presents the overall modeling workflow used as a demonstration case study in the study. The figure shows how datasets can be shared through HydroShare with different data management and distribution systems underneath. All files in HydroShare are stored in iRODS, which is a distributed data storage and management system that also allows for the transfer of large datasets (Yi et al., 2018). Public spatial datasets are also replicated into GeoServer and TDS in HydroShare. We describe how to retrieve spatial datasets from GeoServer and TDS for supporting seamless end-to-end environmental modeling. To do this, we present an example application showing how to use LES datasets within RHESSys on CyberGIS-Jupyter for Water platform, which is a well-tailored CyberGISX instance to support data-intensive and reproducible research in environmental modeling community. Further detail on these steps is provided in the following subsections.

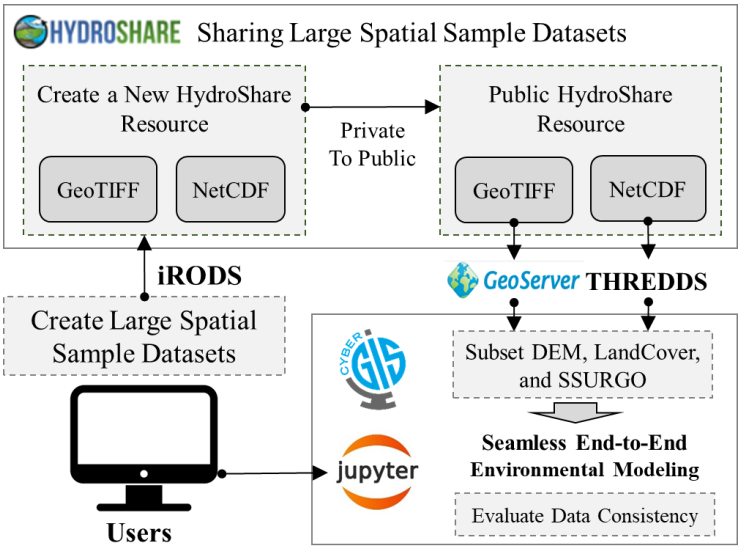

Figure 4.1. The workflows to create, share, subset, apply, and evaluate LES datasets for seamless environmental modeling workflows

4.3.1 Create and Share Large Spatial Sample Datasets

4.3.1.1 Collect Spatial Data

Environmental models often require spatial datasets including digital elevation models (DEMs), land cover, and soil maps to generate model inputs (DeVantier & Feldman, 1993). DEMs are used to delineate a watershed and extract spatial attributes such as flow direction, slope, aspect, and flow accumulation. The land cover is used to calculate surface roughness, evaporation, and transpiration according to the different land cover types such as urban, agricultural, and forest areas. The soil map is used to calculate water movement through soil including infiltration. Currently, there are many web-based spatial data distribution systems that provide spatial data from low to high resolution. They provide low resolution data services such as the 90 meter resolution Shuttle Radar Topography Mission (SRTM) (https://srtm.csi.cgiar.org) and the Google Earth Engine Datasets (https://earthengine.google.com) for various earth science data and analysis. National scale data is supported by federal government organizations such as USGS and USDA

as mentioned earlier. High resolution data services, most often hosted by particular research centers or state government organizations such as Chesapeake Conservancy Conservation Innovation Center, provide 1 meter resolution or higher DEM and land cover data.

Hydrologists most often use 10 or 30 meters resolutions of GeoTIFF data in environmental models considering the size of watersheds. For collecting DEM, we tested different data distribution interfaces and selected the Geospatial Data Gateway (GDG), which is operated by inter-government cooperation between the three service center agencies: Natural Resources Conservation (NRCS), Farm Service Agency (FSA), and Rural Development (RD). We selected GDG because it distributes data by various selection interfaces including by states, counties, bounding box, and custom area of interest. For collecting land cover, we emphasized data continuity for various applications such as land cover change. Therefore, we selected MRLC, which is a group of federal agencies, because this product provides consistent and reliable land cover information from 2001 to 2016 at the national scale. Finally, for collecting soil data, we selected GDG. There are three soil datasets in GDG: 1) the National Soil Geographic Database (NATSGO), which is a very general soil map of the entire U.S., 2) the State Soil Geographic Database (STATSGO), which is less detailed state-wide map, and 3) Soil Survey Geographic Database (SSURGO), which is the most detailed county level data. To support watershed modeling use cases, we obtained the most detailed soil data from SSURGO. In addition, we collected attributes data of SSURGO for environmental models. GeoTIFF datasets have Mukey (Map unit key which is the index to link different soil metadata table) values in each cell. Therefore, we selected five SSURGO attribute tables that RHESSys required to link with the Mukey in the GeoTIFF: 1) mapunit (mukey table), 2) chorizon (horizon table), 3) chtexgrp (horizon texture group table), 4) chtextur (horizon texture table), and 5) comp (component table). These tables are distributed at the county level though the Web Soil Survey web distributed system, which is linked by GDG. Therefore, we downloaded county-level SSURGO metadata for each state and merge them into a single SSURGO attribute table that can be joined to the GeoTIFF through the Mukey attribute.

After selecting a source for obtaining the spatial datasets, we next decided the best scale (National, State, or Local) for storing the datasets in HydroShare. In making this decision, we considered 1) the file size of spatial datasets, 2) the capabilities of GeoServer and TDS at handling different sized datasets, and 3) the reusability of applications across different watersheds. Ultimately, we decided that aggregating the spatial data at the state-scale would be best for the following reasons. First, we considered the feasibility of using national-scale spatial data within GeoServer and TDS because this would allow for truly seamless environmental modeling. The size of national-scale 30 meter resolution DEM, land cover dataset, and SSURGO dataset at the national scale are 44.6GB, 20GB, and 3.7GB, respectively. We were unable to find specific guidelines for the maximum size of datasets distributed using GeoServer and TDS. However, in our experience if a GeoTIFF is over 5~10 GB, it is difficult to manipulate them on most personal computers, therefore while a national-scale dataset may be feasible, it would be difficult to work with and maintain. Next, we considered state-scale data aggregations. In this case and using Virginia as an example, the DEM is 951MB, land cover is 342MB, and SSURGO is 157MB. At this file size, the process of uploading and subsetting the data on GeoServer and TDS went

smoothly. If datasets are smaller than state scale, it will be hard to support seamless modeling because many watersheds cross county boundaries. Therefore, in this study, we chose state scale as the optimal aggregation for distributing LES datasets. Finally, based on this decision, we obtained 10 or 30 m DEM and 30 m SSURGO spatial data from GDG. Also, we collected 30 m land cover spatial data in 2001, 2003, 2006, 2008, 2011, 2013, and 2016 from MRLC to create state scale LES datasets (Figure 4.2). The following subsections describe how these data were processed to have a consistent spatial reference system and then uploaded and shared through HydroShare.

Figure 4.2. The selection of data distribution systems and spatial data to create LES datasets

4.3.1.2 Create Consistent State-Scale Spatial Datasets

In this step, we considered the following factors to create consistent state-scale spatial datasets for use in GeoServer and TDS: 1) using data types optimal for GeoServer and TDS, 2) adopting a consistent coordinate system across all spatial data, 3) applying georeferencing transformations to adjust shifted locations in the merged state-scale DEM, and 4) having complete and meaningful metadata compatible with HydroShare for each datasets.

We adopted the GeoTIFF as the spatial data type for LES datasets. GeoTIFF is an OGC implementation standard for raster data that is commonly used to store grid-based spatial data with geographic metadata that describes the spatial location including spatial extent, coordinate reference system, and resolution. Most data providers serve DEM and land cover data as GeoTIFF, which is also one of the main data types supported by GeoServer. For SSURGO data, the information is typically served at county-scale using a shapefile format with soil attribute metadata. We explored merging the county-scale SSURGO shapefiles into a state-scale shapefile and serving the data though GeoServer. However, doing so resulted in a large dataset that is difficult to manage and feed into environmental models. Fortunately, since Feb 2021, USDA NRCS National Soil Survey Center has started to service National scale SSURGO data as a 30 meter resolution GeoTIFF. In addition, GeoTIFF can store a collection of 2D arrays, so GeoTIFFs can be easily transferred into a NetCDF multidimensional array as well. Given that TDS supports the NetCDF format, we decided to use GeoTIFF in GeoServer and NetCDF in TDS as the data types of distributing replicated copies of the DEM, land cover, and soils spatial datasets at a state-scale.

Adopting a consistent coordinate system is important to create a unified LES dataset to support environmental modeling. Each dataset was distributed in a different spatial coordinate

system. We adopted the UTM geographic coordinate system for consistency at the state-scale. Resampling is required when transforming coordinate systems. For resampling of the LES datasets, we used a bilinear interpolation resampling method for the DEM because it is continuous data and used a nearest neighbor resampling method in land cover and soils data because they are categorical data.

When multiple DEMs are merged into one DEM, there is often overlapping areas at the edges of the each original DEMs. These areas made the merged DEM (GeoServer and TDS LES DEM) shift about 0.3-1.0m compared to the original raw DEM. If users delineate a watershed using the merged LES datasets without recognizing these changes, they will get a different watershed compared to using the original data products. Therefore, we needed to apply a georeferencing tool in ArcGIS to shift linearly the merged DEM to the original location using control points.

Figure 4.3 shows the complete workflow using these three steps to create the state-scale LES datasets in both GeoTIFF and NetCDF formats. First, the data is projected using the appropriate UTM Zone coordinate system for each state. We use a projected coordinate system because environmental models use length units such as meter, instead of degrees used in the geographic coordinate system. After creating the state-scale merged DEM, we applied the georeferencing tool to adjust the locations of the merged DEM to the original location. Land cover data are distributed at the national scale; therefore, we extracted state-scale land cover datasets and projected the data to the same coordinate system as state-scale DEM. Finally, SSURGO data, which is also distributed by the national scale using a USA Contiguous Albers Equal-Area Conic USGS version, was clipped to the state-scale and projected to the same coordinate system as the DEM and land cover data.

Figure 4.3. The workflows to create the state-scale large spatial sample datasets as GeoTIFF and NetCDF format

After creating the state-scale LES datasets, we added metadata directly within the NetCDF datasets to share key information in the original information distributed by GDG as a text file. There was a lot of metadata in the metadata text file and we selected and added 16 useful pieces of information such as data title, bounding coordinates, grid coordinate system name, UTM zone number, scale factor at central meridian, and horizontal datum name.

4.3.1.3 Share Datasets in HydroShare

For this step, we reviewed and used the three tools to share the state-scale LES datasets through HydroShare: 1) iRODS (Integrated Rule-Oriented Data System) to transfer large datasets (over 1 GB) into HydroShare, 2) OWSLib (https://github.com/geopython/OWSLib) and 3) xarray (http://xarray.pydata.org) Python libraries to make the LES datasets interoperable via GeoServer and TDS.

The first step was to upload the datasets into a new HydroShare resource. This step is trivial if the size of datasets is under 1 GB as datasets can then directly be upload through the HydroShare

user interface. However, for datasets over 1GB, users need to use iRODS for the transfer into HydroShare. There are multiple iRODS client including icommands and Cyberduck that can be used to upload large datasets into the HydroShare iRODS user space. After uploading the state-scale LES datasets into the HydroShare resource, datasets were automatically recognized with the proper aggregation type of geographic raster (GeoTIFF) or multidimensional contents (NetCDF). The content type metadata, such as title, keywords, spatial/temporal coverage, and spatial reference, and variable metadata, were automatically extracted by HydroShare as part of the upload process. When the HydorShare resource is made public, it automatically makes the spatial datasets available through GeoServer and TDS.

After the LES datasets are available on HydroShare and through the linked GeoServer and TDS access points, users can easily discover and programmatically interact with the LES datasets from GeoServer and TDS using the newly created HydroShare resource ID. In the GeoServer, users can use OWSLib to request subsets of data from GeoServer. In TDS, users can use xarray to subset particular data of interest. Then users can convert NetCDF output from TDS into GeoTIFF using the rioxarray (https://github.com/corteva/rioxarray) package for using the data as input to environmental models that expect GeoTIFF raster inputs.

4.3.2 Example Application for an Environmental Model Use Case

We used an example application to demonstrate the data service and how it supports seamless, end-to-end environmental modeling workflows. Figure 4.4 shows the workflow steps from seamlessly applying state scale LES datasets as model input for environmental modeling using the DEM, extracted land cover and soil texture maps after creating model inputs. We also used streamflow outputs from the environmental model as part of the evaluation of data consistency between the LES datasets compared to raw datasets provided by federal data providers. We used RHESSys (Tague & Band, 2004b) as the example model. RHESSys is a GIS-based, hydro-ecological modeling framework designed to simulate carbon, water, and nutrient fluxes. The newly developed pyRHESSys (https://github.com/uva-hydroinformatics/pyRHESSys) is an API for RHESSys providing programmatic control of model input creation and manipulation, model execution, and model output visualization and analysis.

We compared three approaches for accessing the spatial data required to parameterize the RHESSys model. 1) The original spatial data provided by federal agencies, 2) the data processed and distributed through GeoServer in the GeoTIFF format, and 3) the same data processed and distributed through TDS in a NetCDF format. We compared the watershed DEM, extracted land cover, and SSURGO data to evaluate data consistency across the three approaches. Then, after executing RHESSys using data from these three approaches as input, we compared streamflow outputs to evaluate the effect of the three data access approaches on model output.

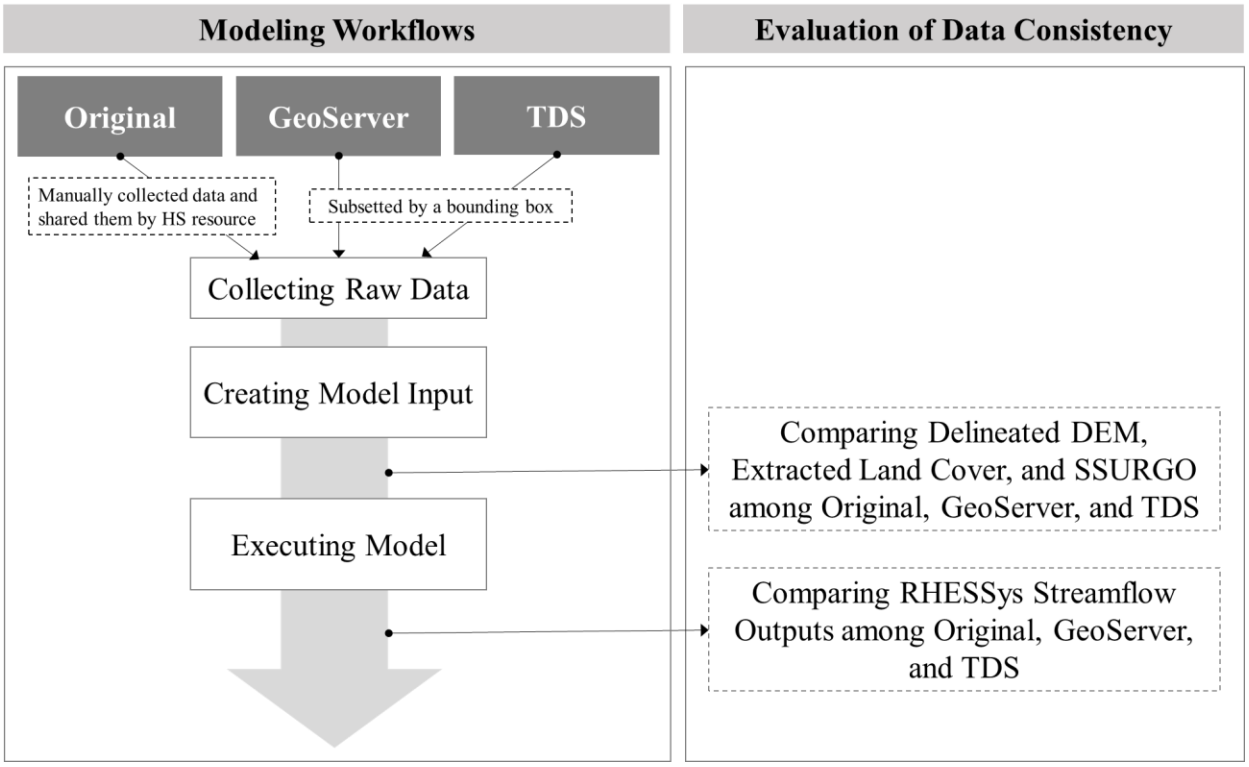

Figure 4.4. Workflows for seamless RHESSys modeling and evaluation of data consistency using LES datasets

4.4 Results

In this section, we present results from the example application where the methodology is applied for three different watersheds for end-to-end modeling using RHESSys. We present the resulting workflows as Jupyter notebooks that show how to create and subset state-scale LES datasets based on Figure 4.1 in the methodology section. Finally, we present results from the evaluation methodology described in Figure 4.4 that aims to measure data consistency across three approaches for distributing spatial data and how each method impacts the results of an end-to-end RHESSys model run.

4.4.1 Example Watersheds

The three watersheds used in this study are 1) Coweeta subbasin18, NC (A=0.126 $km^{2,}$ resolution: 10m), 2) Scotts Level Branch, MD (A=8.36 $km^{2,}$ resolution: 30m), 3) Spout Run, VA (A=55.42 $km^{2,}$ resolution: 60m). The Coweeta Long Term Ecological Research (LTER) station has been measuring hydrologic and ecologic variables from 1980 to 2020, and subbasin 18 is a forest-dominated Coweeta subbasin with moderate topographic relief that is often used in hydrologic studies. Scotts Level Branch is located near Baltimore Maryland and has a USGS streamflow observation station (USGS 01589290) and represents an agricultural watershed. Spout Run is located in Northern Virginia and has a USGS streamflow observation station (USGS 01636316). In addition, there is a neighboring site that is a part of the National Science

Foundation's National Ecological Observatory Network (NEON). Using these three different sized watersheds, we evaluate the applicability of the LES dataset distribution method as part of an end-to-end modeling workflow.

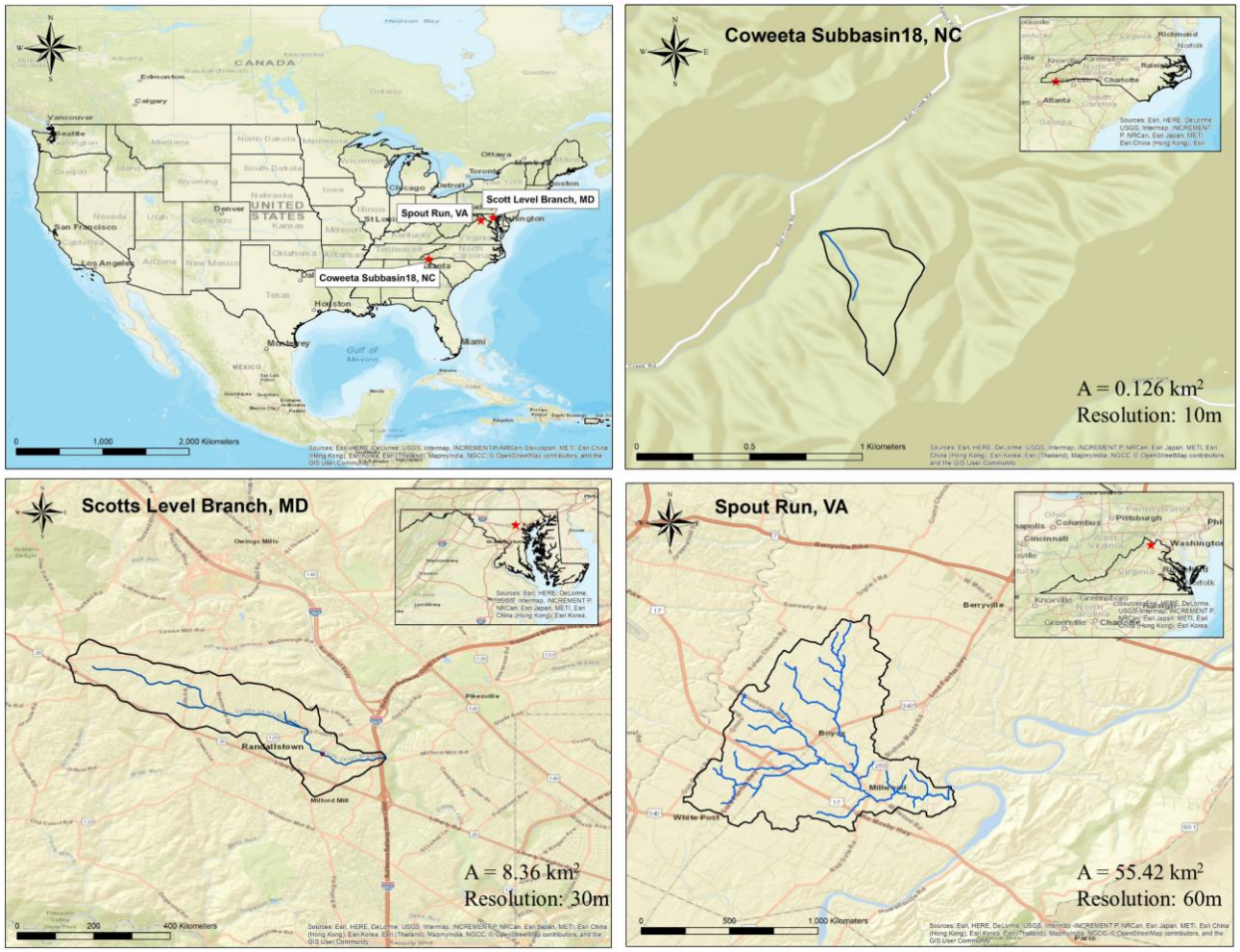

Figure 4.5 Three different scale watersheds to evaluate data consistency in different resolutions using state scale LES datasets: 1) Coweeta subbasin18, NC (A=0.126 km^{2,} resolution: 10m), 2) Scotts Level Branch, MD (A=8.36 km^{2,} resolution: 30m), 3) Spout Run, VA (A=55.42 km^{2,} resolution: 60m)

4.4.2 Creating the State-Scale LES Datasets

We created a Jupyter notebook for each state to automate the data processing workflow required to create the LES datasets (Choi., 2021) (HS 17). "HS number" used to distinguish each HS resource in the collection HS resource that includes 18 HS resources lists (Choi., 2021) and 18 HS resources are explained in Data availability section. In these workflows, GIS processing was done first to merge, extract and project GeoTIFF data was the most important process. For this process, we used ArcPy which is a python package to perform geographic data analysis, data conversion, and data management in ArcGIS (Toms, 2015). After creating state scale LES datasets in GeoTIFF format, we converted GeoTIFF to NetCDF using xarray and rioxarray Python packages. Xarray is a Python package to work with multi-dimensional arrays and rioxarray is rasterio xarray extension. Rasterio is a Python library to read and write GeoTIFF and other raster formats. We used xarray to manipulate data type and add metadata in NetCDF file and rioxarray

to save GeoTIFF to NetCDF format. Through these procedures, we created three composite HydroShare resources to share state scale LES datasets (Choi., 2021) (HS 2-4).

The automated workflows consist of three parts (DEM, land cover, and SSURGO) as we mentioned in Figure 4.3 in subsection 4.3. In this section, we demonstrated an example to create Virginia LES DEM as a GeoTIFF format (Figure 4.6). Before starting this procedure, we created Arcpy Conda virtual environments from ArcGIS Pro 2.1. Then we created Jupyter notebooks to capture these automated workflows (Figure 4.6). We imported required libraries such as Arcpy, xarray, and numpy. Then, after collecting 30m resolution DEM from GDG, we unified the multiple projected coordinate systems of the original DEMs into one projected coordinate system using *ProjectRaster_management* module in Arcpy. In the case of Virginia, DEM has UTM Zone 17N and 18N projected coordinate system, so we unified them to UTM Zone 17N. After that, we merged each DEM into one state scale DEM as GeoTIFF format using *MosaicToNewRaster_management* module in Arcpy. After that, we read the created Virginia GeoTIFF file and created xarray data format using rasterio Python library. Then we added metadata of original DEM, such as spatial domain, UTM detail information, and geodetic model information, into NetCDF. Finally, we saved a xarray format data to NetCDF. Following similar procedures, we created state scale LES land cover and SSURGO using this Jupyter notebook. Due to the limitation of ArcGIS Pro license which is a commercial GIS software, we developed these Jupyter notebooks in the Windows OS. Therefore, researchers cannot use these notebooks in CyberGIS Jupyter for Water. To create state scale LES datasets to different states, researchers can create state scale LES datasets using these Jupyter notebooks.

```
# Import required libraries
import arcpy
import glob
import xarray as xr
import os, shutil
import numpy as np

# Unify the original DEM into one projected coordinate system
proj_path = "C:/Users/Choi/Documents/large_sample_datasets/DEM_STATES/Virginia \
             /elevation_NED30M_va_3902660_02/Project_UTM17"
for i, value in enumerate(raw_dem):
    dataset = xr.open_rasterio(value)
    CRS = dataset.crs.split(":")[1]
    if CRS == '26918':
        arcpy.ProjectRaster_management(value, proj_path+"/"+ raw_dem[i].split("\\")[-1],
            "PROJCS['NAD_1983_UTM_Zone_17N',GEOGCS['GCS_North_American_1983',\
            DATUM['D_North_American_1983',SPHEROID['GRS_1980',6378137.0,298.257222101]],\
            PRIMEM['Greenwich',0.0],UNIT['Degree',0.0174532925199433]],\
            PROJECTION['Transverse_Mercator'],PARAMETER['False_Easting',500000.0],\
            PARAMETER['False_Northing',0.0],PARAMETER['Central_Meridian',-81.0],\
            PARAMETER['Scale_Factor',0.9996],PARAMETER['Latitude_Of_Origin',0.0],\
            UNIT['Meter',1.0]]", "NEAREST", "30 30", "", "", "PROJCS['NAD_1983_UTM_Zone_18N',\
            GEOGCS['GCS_North_American_1983',DATUM['D_North_American_1983',\
            SPHEROID['GRS_1980',6378137.0,298.257222101]],PRIMEM['Greenwich',0.0],\
            UNIT['Degree',0.0174532925199433]],PROJECTION['Transverse_Mercator'],\
            PARAMETER['False_Easting',500000.0],PARAMETER['False_Northing',0.0],\
            PARAMETER['Central_Meridian',-75.0],PARAMETER['Scale_Factor',0.9996],\
            PARAMETER['Latitude_Of_Origin',0.0],UNIT['Meter',1.0]]", "NO_VERTICAL")
    elif CRS == '26917':
        shutil.copy(value, proj_path)
    else:
        print("Need to add more coordinate system to project DEM")

# Merge the DEM into one state-scale DEM
merged_dem_name = "VA_DEM30m_UTM17.tif"
collect_path = "C:/Users/Choi/Documents/large_sample_datasets/Large_Datasets/Virginia"
arcpy.MosaicToNewRaster_management(proj_dem, collect_path, merged_dem_name, "", \
                                   "32_BIT_FLOAT", "", "1", "LAST", "FIRST")

# Open GeoTIFF using rasterio library
dem_dataset = xr.open_rasterio(os.path.join(collect_path, merged_dem_name))
dem_dataset

# Read metadata from a text file and add them into NetCDF
dataset1 = []
filename = os.path.join(raw_dem_path, 'gway_3902660_02_NED30M.txt')
with open(filename, 'r') as f_in:
    for items in f_in:
        dataset1.append(items.split("\n"))
new_dem_dataset.attrs.update({dataset1[4][0].split(":")[0].split()[0]: dataset1[4][0].split(":")[1]})

# Save xarray to NetCDF
new_dem_dataset.to_netcdf(os.path.join(collect_path,'VA_DEM30m_UTM17.nc'))
```

Figure 4.6. An automated workflow to create Virginia LES DEM as a GeoTIFF and NetCDF format

Using these automated workflows, we created GeoTIFF and NetCDF LES datasets for the DEM, land cover, and SSURGO in North Carolina, Maryland, and Virginia. In this study, considering drainage areas of Coweeta Subbasin18 and the evaluation of higher resolution data consistency, we created 10 m resolution DEM as GeoTIFF in North Carolina. However, we could not create NetCDF LES datasets due to memory limitation in both a local computer and CyberGIS Jupyter for water. In Maryland and Virginia, we created 30 m resolution LES DEM in GeoTIFF format. We applied 30 m resolution DEM to Scotts Level Branch as an example for general resolution application. Considering drainage areas of Spout Run and the evaluation of lower

resolution data consistency, we resampled 30 m to 60 m resolution DEM. Land cover from MLRC and SSURO from GDG only have 30 meter resolution GeoTIFF; therefore, we created 30 m resolution land cover and SSURGO state scale LES datasets. After creating GeoTIFF LES datasets for DEM, land cover, and SSURGO, we converted them to NetCDF LES datasets. Since NetCDF is easy to create time or variable stacked NetCDF in the same domain using dimension and coordinate structures, we created a stacked land cover NetCDF LES dataset using seven GeoTIFF LES datasets from 2011 to 2016.

Table 4.1 shows the file sizes and resolutions of GeoTIFF and NetCDF LES datasets in the three states. In general, the original LES datasets which is not the compressed file are very big to control. To minimize the file sizes, we used a compressed format of GeoTIFF and NetCDF. We used a LZW compression algorithm (Akoguz et al., 2016) for GeoTIFF in ArcGIS. The nccopy tool, which is a command-line utility to compress NetCDF files, supports to specify the level of compression (level 0-9, a high value supports high compression and requires more time) for variable data in NetCDF. We used the level 1 compression command "*nccop –d1 input.nc output.nc*" to create compressed NetCDF. For example, the original size of Virginia GeoTIFF DEM (30m) was 1.53 GB, and the compressed size was 0.95 GB. The original size of Virginia NetCDF DEM (30m) was 1.60 GB and the compressed size was 0.78 GB. Therefore, for the convenience of creating and transferring data according to the file size, we recommend using the compressed format. In the case of North Carolina GeoTIFF DEM (10m), the file size was 5.66 GB which is very big; however, there was no problem to create GeoTIFF DEM. Yet converting GeoTIFF to NetCDF LES datasets had a problem of memory limitation in both the local computer and CyberGIS Jupyter for water, we could not create NetCDF LES datasets. To create, upload and interoperate large datasets, the capacity of local computer and server is important. However, it is difficult to give an exact guideline for the size of large datasets. Especially, the server of GeoServer and TDS requires a significant amount of testing considering how many and what size of transfers are occurring simultaneously using various combinations. Therefore, based on these experiences, we recommend using 1~2 GB for the size of large datasets on GeoServer and TDS in HydroShare.

Table 4.1. Compressed file sizes and resolutions of GeoTIFF and NetCDF in the three states

States		DEM		Land Cover (7 Years)		SSURGO	
		GeoTIFF	NetCDF	GeoTIFF	NetCDF	GeoTIFF	NetCDF
North Carolina	File Size (MB)	5,659	-	257	304	112	80
Maryland		358	294	134	165	52	44
Virginia		951	783	342	422	157	121
North Carolina	Resolution (m)	10	-	30	30	30	30
Maryland		30	30	30	30	30	30
Virginia		60	60	30	30	30	30

4.4.3 Subset State Scale Large Spatial Sample Datasets

After creating three states' LES datasets and sharing them on the HydroShare GeoServer and TDS, we subsetted these datasets to collect spatial model input in specific watersheds for RHESSys preprocessing. In Figures 4.7 and 4.8, we presented Scotts Level Branch, MD to

demonstrate how to subset LES datasets on GeoServer and TDS in HydroShare. This subsetting procedure is shared with RHESSys workflow notebooks in HydroShare (Choi., 2021) (HS 5-13).

4.4.3.1 Subset LES Datasets from GeoServer

We used OWSLib to subset GeoTIFF DEM from GeoServer using the Jupyter notebook in CyberGIS-Jupyter for Water. Figure 4.7 shows the procedure to subset from the Maryland LES Datasets (GeoTIFF) from GeoSever in HydroShare, as an example. First, we imported the required Python libraries to use WCS service in GeoServer. Second, we requested GeoTIFF as an object using a *WebCoverageService* module in OWSLib. Then we subsetted certain areas using a *getCoverage* method with a bounding box. Finally, we saved the subsetted object to GeoTIFF format.

```
# Import required libraries
from owslib.wcs import WebCoverageService
import shutil, os

# Set name and HS resource ID
PROJECT_NAME = "SLB_30m"
name = "geoserver_dem30m.tif"
gis_folder = os.path.join(os.getcwd(), PROJECT_NAME, "gis_data")
resourcd_id = "4f5a33d96a004bd496747956c45cae7a"
dem_name = "MD_DEM30m_UTM18"

# Subset GeoTIFF using A WCS reques from HydroShare-provisioned GeoServer
url = "https://geoserver.hydroshare.org/geoserver/wcs?service=WCS&version=1.1.0&request=GetCapa
    bilities&namespace=HS-"+resourcd_id+":"+dem_name
dem = WebCoverageService("https://geoserver.hydroshare.org/geoserver/wcs", version='2.0.1')
dem_subset=dem.getCoverage(identifier=["HS-"+resourcd_id+":"+dem_name],
                           subsets=[('E',x_min, x_max), ('N',y_min,y_max)], format='image/tiff')

# Save subsetted GeoTIFF
dem_tif=dem_subset.read()

f = open('./'+name, 'wb')
f.write(dem_tif)
f.close()
```

Figure 4.7. Example of subsetting LES Datasets (GeoTIFF) from GeoServer using OWSLib

4.4.3.2 Subset Large Spatial Sample Datasets from TDS

This procedure to subset land cover LES Datasets in TDS (NetCDF) is simpler than the procedure for subsetting in GeoTIFF because users can access directly TDS using xarray. Therefore, users can easily create xarray array format (*xarray.DataArray*) using a xarray *open_dataset* module. Then users can subset the land cover data in the Scotts Level Branch watershed for the year 2006 using slicing range for x and y coordinate and years (Figure 4.8). Finally, users can convert xarray data array to GeoTIFF format using the rioxarry library.

```
# Import required libraries
import xarray as xr
import rioxarray
import shutil

# Read NetCDF using xarray
nlcd = xr.open_dataset('http://thredds.hydroshare.org/thredds/dodsC/hydroshare/resources/ \
                        4f5a33d96a004bd496747956c45cae7a/data/contents/MN_NLCD30m_UTM18.nc')

# Subset NetCDF using x, y range and land cover year
nlcd_subset = nlcd.sel(y=slice(4362188, 4357920), x=slice(340347, 349049), years=2006)

# Save xarray to GeoTIFF format
nlcd_subset.rio.to_raster("opendap_nlcd_2006_30m.tif")
```

Figure 4.8. Example of subsetting LES Datasets (NetCDF) from TDS using xarray

4.4.4 Evaluation of Data Consistency

In this section, we created RHESSys input from state-scale LES datasets and executed RHESSys using an end-to-end RHESSys Jupyter notebook (Choi., 2021) (HS 5-13) to evaluate the data consistency in three different watersheds with different spatial data resolutions: 10, 30, and 60 m. In these procedures, we created nine case studies using three different datasets (Original data, GeoServer, and TDS) and three watersheds (Coweeta subbasin18, Scotts Level Branch, and Spout Run). For evaluation of the original datasets, the spatial datasets were manually collected to represent the traditional approach, we created three model instance resources for each watershed in HydroShare (Choi., 2021) (HS 14-16). We then presented the evaluation results of data consistency in three different watersheds using Jupyter notebooks (Choi., 2021) (HS 18). For evaluation, we used difference maps between original data and LES datasets (GeoServer and TDS) for model inputs (watershed DEMs, extracted land covers, and SSURGO maps) (Figure 4.9-4.12) and regression plots for model outputs (RHESSys streamflow outputs) (Figure 4.13-4.16).

4.4.4.1 Evaluation of Spatial Model Input

Results of the data consistency analysis where RHESSys ouputs were compared before and after applying georeferencing to the LES datasets. As we explained earlier, when we create LES datasets, the application of appropriate coordinate systems and the georeferencing tool is important. Researchers often consider the appropriate coordinate systems because improper coordinate systems can cause errors. However, if researchers do not apply the georeferencing steps described in this paper, in some cases, environmental models can be executed without any errors depending on the watershed. In other cases, the shape of delineated watershed is changed, so users can recognize the problem. In the first cases, users can not reproduce the same results if there is a prior study. However, if users do not have the prior result, they may think the LES datasets are the same as the original data. They set up the models and tune the model parameters. Therefore, Figures 4.9, 4.10, and 4.11 demonstrate how much with or without the application of georeferencing affects spatial model input between original data (HydroShare) and LES datasets (GeoServer or TDS).

Figure 4.9 shows the differences in the DEM elevation between the original data (HydroShare) and the LES datasets (GeoServer or TDS) at (a) Coweeta Subbbasin18 (10 m), NC, (b) Scott Level Branch (30 m), MD, and (c) Spout Run (60 m), VA before applying georeferencing. Coweeta Subbasin18 (Figure 4.9 (a)) is forest dominant watershed and others are urban watersheds. Therefore, Coweeta Subbasin18 shows the biggest elevation differences compared to other two watersheds. In addition, the cell values (elevation) of the raw DEM are float data type, thus the values of most cells are changed by resampling and merging multiple DEMs, it is not meaningful to present the ratio of changed DEM cells. We present the ratio of land cover and soil maps because these used land cover classification code and soil texture code as discrete integer values.

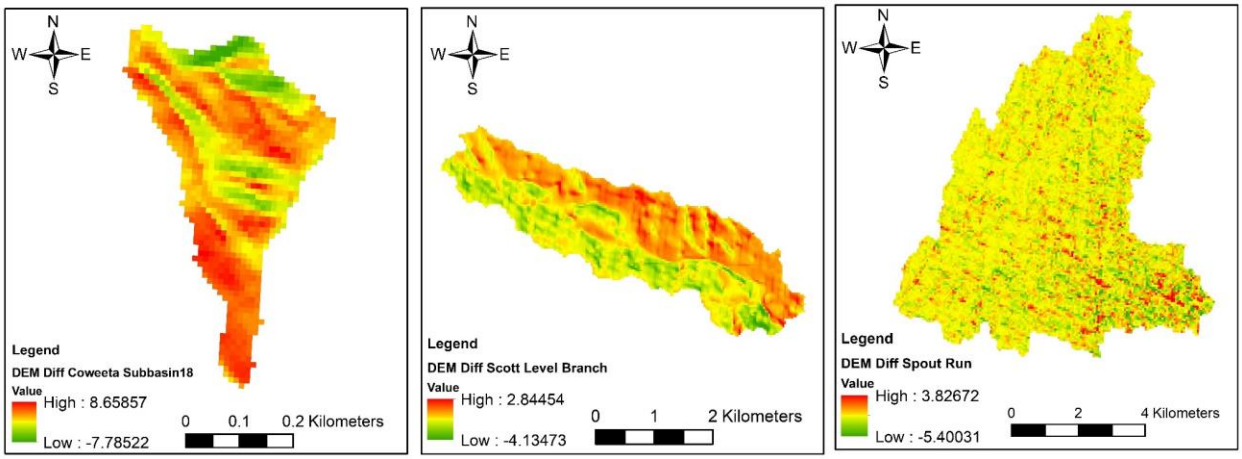

Figure 4.9. Difference maps of DEM elevation between original data (HydroShare) and LES datasets (GeoServer or TDS) at (a) Coweeta Subbbasin18, NC, (b) Scott Level Branch, MD, (c) Spout Run, VA, before applying georeferencing

Figure 4.10 shows the differences of extracted land cover classification code between original data (HydroShare) and LES datasets (GeoServer or TDS) at (a) Coweeta Subbbasin18, NC, (b) Scott Level Branch, MD, (c) Spout Run, VA, before applying georeferencing. The original land covers were extracted from national scale land cover maps, therefore, theoretically, the value of original data and LES data in the same location are the same. However, in the model preprocessing, watershed DEMs are used to extract land cover and soil maps. So watershed DEMs affect the difference of land cover classification codes. In Figure 4.10, the red color indicates a negative difference (= original cell values – GeoServer cell values) and the blue color indicates a positive difference (= original cell values – TDS cell values). Figure 4.10 (a), Figure 4.10 (b), and Figure 4.10 (c) show 4.2%, 9.1%, and 9.4% (= the count of the changed cells/the count of the total cells) of land cover were changed by creating LES datasets and RHESSys preprocessing.

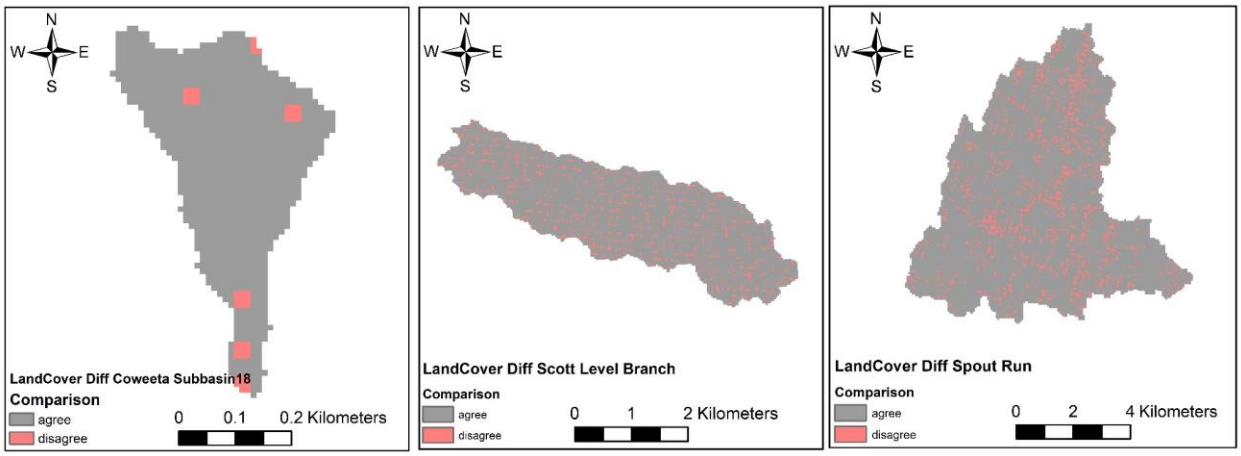

Figure 4.10. Difference maps of extracted Land Cover classification code between original data and LES datasets (GeoServer or TDS) at (a) Coweeta Subbbasin18, NC, (b) Scott Level Branch, MD, (c) Spout Run, VA, before applying georeferencing

Figure 4.11 shows the differences of extracted SSURGO soil texture between the original data and the LES datasets (GeoServer or TDS) at (a) Coweeta Subbbasin18, NC, (b) Scott Level Branch, MD, (c) Spout Run, VA before applying georeferencing. The reason for the difference in the land cover is the same as the DEM data discussed earlier. Figure 4.11 (a), Figure 4.11 (b), and Figure 4.11 (c) show 18.0%, 4.9%, and 5.3% (= the count of the changed cells/the count of the total cells) of soil maps were changed by creating LES datasets and RHESSys preprocessing.

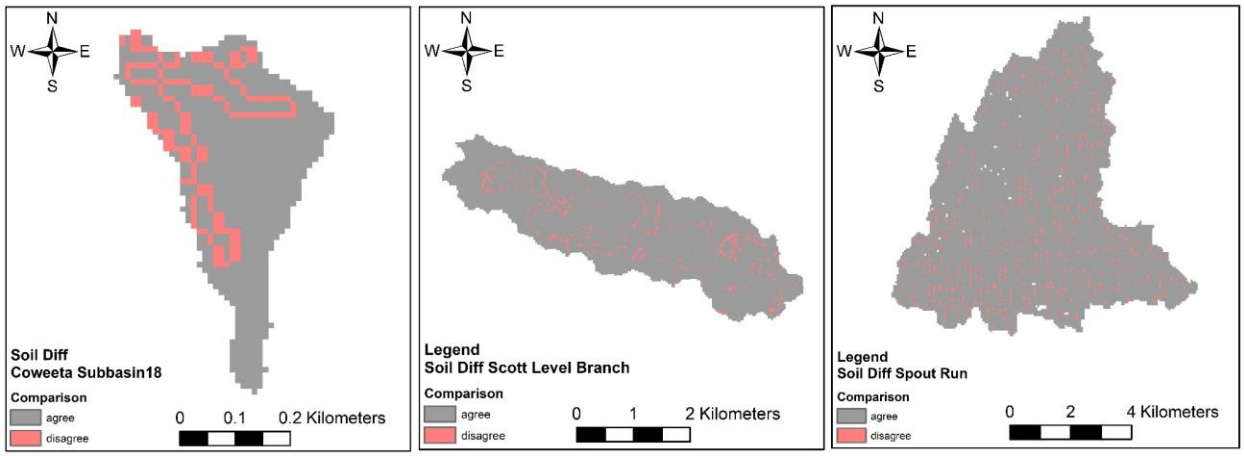

Figure 4.11. Difference maps of extracted SSURGO soil texture between original data (HydroShare) and LES datasets (GeoServer or TDS) at (a) Coweeta Subbbasin18, NC, (b) Scott Level Branch, MD, (c) Spout Run, VA, before applying georeferencing

Figure 4.12 shows the differences of (a) the watershed DEM elevation, extracted Land Cover classification code, and extracted SSURGO soil texture at Coweeta Subbbasin18, NC, (b) the watershed DEM elevation, extracted Land Cover classification code, and extracted SSURGO soil textures at Scott Level Branch, MD, (c) the watershed DEM elevation between original data (HydroShare) and LES datasets (GeoServer or TDS) after applying georeferencing. From the recognition of differences between the original data and LES datasets (GeoServer or TDS), we applied a georeferencing tool in ArcGIS. In general, georeferenced method use at least three points

to transform a raster or shapefile; however, resampling and merging multiple GeoTiff files only changed the cell values and shifted the cell location without distortion Therefore, we only used one control point to shift linearly the merged DEM to the original location. As a result, we can eliminate the DEM differences except for the DEM of Spout Run (60 m resolution) because Coweeta Subbasin 18 (30 m resolution) and Scott Level Branch (30 m resolution) have the same DEM resolution as the original DEM and LES DEM. However, in the case of Spout Run (60 m resolution), we resampled the 30 m to 60 m resolution to evaluate the applicability of large resolution. Therefore, every difference map of the DEMs, land cover, and soil in Coweeta Subbasin18 and Scott Level Branch is the same.

Coweeta subbasin18 and Scott Level Branch used the same resolution of DEM as the original data. However, Spout Run resampled the original DEM (30 m) to 60 m resolution to evaluate the applicability of different resolutions for the large watershed. This is the reason why there are still slight differences between original data (HydroShare) and LES datasets (GeoServer or TDS). However, the range of difference in elevation is below 5 cm.

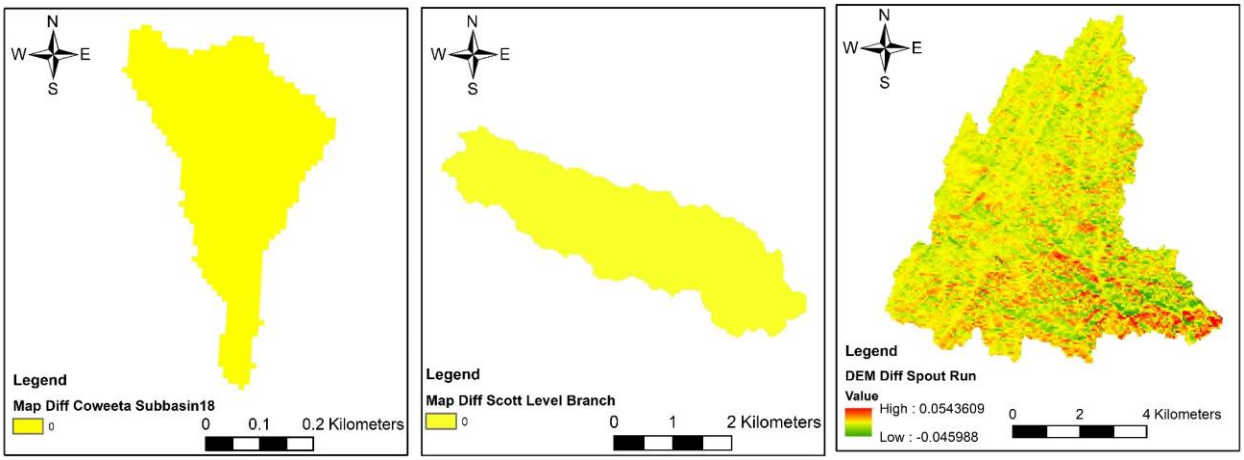

Figure 4.12. Difference maps of (a) DEM elevation (in meters), extracted Land Cover classification code, and extracted SSURGO soil texture at Coweeta Subbbasin18, NC, (b) DEM elevation, extracted Land Cover classification code, and extracted SSURGO soil texture at Scott Level Branch, MD, (c) DEM between the original data (HydroShare) and LES datasets (GeoServer or TDS), after applying georeferencing

4.4.4.2 Evaluation of Model Output

Figures 4.13-4.16 show three regression analyses, each comparing two RHESSys outputs from the three different data input approaches: original, GeoServer, and TDS. The results are provided for the three watersheds: Coweeta subbasin18 in North Carolina, Scott Level Branch in Maryland, and Spout Run in Virginia. At first, to emphasize the importance of georeferencing, we presented Figure 4.13 to explain the performance results of RHESSys outputs without appling the georeferencing tool (Original vs GeoServer: NSE 0.684, Original vs TDS: NSE 0.647). As explained earlier, after applying the georeferencing tool we significantly improved the results and the RHESSys outputs from the original data compared to the LES data. Figure 4.14 (a) and Figure 4.14 (b) showed perfect agreement (Original vs GeoServer: NSE 1.0, Original vs TDS: NSE 1.0). Also, Figures 4.15 and 4.16 showed perfect agreement after applying the georeferencing tool. As

result, the application of georeferencing is important for data consistency so that spatial data input results in the expected modeled streamflow output.

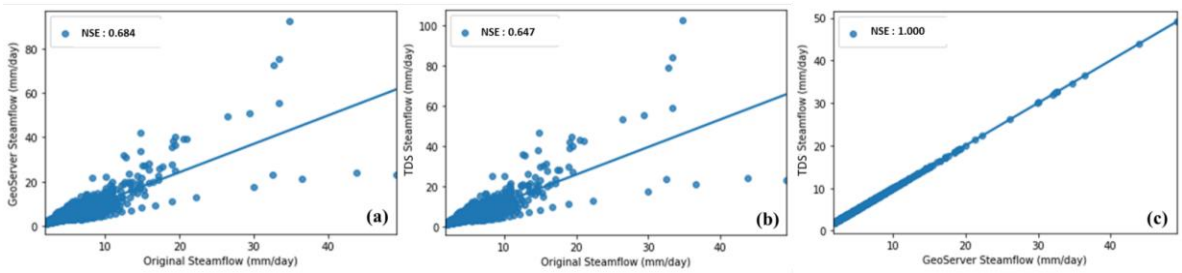

Figure 4.13. Three regression analyses, each comparing two RHESSys outputs out of the three different approaches: (a) original vs GeoServer, (b) original vs TDS, and (c) Geoserver vs TDS at Coweeta subbasin18 in North Carolina, before applying georeferencing

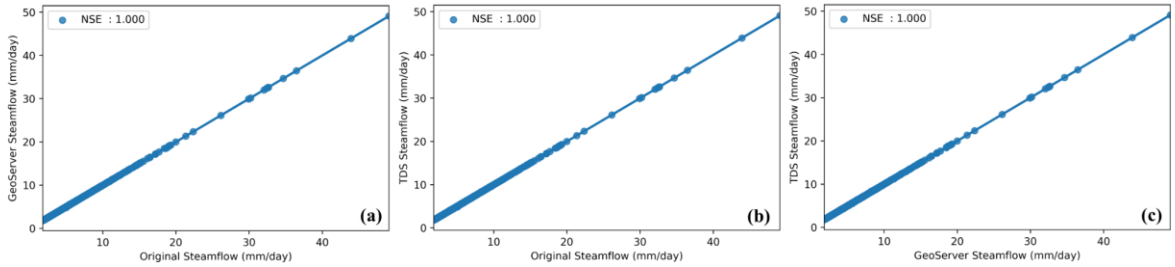

Figure 4.14 Three regression analyses, each comparing two RHESSys outputs out of the three different approaches: (a) original vs GeoServer, (b) original vs TDS, and (c) Geoserver vs TDS at Coweeta subbasin18 in North Carolina, after applying georeferencing

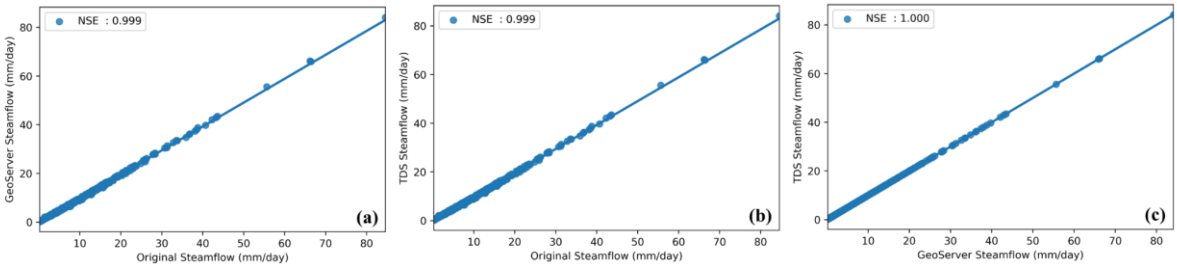

Figure 4.15 Three regression analyses, each comparing two RHESSys outputs out of the three different approaches: (a) original vs GeoServer, (b) original vs TDS, and (c) Geoserver vs TDS at Scotts Level Branch in Maryland, after applying georeferencing

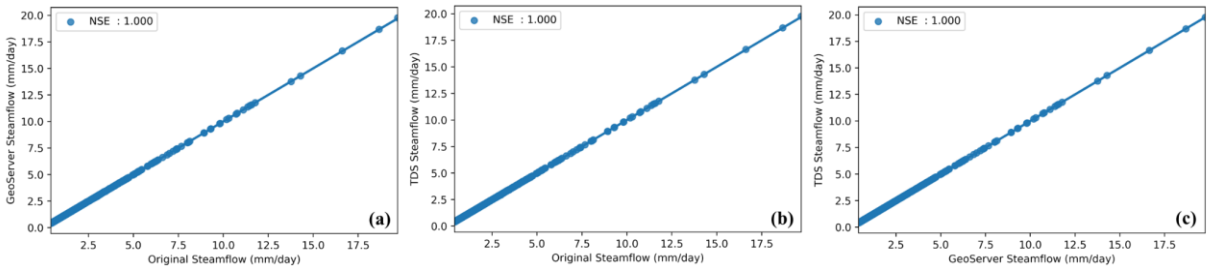

Figure 4.16 Three regression analyses, each comparing two RHESSys outputs out of the three different approaches: (a) original vs GeoServer, (b) original vs TDS, and (c) Geoserver vs TDS at Spout Run in Virginia, after applying georeferencing

4.5 Discussion

This research focuses on integrating HydroShare with GeoServer and TDS for exposing LES datasets to environmental models for open and reproducible seamless environmental modeling. This approach improved limitations of previous spatial data sharing for hydrologic research. However, there are still limitations to use various types of geographic datasets. We mainly used GeoTIFF format with GeoServer, though shapefile is also a popular geographic format to share spatial data. However, subsetting the SSURGO shapefile requires more memory because it is very complicated to support the heterogenous details of soil attributes in a shapefile format, unlike grid-based geographic data. Therefore, to effectively use state-scale LES datasets in the future, additional approaches or capabilities to leverage shapefiles with attributes (dbf table) that are well suited for environmental modeling and that require a similar amount of memory as grids would be valuable.

In this study, we used state-scale as the spatial unit for distributing LES datasets. However, other spatial aggregations may also be used. For example, for hydrology applications alone the Hydrologic Unit Code (HUC) (Seaber et al., 1987) may make more sense than administrative maps such as states. The advantage for using states as the spatial unit is current federal web-based distribution systems easily provide the data with this aggregation, but it requires minimal processing to prepare the data for GeoServer and TDS distribution. However, the general methods of the approach are agnostic to the specific spatial aggregation unit used. Therefore, if researchers want to set up datasets for inter-state boundary watersheds, they can follow the same steps for creating and sharing the datasets presented in this study.

In the introduction we compared our approach to two other approaches for supporting seamless environmental modeling in the literature: EcoHydroLib and HydroTerre. A third approach that is important to consider is the Google Earth Engine (GEE) (Gorelick et al., 2017). GEE is a cloud-based platform for planetary-scale geospatial analysis supporting such applications as climate change, disease, environmental protection, and water management. Over 450 journal articles published in 150 journals have used GEE datasets (L. Kumar & Mutanga, 2018), and the datasets available through GEE are continuously updated at a rate of nearly 6000 scenes per day (Gorelick et al., 2017). However, while GEE does allow users to upload their own data, datasets

like SSURGO are not natively included in GEE. More importantly, it is valuable to have an open and fully transparent alternative to GEE to support scientific modeling where users have control over the spatial data used as model input and the data can be easily shared with appropriate metadata from HydroShare.

In addition, containerizing proprietary software such as ArcPy, which is part of ArcGIS, is an issue for reproducibility because not everyone may have access to this software. In this study, we used ArcPy to create the LES datasets. ArcPy currently only supports Windows operating system, owever Docker Containers on Windows is experimental and only available on Window Server 2019. This said, it is possible to containerize proprietary software like ArcPy that operate only on Windows operating systems. Thus, if we can install the proprietary software with containerization tools in the same operating system and if there is no problem to recognize the license for the software, it is possible to containerize proprietary software. The issue remains as to access to the software license so that anyone, and not only those with access to the software, can reproduce the work.

The availability of data to support environmental modeling is increasing rapidly and the replication of this data across data distribution systems can present problems. For one, there may be issues of copyright for some data because these data represent important intellectual property (Abubahia & Cocea, 2017). Even if data can be freely used and copied, like the data used in this study, it will become increasingly difficult to understand if verified data are being used to support a study. In this study, we had to manipulate the raw data provided by the federal agencies in order to give the data a consistent and accurate spatial coordinate system. For reproducibility, it is important to document these changes and associate the procedure for making the changes with the new data product. As a result, geographic data ownership and provenance are important concepts (Licens. Geogr. Data Serv., 2004). In the broader technology landscape, similar challenges are being addressed through blockchain technology where a distributed digital ledger can be used to track changes to a digital object. Related to blockchain technology, the concept of Non-Fungible Tokens (NFTs) (Farnaghi & Mansourian, 2020; Franke et al., 2020) where digital objects are uniquely identified within the blockchain could prove valuable for identifying digital objects used in environmental modeling (e.g., both data and processing scripts) and tracking the provenance of these objects in a consistent, globally assessable, and secure way. Therefore, an extension of this work would be to consider spatial datasets as NFTs with ownership and provenance, thereby adding blockchain to the existing HydroShare data management capabilities, to clarify the specific attributes and provenance of the growing number of raw and processed datasets required in reproducible, open environmental modeling.

4.6 Conclusions

Spatial data is an important component for open and reproducible seamless environmental modeling. Recently, there have been many efforts to improve the use of spatial data as model input. HydroShare provides a means for easily sharing datasets including spatial datasets. It also provides the ability to expose spatial data stored in HydroShare through APIs for programmatic data access within environmental modeling. Currently, HydroShare provides the capability for

spatial datasets to be distributed using GeoServer and TDS, which each can be accessed using APIs. However, these capabilities have been underutilized to data, serving mainly data visualization use cases. This research demonstrates how these capabilities can be used to support seamless, end-to-end environmental modeling workflows.

Therefore, the primary contribution of this research is methodologies for integrating HydroShare with GeoServer and TDS for exposing LES datasets to models for open and reproducible seamless environmental modeling. We demonstrate how to create, share and subset large datasets as input to environmental model, thereby advancing the concept of seamless modeling where model inputs can be constructed using end-to-end workflows driven using common base datasets. Through three different watershed applications in three different states at three different spatial scales, we show the applicability of methodologies. We show, using the RHESSys model for each watershed, that no significant error is introduced when using the new data distribution system compared to traditional approaches. We offer discussion on ways the proposed approach can be further advanced by, for example, using other spatial aggregation of large data, beyond the state-scale aggregation used in this paper. Finally, we offer discussion on the challenge of data tracking and provenance, especially across systems and in the context of environmental modeling where data from multiple sources is needed and each dataset requires extensive preprocessing. We suggest that blockchain technology and the concept of NFTs could offer assistances to this problem by identifying universally unique digital assets in data processing and modeling workflows common in environmental modeling. Using these novel technologies could offer a way to improve reproducibility in complex digital workflows where the management of data and provenance tracking across various data providers and processing steps remains a challenge in achieving the vision of open science.

Beyond sharing large, national-scale datasets maintained by federal agencies, the methods used in this work can also be deployed by individual scientists. Current data sharing through online repositories allows for data publication at the file level, which is an important step toward reproducibility. The proposed approach builds on this step to show how technologies like GeoServer and TDS, when integrated with an online repository, provides a means for scientists to create and share file-based scientific data in a way that provides programmatic access, without the need to deploy their own web-based data distribution systems. Scientists can easily share, update, and extend their data through such systems, including HydroShare as demonstrated in this research, to support reproducibility and replicability through robust API-based access to their data. Creating and sharing datasets online using this approach offer a powerful means for scientists to achieve FAIR guiding principles including reusability of data for multiple applications in different case studies and interoperability for programmatic access to multiple data collections using a consistent access protocol.

Data Availability

All data used in this study are available through eighteen HydroShare resources. We published all data with persistent digital object identifiers (DOI’s) on HydroShare and shared all

data in a collection resource in HydroShare (Choi., 2021). This collection resource provides the links for all HydroShare resources as "Collection Contents." Eighteen HydroShare resources consist of the following: one collection resource (HS 1), three composite resources for three state scale LES datasets (HS 2-4), nine composite resources with Jupyter notebooks for three different approaches and three different watersheds (HS 5-13), three model instance resources for RHESSys input of the original approaches in three different watersheds (HS 14-16), one composite resource with Jupyter notebooks for automate workflows to create LES datasets (HS 17), and one composite resource with Jupyter notebooks for evaluation of data consistency (HS 18).

List of Relevant URLs

Chesapeake Conservancy conservation innovation center:

https://www.chesapeakeconservancy.org/conservation-innovation-center

Creating Python Conda virtual environment (arcpy) in ArcGIS Pro:

https://pro.arcgis.com/en/pro-app/latest/arcpy/get-started/work-with-python-environments.htm

CUAHSI JupyterHub: https://jupyterhub.cuahsi.org

CyberGIS-Jupyter for water: http://go.illinois.edu/cybergis-jupyter-water

CyberDuck: https://cyberduck.io

Cyberduck application: https://help.hydroshare.org/creating-and-managing-resources/accessing-hydroshare-irods-from-a-windows-pc-or-mac

icommands: https://help.hydroshare.org/creating-and-managing-resources/accessing-hydroshare-irods-from-linux

MRLC (Multi-Resolution Land Characteristics Consortium): https://www.mrlc.gov

National scale SSURGO 30 meter resolution GeoTIFF data:

https://nrcs.app.box.com/v/soils/folder/132131296196

nccopy: https://www.unidata.ucar.edu/software/netcdf/workshops/2011/utilities/Nccopy.html

OGC implementation standard: http://docs.opengeospatial.org/is/19-008r4/19-008r4.html

OWSLib: https://github.com/geopython/OWSLib

pyRHESSys: https://github.com/uva-hydroinformatics/pyRHESSys

rioxarray: https://github.com/corteva/rioxarray

SSUGRO Mukey Grids (GeoTIFF): https://nrcs.app.box.com/v/soils/folder/132131296196

USDA NRCS Geospatial Data Gateway: https://datagateway.nrcs.usda.gov

USGS 3D Elevation Program (3DEP): https://www.usgs.gov/core-science-systems/ngp/3dep

Web Soil Survey web distributed system:

https://websoilsurvey.sc.egov.usda.gov/App/WebSoilSurvey.aspx

xarray: http://xarray.pydata.org

Chapter 5

Conclusions

This dissertation presents research that advances approaches for improving openness, reproducibility, and replicability in computational environmental modeling. These approaches can supplement the current reproducibility research that individually focuses on sharing online data, containerizing computational environments, and encapsulating computational workflows. In addition, each advanced approach for each component (computational environments and modeling workflows) can strengthen reproducibility for computational environmental modeling. The contributions are 1) the development of an approach for integrating online data repositories, computational environments, and model APIs to enable more open and reproducible environmental modeling, 2) suggestion of best practices and guidance for which approach is most appropriate to achieve modeling objectives, specifically for simulating environmental systems, and 3) the integration of HydroShare with server-side methods (GeoServer and TDS) using large-extent spatial datasets for open and reproducible seamless environmental modeling.

This research also highlights the selection and integration of key components for reproducibility in computational environmental modeling. Chapter 2 presents an example implementation of this approach by leveraging 1) HydroShare as a data sharing repository, 2) CUAHSI JupyterHub and CyberGIS Jupyter for water as a notebook-based, containerized, and cloud-based computational environment, and 3) pySUMMA as an example model API able to abstract lower-level details for model configuration, execution, and visualization from end users. Using the example implementation, I demonstrate how modeling analyses can be completed in a more open and reproducible way using a prior study presenting a series of modeling experiments applying SUMMA at the Reynolds Mountain East Area in the Reynolds Creek Experimental Watershed in Idaho, USA (Clark et al., 2015b). As part of the research, I created a prototype version of pySUMMA, which is a Python based model API for manipulating, executing, and analyzing the SUMMA hydrological model.

The research in this dissertation also presents best practices and guidance to reproduce computational environments for achieving various modeling objectives. In Chapter 3, the example application was evaluated using 1) HydroShare as an online repository, 2) SUMMA as the core software and other secondary software, and 3) pySUMMA and Jupyter notebooks for a model API and workflows. These example results show that each method had its own strengths and weaknesses from the developer and user's perspectives. With regard to educational purposes, the best methods for online education were using CUAHSI JupyterHub, CyberGIS Jupyter for Water, and Binder (Approaches 6, 7, and 10) and the best method for offline education was using Sciunit (Approach 5). With regard to research purposes, the only method for model development was

compiling the core model software (Approach 1), the best methods for model application were containrizing the core model software only with Docker and using CUAHSI JupyterHub (Approaches 2 and 6), and the best methods for data-intensive computing were using CyberGIS Jupyter for Water and a HPC cluster (Approaches 7 and 11).

Finally, the research in this dissertation advances methods for the preprocessing of data to serve as input to environmental modeling. This was done by integrating HydroShare with GeoServer and TDS for exposing LES datasets to models for seamless environmental modeling. Chapter 4 presented three example watersheds were this method was evaluated: 1) Coweeta subbasin18, NC, 2) Scotts Level Branch, MD, and 3) Spout Run, VA. The results show data consistency in RHESSys model input and output after running the RHESSys end-to-end modeling workflows. This approach can save significant time to collect, clean, and apply large-extent spatial data for environmental models. Finally, this approach can inspire scientists to share their data without the need to deploy their own web-based data distribution system.

Although this dissertation focuses on the computational environmental modeling, it could be applied to broader areas of computational research to enhance reproducibility and replicability. For example, the integration of three key components (i.e., online repositories, computational environments, and model APIs) will be helpful for reproducibility in computational geospatial science, bioinformatics, and many other research areas. Researchers can use best practices and guidance for containerization of scientific modeling workflows from the second study for other computational research in addition to environmental modeling. Lastly, integrating cyberinfrastructures with server-side methods for exposing large datasets for various analyses can be helpful for researchers in other fields as well. In particular, these approaches can be a specific guidance for achieving FAIRer guiding principles in open science. Lastly, the hope of this research is for these approaches to aid in fostering a “culture of reproducibility” within scientific communities that rely on computational research to advancement of science.

References

Abubahia, A., & Cocea, M. (2017). Advancements in GIS map copyright protection schemes - a critical review. *Multimedia Tools and Applications*, *76*(10). https://doi.org/10.1007/s11042-016-3441-z

Addor, N., Newman, A. J., Mizukami, N., & Clark, M. P. (2017). The CAMELS data set: catchment attributes and meteorology for large-sample studies. *Hydrology and Earth System Sciences Discussions*. https://doi.org/10.5194/hess-2017-169

Addor, N., Do, H. X., Alvarez-Garreton, C., Coxon, G., Fowler, K., & Mendoza, P. A. (2020). Large-sample hydrology: recent progress, guidelines for new datasets and grand challenges. *Hydrological Sciences Journal*, *65*(5). https://doi.org/10.1080/02626667.2019.1683182

Akoguz, A., Bozkurt, S., Gozutok, A. A., Alp, G., Turan, E. G., Bogaz, M., & Kent, S. (2016). Comparison of open source compression algorithms on VHR remote sensing images for efficient storage hierarchy. In *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences - ISPRS Archives* (Vol. 41). https://doi.org/10.5194/isprsarchives-XLI-B4-3-2016

Alvarez-Garreton, C., Mendoza, P. A., Pablo Boisier, J., Addor, N., Galleguillos, M., Zambrano-Bigiarini, M., et al. (2018). The CAMELS-CL dataset: Catchment attributes and meteorology for large sample studies-Chile dataset. *Hydrology and Earth System Sciences*, *22*(11). https://doi.org/10.5194/hess-22-5817-2018

Atmanspacher, H., Bezzola Lambert, L., Folkers, G., & Schubiger, P. A. (2014). Relevance relations for the concept of reproducibility. *Journal of the Royal Society Interface*, *11*(94). https://doi.org/10.1098/rsif.2013.1030

Avila, D., Bussonier, M., & Corlay, S. (2020). Jupyter.

Baker, Monya; (2016). 1,500 scientists lift the lid on reproducibility. *Nature*, *533*(May 26), 452–454. https://doi.org/10.1038/533452a

Baker, Monya. (2016). Is there a reproducibility crisis? *Nature*, 533.

Bakker, M., Post, V., Langevin, C. D., Hughes, J. D., White, J. T., Starn, J. J., & Fienen, M. N. (2016). Scripting MODFLOW Model Development Using Python and FloPy. *Groundwater*, *54*(5), 733–739. https://doi.org/10.1111/gwat.12413

Ball, J. T., Woodrow, I. E., & Berry, J. A. (1987). A Model Predicting Stomatal Conductance and its Contribution to the Control of Photosynthesis under Different Environmental Conditions. In *Progress in Photosynthesis Research* (pp. 221–224). Dordrecht: Springer Netherlands. https://doi.org/10.1007/978-94-017-0519-6_48

Bandaragoda, C., Castronova, A., Istanbulluoglu, E., Strauch, R., Nudurupati, S. S., Phuong, J., et al. (2019). Enabling Collaborative Numerical Modeling in Earth Sciences using Knowledge Infrastructure. *Environmental Modelling and Software*, *120*. https://doi.org/10.1016/j.envsoft.2019.03.020

References

Bast, R. (2019). A FAIRer future. *Nature Physics*. https://doi.org/10.1038/s41567-019-0624-3

Baumer, B., Çetinkaya-Rundel, M., Bray, A., Loi, L., & Horton, N. J. (2014). R Markdown: Integrating a reproducible analysis tool. *Technology Innovations in Statistics Education*, *8*(1).

Beaulieu-Jones, B. K., & Greene, C. S. (2017). Reproducibility of computational workflows is automated using continuous analysis. *Nature Biotechnology*, *35*(4). https://doi.org/10.1038/nbt.3780

Blair, G. S., Beven, K., Lamb, R., Bassett, R., Cauwenberghs, K., Hankin, B., et al. (2019). Models of everywhere revisited: A technological perspective. *Environmental Modelling and Software*, *122*. https://doi.org/10.1016/j.envsoft.2019.104521

Boettiger, C. (2015). An introduction to Docker for reproducible research. *ACM SIGOPS Operating Systems Review*, *49*(1), 71–79. https://doi.org/10.1145/2723872.2723882

Boulmaiz, T., Guermoui, M., & Boutaghane, H. (2020). Impact of training data size on the LSTM performances for rainfall–runoff modeling. *Modeling Earth Systems and Environment*, *6*(4). https://doi.org/10.1007/s40808-020-00830-w

Brinckman, A., Chard, K., Gaffney, N., Hategan, M., Jones, M. B., Kowalik, K., et al. (2019). Computing environments for reproducibility: Capturing the "Whole Tale." *Future Generation Computer Systems*. https://doi.org/10.1016/j.future.2017.12.029

Brooks, G. (2013). Benefits of APIs. Retrieved January 14, 2020, from https://digital.gov/2013/03/12/benefits-of-apis/

Buytaert, W. (2011). topmodel: Implementation of the hydrological model TOPMODEL in R. *Global Change Biology*. https://doi.org/10.1111/j.1365-2486.2006.01305.x

Castronova, A.; M. Seul, P. D. (2018). A General Approach for Cloud-based Hydrologic Modeling using Jupyter Notebooks. Retrieved from http://www.hydroshare.org/resource/075664b0f0df4c58892cb4665e77e497

Celia, M. A., Bouloutas, E. T., & Zarba, R. L. (1990). A general mass-conservative numerical solution for the unsaturated flow equation. *Water Resources Research*, *26*(7). https://doi.org/10.1029/WR026i007p01483

Ceola, S., Arheimer, B., Baratti, E., Blöschl, G., Capell, R., Castellarin, A., et al. (2015). Virtual laboratories: New opportunities for collaborative water science. *Hydrology and Earth System Sciences*. https://doi.org/10.5194/hess-19-2101-2015

Chen, M., Voinov, A., Ames, D. P., Kettner, A. J., Goodall, J. L., Jakeman, A. J., et al. (2020). Position paper: Open web-distributed integrated geographic modelling and simulation to enable broader participation and applications. *Earth-Science Reviews*. https://doi.org/10.1016/j.earscirev.2020.103223

Choi, Y.-D., Goodall, J. L., Sadler, J. M., Castronova, A. M., Bennett, A., Li, Z., et al. (2021). Toward open and reproducible environmental modeling by integrating online data repositories, computational environments, and model Application Programming Interfaces. *Environmental Modelling and Software*. https://doi.org/10.1016/j.envsoft.2020.104888

References

Choi, Y., Goodall., J., Sadler., J., Castronova., A. M., Bennett., A., LI., Z., et al. (2020). Toward Open and Reproducible Environmental Modeling by Integrating Online Data Repositories, Computational Environments, and Model Application Programming Interfaces. Retrieved from https://www.hydroshare.org/resource/33cfb9622a354442b2b0a869b15ea6b0/

Clark, M. P., Nijssen, B., Lundquist, J. D., Kavetski, D., Rupp, D. E., Woods, R. A., Freer, J. E., Gutmann, E. D., Wood, A. W., Brekke, L. D., et al. (2015). A unified approach for process-based hydrologic modeling: 1. Modeling concept. *Water Resources Research*. https://doi.org/10.1002/2015WR017198

Clark, M. P., Nijssen, B., Lundquist, J. D., Kavetski, D., Rupp, D. E., Woods, R. A., Freer, J. E., Gutmann, E. D., Wood, A. W., Gochis, D. J., et al. (2015). A unified approach for process-based hydrologic modeling: 2. Model implementation and case studies. *Water Resources Research*. https://doi.org/10.1002/2015WR017200

Craig, W., & Victoria., S. (2020). Trust but Verify: How to Leverage Policies, Workflows, and Infrastructure to Ensure Computational Reproducibility in Publication. *Harvard Data Science Review*. https://doi.org/doi.org/10.1162/99608f92.25982dcf

Crawley, S., Ames, D., Li, Z., & Tarboton, D. (2017). HydroShare GIS: Visualizing Spatial Data in the Cloud. *Open Water Journal*, *4*(1).

Crosas, M. (2020). Fair Principles and Beyond: Implementation in Dataverse. *Septentrio Conference Series*, (2). https://doi.org/10.7557/5.5334

CrowdFlower. (2016). Data Science Report - 2016.

DeVantier, B. A., & Feldman, A. D. (1993). Review of GIS Applications in Hydrologic Modeling. *Journal of Water Resources Planning and Management*, *119*(2). https://doi.org/10.1061/(asce)0733-9496(1993)119:2(246)

Duan, J., & Miller, N. L. (1997). A generalized power function for the subsurface transmissivity profile in TOPMODEL. *Water Resources Research*, *33*(11). https://doi.org/10.1029/97WR02186

Duan, Q., Schaake, J., Andréassian, V., Franks, S., Goteti, G., Gupta, H. V., et al. (2006). Model Parameter Estimation Experiment (MOPEX): An overview of science strategy and major results from the second and third workshops. In *Journal of Hydrology* (Vol. 320). https://doi.org/10.1016/j.jhydrol.2005.07.031

Epskamp, S. (2019). Reproducibility and Replicability in a Fast-Paced Methodological World. *Advances in Methods and Practices in Psychological Science*, *2*(2). https://doi.org/10.1177/2515245919847421

Essawy, B. T., Goodall, J. L., Zell, W., Voce, D., Morsy, M. M., Sadler, J., et al. (2018). Integrating scientific cyberinfrastructures to improve reproducibility in computational hydrology: Example for HydroShare and GeoTrust. *Environmental Modelling and Software*, *105*, 217–229. https://doi.org/10.1016/j.envsoft.2018.03.025

Essawy, B. T., Goodall, J. L., Voce, D., Morsy, M. M., Sadler, J. M., Choi, Y. D., et al. (2020). A taxonomy for reproducible and replicable research in environmental modelling. *Environmental Modelling and Software*, *134*. https://doi.org/10.1016/j.envsoft.2020.104753

Eynard-Bontemps, G., Abernathey, R., Hamman, J., Ponte, A., & Rath, W. (2019). The PANGEO Big Data Ecosystem and its use at CNES. In *Proc. of the 2019 conference on Big Data from Space (BiDS?2019), EUR 29660 EN* (pp. 49–52). Luxembourg: ARRAY(0xc86ce6c). https://doi.org/doi:10.2760/848593

Farnaghi, M., & Mansourian, A. (2020). Blockchain, an enabling technology for transparent and accountable decentralized public participatory GIS. *Cities*, *105*. https://doi.org/10.1016/j.cities.2020.102850

Forde, J., Head, T., Holdgraf, C., & Sundell, E. (2018). Reproducible Research Environments with Repo2Docker. In *Reproducibility in ML Workshop (ICML '18)*.

Franke, L., Schletz, M., & Salomo, S. (2020). Designing a blockchain model for the paris agreement's carbon market mechanism. *Sustainability (Switzerland)*, *12*(3). https://doi.org/10.3390/su12031068

Fuka, D. R., Walter, M. T., Macalister, C., Steenhuis, T. S., & Easton, Z. M. (2014). SWATmodel: A multi-operating system, multi-platform SWAT model package in R1. *Journal of the American Water Resources Association*, *50*(5). https://doi.org/10.1111/jawr.12170

Gan, T., Tarboton, D. G., Horsburgh, J. S., Dash, P., Idaszak, R., & Yi, H. (2020). Collaborative sharing of multidimensional space-time data in a next generation hydrologic information system. *Environmental Modelling and Software*, *129*. https://doi.org/10.1016/j.envsoft.2020.104706

Garijo, D., Kinnings, S., Xie, L., Xie, L., Zhang, Y., Bourne, P. E., & Gil, Y. (2013). Quantifying reproducibility in computational biology: The case of the tuberculosis drugome. *PLoS ONE*. https://doi.org/10.1371/journal.pone.0080278

Gil, Y., David, C. H., Demir, I., Essawy, B. T., Fulweiler, R. W., Goodall, J. L., et al. (2016). Toward the Geoscience Paper of the Future: Best practices for documenting and sharing research from data to software to provenance. *Earth and Space Science*, *3*(10), 388–415. https://doi.org/10.1002/2015EA000136

Goble, C., Cohen-Boulakia, S., Soiland-Reyes, S., Garijo, D., Gil, Y., Crusoe, M. R., et al. (2020). FAIR Computational Workflows. *Data Intelligence*. https://doi.org/10.1162/dint_a_00033

Gomes, J., Bagnaschi, E., Campos, I., David, M., Alves, L., Martins, J., et al. (2018). Enabling rootless Linux Containers in multi-user environments: The udocker tool. *Computer Physics Communications*, *232*. https://doi.org/10.1016/j.cpc.2018.05.021

Gorelick, N., Hancher, M., Dixon, M., Ilyushchenko, S., Thau, D., & Moore, R. (2017). Google Earth Engine: Planetary-scale geospatial analysis for everyone. *Remote Sensing of Environment*, *202*. https://doi.org/10.1016/j.rse.2017.06.031

Grüning, B., Chilton, J., Köster, J., Dale, R., Soranzo, N., van den Beek, M., et al. (2018). Practical Computational Reproducibility in the Life Sciences. *Cell Systems*. https://doi.org/10.1016/j.cels.2018.03.014

Gupta, H. V., Perrin, C., Blöschl, G., Montanari, A., Kumar, R., Clark, M., & Andréassian, V.

(2014). Large-sample hydrology: A need to balance depth with breadth. *Hydrology and Earth System Sciences*, *18*(2). https://doi.org/10.5194/hess-18-463-2014

Hamman, J., Rocklin, M., & Abernathy, R. (2018). *Pangeo: A Big-data Ecosystem for Scalable Earth System Science*. *Geophysical Research Abstracts* (Vol. 20).

Hobley, D. E. J., Adams, J. M., Siddhartha Nudurupati, S., Hutton, E. W. H., Gasparini, N. M., Istanbulluoglu, E., & Tucker, G. E. (2017). Creative computing with Landlab: An open-source toolkit for building, coupling, and exploring two-dimensional numerical models of Earth-surface dynamics. *Earth Surface Dynamics*, *5*(1), 21–46. https://doi.org/10.5194/esurf-5-21-2017

Hodson, S., Jones, S., Collins, S., Genova, F., Harrower, N., Laaksonen, L., et al. (2018). Turning FAIR data into reality: interim report from the European Commission Expert Group on FAIR data (Version Interim draft). *Interim Report from the European Commission Expert Group on FAIR Data*, (June).

Horsburgh, J. S., Morsy, M. M., Castronova, A. M., Goodall, J. L., Gan, T., Yi, H., et al. (2016). HydroShare: Sharing Diverse Environmental Data Types and Models as Social Objects with Application to the Hydrology Domain. *Journal of the American Water Resources Association*, *52*(4). https://doi.org/10.1111/1752-1688.12363

Hothorn, T., & Leisch, F. (2011). Case studies in reproducibility. *Briefings in Bioinformatics*, *12*(3). https://doi.org/10.1093/bib/bbq084

Hut, R. W., van de Giesen, N. C., & Drost, N. (2017, May 1). Comment on "Most computational hydrology is not reproducible, so is it really science?" by Christopher Hutton et al.: Let hydrologists learn the latest computer science by working with Research Software Engineers (RSEs) and not reinvent the waterwheel ourselves. *Water Resources Research*. Blackwell Publishing Ltd. https://doi.org/10.1002/2017WR020665

Hutton, C., Wagener, T., Freer, J., Han, D., Duffy, C., & Arheimer, B. (2016). Most computational hydrology is not reproducible, so is it really science? *Water Resources Research*, *52*(10), 7548–7555. https://doi.org/10.1002/2016WR019285

Ince, D. C., Hatton, L., & Graham-Cumming, J. (2012). The case for open computer programs. *Nature*, *482*(7386), 485–488. https://doi.org/10.1038/nature10836

Jarvis, P. (1976). The interpretation of the variations in leaf water potential and stomatal conductance found in canopies in the field. *Trans. R. Soc. B*, *273(927)*, 593–610. https://doi.org/10.1098/rstb.1976.0035

Jupyter Project, Bussonnier, M., Forde, J., Freeman, J., Granger, B., Head, T., et al. (2018). Binder 2.0 - Reproducible, interactive, sharable environments for science at scale. *Proceedings of the 17th Python in Science Conference*, (Scipy). https://doi.org/10.25080/majora-4af1f417-011

Kerandi, N., Arnault, J., Laux, P., Wagner, S., Kitheka, J., & Kunstmann, H. (2018). Joint atmospheric-terrestrial water balances for East Africa: a WRF-Hydro case study for the upper Tana River basin. *Theoretical and Applied Climatology*. https://doi.org/10.1007/s00704-017-2050-8

Kery, M. B., Radensky, M., Arya, M., John, B. E., & Myers, B. A. (2018). The story in the notebook: Exploratory data science using a literate programming tool. In *Conference on Human Factors in Computing Systems - Proceedings* (Vol. 2018-April). https://doi.org/10.1145/3173574.3173748

Kim, Y. M., Poline, J. B., & Dumas, G. (2018). Experimenting with reproducibility: A case study of robustness in bioinformatics. *GigaScience*. https://doi.org/10.1093/gigascience/giy077

Kluyver, T., Ragan-kelley, B., Pérez, F., Granger, B., Bussonnier, M., Frederic, J., et al. (2016). Jupyter Notebooks—a publishing format for reproducible computational workflows. *Positioning and Power in Academic Publishing:* https://doi.org/10.3233/978-1-61499-649-1-87

Knuth, D. E. (1984). LITERATE PROGRAMMING. *Computer Journal*. https://doi.org/10.1093/comjnl/27.2.97

Kouwen, N., Soulis, E. D., Pietroniro, A., Donald, J., & Harrington, R. A. (1993). Grouped Response Units for Distributed Hydrologic Modeling. *Journal of Water Resources Planning and Management*, *119*(3), 289–305. https://doi.org/10.1061/(ASCE)0733-9496(1993)119:3(289)

Kovács, Á. (2017). Comparison of different linux containers. In *2017 40th International Conference on Telecommunications and Signal Processing, TSP 2017* (Vol. 2017-January). https://doi.org/10.1109/TSP.2017.8075934

Kozhirbayev, Z., & Sinnott, R. O. (2017). A performance comparison of container-based technologies for the Cloud. *Future Generation Computer Systems*, *68*. https://doi.org/10.1016/j.future.2016.08.025

Kuentz, A., Arheimer, B., Hundecha, Y., & Wagener, T. (2017). Understanding hydrologic variability across Europe through catchment classification. *Hydrology and Earth System Sciences*, *21*(6). https://doi.org/10.5194/hess-21-2863-2017

Kumar, L., & Mutanga, O. (2018). Google Earth Engine applications since inception: Usage, trends, and potential. *Remote Sensing*, *10*(10). https://doi.org/10.3390/rs10101509

Kumar, M., Bhatt, G., & Duffy, C. J. (2010). An object-oriented shared data model for GIS and distributed hydrologic models. *International Journal of Geographical Information Science*, *24*(7). https://doi.org/10.1080/13658810903289460

Kurtzer, G. M., Sochat, V., & Bauer, M. W. (2017). Singularity: Scientific containers for mobility of compute. *PLoS ONE*, *12*(5), e0177459. https://doi.org/10.1371/journal.pone.0177459

Lampert, D. J., & Wu, M. (2015). Development of an open-source software package for watershed modeling with the Hydrological Simulation Program in Fortran. *Environmental Modelling & Software*, *68*, 166–174. https://doi.org/10.1016/J.ENVSOFT.2015.02.018

Laniak, G. F., Olchin, G., Goodall, J., Voinov, A., Hill, M., Glynn, P., et al. (2013). Integrated environmental modeling: A vision and roadmap for the future. *Environmental Modelling and Software*, *39*. https://doi.org/10.1016/j.envsoft.2012.09.006

References

Leonard, L., & Duffy, C. (2016). Visualization workflows for level-12 HUC scales: Towards an expert system for watershed analysis in a distributed computing environment. *Environmental Modelling and Software*. https://doi.org/10.1016/j.envsoft.2016.01.001

Leonard, L., & Duffy, C. J. (2013). Essential terrestrial variable data workflows for distributed water resources modeling. *Environmental Modelling and Software*. https://doi.org/10.1016/j.envsoft.2013.09.003

Leonard, L., & Duffy, C. J. (2014). Automating data-model workflows at a level 12 HUC scale: Watershed modeling in a distributed computing environment. *Environmental Modelling and Software*. https://doi.org/10.1016/j.envsoft.2014.07.015

Leonard, L., Miles, B., Heidari, B., Lin, L., Castronova, A. M., Minsker, B., et al. (2019). Development of a participatory Green Infrastructure design, visualization and evaluation system in a cloud supported jupyter notebook computing environment. *Environmental Modelling and Software*. https://doi.org/10.1016/j.envsoft.2018.10.003

Leonard, L. N. (2015). HydroTerre: Towards an expert system for scaling hydrological data and models from hill-slopes to major-river basins. *ProQuest Dissertations and Theses Global*.

Li, Y. (2020). Towards fast prototyping of cloud-based environmental decision support systems for environmental scientists using R Shiny and Docker. *Environmental Modelling and Software, 132*. https://doi.org/10.1016/j.envsoft.2020.104797

Licensing Geographic Data and Services. (2004). *Licensing Geographic Data and Services*. https://doi.org/10.17226/11079

de Lusignan, S., & van Weel, C. (2006). The use of routinely collected computer data for research in primary care: Opportunities and challenges. *Family Practice*. https://doi.org/10.1093/fampra/cmi106

Lyu, F., Yin, D., Padmanabhan, A., Choi, Y., Goodall, J. L., Castronova, A., et al. (2019). Reproducible Hydrological Modeling with CyberGIS-Jupyter: A Case Study on SUMMA. In *Proceedings of the Practice and Experience in Advanced Research Computing on Rise of the Machines (Learning)* (pp. 21:1--21:6). New York, NY, USA: ACM. https://doi.org/10.1145/3332186.3333052

Markham, K. (2019). Six easy ways to run your Jupyter Notebook in the cloud Retrieved from. Retrieved December 10, 2019, from https://www.dataschool.io/cloud-services-for-jupyter-notebook/

McDonnell, B., Ratliff, K., Tryby, M., Wu, J., & Mullapudi, A. (2020). PySWMM: The Python Interface to Stormwater Management Model (SWMM). *Journal of Open Source Software, 5*(52). https://doi.org/10.21105/joss.02292

McDonnell, B. E. (2017). Pyswmm 0.4. 5: Python Wrapper for SWMM5 API. Retrieved December 10, 2019, from https://github.com/OpenWaterAnalytics/pyswmm

McNutt, M. (2014). Reproducibility. *Science*. https://doi.org/10.1126/science.1250475

Merkel, D. (2014). Docker: lightweight Linux containers for consistent development and deployment. Retrieved January 21, 2020, from

https://www.linuxjournal.com/content/docker-lightweight-linux-containers-consistent-development-and-deployment

Miles, B., & Band, L. E. (2015). Ecohydrology Models without Borders? https://doi.org/10.1007/978-3-319-15994-2_31

Mizukami, N., Clark, M. P., Newman, A. J., Wood, A. W., Gutmann, E. D., Nijssen, B., et al. (2017). Towards seamless large-domain parameter estimation for hydrologic models. *Water Resources Research*, *53*(9). https://doi.org/10.1002/2017WR020401

Morsy, M. M., Goodall, J. L., Castronova, A. M., Dash, P., Merwade, V., Sadler, J. M., et al. (2017). Design of a metadata framework for environmental models with an example hydrologic application in HydroShare. *Environmental Modelling & Software*, *93*, 13–28. https://doi.org/10.1016/J.ENVSOFT.2017.02.028

National Academies of Sciences. (2019). *Reproducibility and Replicability in Science*. https://doi.org/https://doi.org/10.17226/25303

Nativi, S., Caron, J., Domenico, B., & Bigagli, L. (2008). Unidata's Common Data Model mapping to the ISO 19123 Data Model. *Earth Science Informatics*, *1*(2). https://doi.org/10.1007/s12145-008-0011-6

Newman, A. J., Clark, M. P., Sampson, K., Wood, A., Hay, L. E., Bock, A., et al. (2015). Development of a large-sample watershed-scale hydrometeorological data set for the contiguous USA: Data set characteristics and assessment of regional variability in hydrologic model performance. *Hydrology and Earth System Sciences*, *19*(1). https://doi.org/10.5194/hess-19-209-2015

Pauliuk, S. (2020). Making sustainability science a cumulative effort. *Nature Sustainability*. https://doi.org/10.1038/s41893-019-0443-7

Peckham, S. D., Hutton, E. W. H., & Norris, B. (2013). A component-based approach to integrated modeling in the geosciences: The design of CSDMS. *Computers and Geosciences*, *53*. https://doi.org/10.1016/j.cageo.2012.04.002

Pérez, F., & Granger, B. E. (2007). IPython: A system for interactive scientific computing. *Computing in Science and Engineering*, *9*(3). https://doi.org/10.1109/MCSE.2007.53

Piccolo, S. R., & Frampton, M. B. (2016). Tools and techniques for computational reproducibility. *GigaScience*, *5*(1), 30. https://doi.org/10.1186/s13742-016-0135-4

Pimentel, J. F., Murta, L., Braganholo, V., & Freire, J. (2019). A large-scale study about quality and reproducibility of jupyter notebooks. In *IEEE International Working Conference on Mining Software Repositories* (Vol. 2019-May). https://doi.org/10.1109/MSR.2019.00077

Prasad, C., Nancy, W., Mark, M., & Emre H, B. (2020). Measuring success for a future vision: Defining impact in science gateways/virtual research environments. *Concurrency Computation Practice and Experience*. https://doi.org/10.1002/cpe.6099

Reddy, M. (2011). *API design for c++*. https://doi.org/10.1016/C2010-0-65832-9

Rocklin, M. (2015). Dask: Parallel Computation with Blocked algorithms and Task Scheduling.

In *Proceedings of the 14th Python in Science Conference*. https://doi.org/10.25080/majora-7b98e3ed-013

Rosenberg, D. E., Filion, Y., Teasley, R., Sandoval-Solis, S., Hecht, J. S., van Zyl, J. E., et al. (2020). The Next Frontier: Making Research More Reproducible. *Journal of Water Resources Planning and Management*, *146*(6). https://doi.org/10.1061/(asce)wr.1943-5452.0001215

Rstudio Team. (2020). RStudio: Integrated development for R. RStudio, Inc., Boston MA. *RStudio*.

Sacks, J., Welch, W. J., Mitchell, T. J., & Wynn, H. P. (1989). Design and analysis of computer experiments. *Statistical Science*. https://doi.org/10.1214/ss/1177012413

Samaniego, L., Kumar, R., Thober, S., Rakovec, O., Zink, M., Wanders, N., et al. (2017). Toward seamless hydrologic predictions across spatial scales. *Hydrology and Earth System Sciences*, *21*(9). https://doi.org/10.5194/hess-21-4323-2017

Sandve, G. K., Nekrutenko, A., Taylor, J., & Hovig, E. (2013). Ten Simple Rules for Reproducible Computational Research. *PLoS Computational Biology*. https://doi.org/10.1371/journal.pcbi.1003285

Seaber, P. R., Kapinos, F. P., & Knapp, G. L. (1987). Hydrologic Unit Maps (USA). *US Geological Survey Water-Supply Paper*, *2294*. https://doi.org/10.3133/wsp2294

Shen, C. (2018). A Transdisciplinary Review of Deep Learning Research and Its Relevance for Water Resources Scientists. *Water Resources Research*. https://doi.org/10.1029/2018WR022643

Shuler, C. K., & Mariner, K. E. (2020). Collaborative groundwater modeling: Open-source, cloud-based, applied science at a small-island water utility scale. *Environmental Modelling and Software*, *127*. https://doi.org/10.1016/j.envsoft.2020.104693

Signell, R. P., & Pothina, D. (2019). Analysis and visualization of coastal ocean model data in the cloud. *Journal of Marine Science and Engineering*. https://doi.org/10.3390/jmse7040110

Slater, L. J., Thirel, G., Harrigan, S., Delaigue, O., Hurley, A., Khouakhi, A., et al. (2019). Using R in hydrology: A review of recent developments and future directions. *Hydrology and Earth System Sciences*. https://doi.org/10.5194/hess-23-2939-2019

Stagge, J. H., Rosenberg, D. E., Abdallah, A. M., Akbar, H., Attallah, N. A., & James, R. (2019). Assessing data availability and research reproducibility in hydrology and water resources. *Scientific Data*, *6*. https://doi.org/10.1038/sdata.2019.30

Stodden, V., & Miguez, S. (2013). Best Practices for Computational Science: Software Infrastructure and Environments for Reproducible and Extensible Research. *SSRN Electronic Journal*. https://doi.org/10.2139/ssrn.2322276

SUMMA Online Document. (2020). SUMMA Online Document. Retrieved February 25, 2020, from https://summa.readthedocs.io/en/latest/

Tague, C. L., & Band, L. E. (2004a). RHESSys: Regional Hydro-Ecologic Simulation System—An Object-Oriented Approach to Spatially Distributed Modeling of Carbon, Water, and Nutrient Cycling. *Earth Interactions*. https://doi.org/10.1175/1087-3562(2004)8<1:rrhsso>2.0.co;2

Tague, C. L., & Band, L. E. (2004b). RHESSys: Regional Hydro-Ecologic Simulation System—An Object-Oriented Approach to Spatially Distributed Modeling of Carbon, Water, and Nutrient Cycling. *Earth Interactions*, *8*(19). https://doi.org/10.1175/1087-3562(2004)8<1:rrhsso>2.0.co;2

Tague, C. L., Band, L. E., Tague, C. L., & Band, L. E. (2004). RHESSys: Regional Hydro-Ecologic Simulation System—An Object-Oriented Approach to Spatially Distributed Modeling of Carbon, Water, and Nutrient Cycling. *Earth Interactions*, *8*(19), 1–42. https://doi.org/10.1175/1087-3562(2004)8<1:RRHSSO>2.0.CO;2

Tarboton, David, Idaszak, Ray, & Horsburgh, J. (2018). HydroShare tools and recommended practices for sharing and publishing data and models in support of collaborative reproducible research. *AGU 2018 Fall Meeting*. https://doi.org/10.1002/essoar.10500174.1

Tarboton, D. G. (1997). A new method for the determination of flow directions and upslope areas in grid digital elevation models. *Water Resources Research*, *33*(2), 309–319. https://doi.org/10.1029/96WR03137

Tarboton, D. G., Idaszak, R., Horsburgh, J. S., Heard, J., Ames, D., Goodall, J. L., et al. (2014). Hydro share: Advancing collaboration through hydrologic data and model sharing. *Proceedings - 7th International Congress on Environmental Modelling and Software: Bold Visions for Environmental Modeling, IEMSs 2014*, *1*(October), 23–29. https://doi.org/10.13140/2.1.4431.6801

That, D. H. T., Fils, G., Yuan, Z., & Malik, T. (2017). Sciunits: Reusable research objects. In *Proceedings - 13th IEEE International Conference on eScience, eScience 2017*. https://doi.org/10.1109/eScience.2017.51

Toms, S. (2015). *ArcPy and ArcGIS - Geospatial Analysis with Python*. *Packt Publishing* (Vol. 1).

Viglione, A., & Parajka, J. (2020). TUWmodel. *R Package Documentation*. https://doi.org/10.1002/hyp.6253

Vogel, R. M., Lall, U., Cai, X., Rajagopalan, B., Weiskel, P. K., Hooper, R. P., & Matalas, N. C. (2015). Hydrology: The interdisciplinary science of water. *Water Resources Research*, *51*(6). https://doi.org/10.1002/2015WR017049

Volk, J. M., & Turner, M. A. (2019). PRMS-Python: A Python framework for programmatic PRMS modeling and access to its data structures. *Environmental Modelling and Software*. https://doi.org/10.1016/j.envsoft.2019.01.006

Wenjue, J., Yumin, C., & Jianya, G. (2004). Implementation of OGC web map service based on web service. *Geo-Spatial Information Science*, *7*(2). https://doi.org/10.1007/BF02826653

Wilkinson, M. D., Dumontier, M., Aalbersberg, Ij. J., Appleton, G., Axton, M., Baak, A., et al. (2016). Comment: The FAIR Guiding Principles for scientific data management and

stewardship. *Scientific Data*. https://doi.org/10.1038/sdata.2016.18

Wilkinson, M. D., Verborgh, R., da Silva Santos, L. O. B., Clark, T., Swertz, M. A., Kelpin, F. D. L., et al. (2017). Interoperability and FAIRness through a novel combination of Web technologies. *PeerJ Computer Science*, *2017*(4). https://doi.org/10.7717/peerj-cs.110

Wilson, G., Bryan, J., Cranston, K., Kitzes, J., Nederbragt, L., & Teal, T. K. (2017). Good enough practices in scientific computing. *PLoS Computational Biology*, *13*(6). https://doi.org/10.1371/journal.pcbi.1005510

Yi, H., Idaszak, R., Stealey, M., Calloway, C., Couch, A. L., & Tarboton, D. G. (2018). Advancing distributed data management for the HydroShare hydrologic information system. *Environmental Modelling and Software*. https://doi.org/10.1016/j.envsoft.2017.12.008

Yin, D., Liu, Y., Padmanabhan, A., Terstriep, J., Rush, J., & Wang, S. (2017). A cybergis-jupyter framework for geospatial analytics at scale. In *ACM International Conference Proceeding Series* (Vol. Part F128771). Association for Computing Machinery. https://doi.org/10.1145/3093338.3093378

Yuan, Z., Ton That, D. H., Kothari, S., Fils, G., & Malik, T. (2018). Utilizing provenance in reusable research objects. *Informatics*. https://doi.org/10.3390/informatics5010014

Appendix

Appendix-1 Total scores of complexity (Table A.1-7), size of reproducible artifacts (Table A.8-12), and reproduced figures (Figure A.1 and A.2)

Table A.1. An example of reproducible approach for Approach-1: Compiling Model Software

Developer Work

Steps	Level of Difficulty	Scores
1. Create a Reproducible Approach for SUMMA		**16**
1.1 Create SUMMA Makefile	Difficult	10
1.2 Compile and Create Binary Executable	Medium	5
1.3 Share Source Code and Makefile	Easy	1
2. Create a Reproducible Approach for pySUMMA		**6**
2.1 Create pySUMMA environment.yml	Medium	5
2.2 Share Source Code and environment.yml	Easy	1
3.Create a Reproducible Approach of Modeling Workflows		**12**
3.1 Create Jupyter Notebooks	Difficult	10
3.2 Share Jupyter Notebooks	Easy	1
3.3 Share SUMMA Input	Easy	1
Total Score		**34**

User Work

Steps	Level of Difficulty	Scores
1.Reproduce SUMMA		**7**
1.1 Download Source Code and Makefile	Easy	1
1.2 Edit Makefile	Medium	5
1.3 Compile and Create Binary Executable	Easy	1
2. Reproduce pySUMMA and Modeling Workflows		**2**
2.1 Download Jupyter Notebooks	Easy	1
2.2 Open and Run Jupyter Notebooks - Install pySUMMA - Download SUMMA input - Execute SUMMA	Easy	1
Total Score		**9**

Table A.2. An example of reproducible approach for Approach 3: Containerizing All Software with Docker

Developer Work

Steps	Level of Difficulty	Scores
1. Create a Reproducible Approach for SUMMA, pySUMMA, and Modeling Workflows		**23**
1.1 Create Jupyter Notebooks	Difficult	10
1.2 Create SUMMA and pySUMMA Dockerfile	Difficult	10
1.3 Create SUMMA and pySUMMA Docker Image	Easy	1
1.4 Share SUMMA and pySUMMA Docker Image	Easy	1
1.5 Share SUMMA Input	Easy	1
Total Score		**23**

User Work

Steps	Level of Difficulty	Scores
1.Reproduce SUMMA, pySUMMA, and Modeling Workflows		**3**
1.1 Install Docker	Easy	1
1.2 Download and Run Docker Image	Easy	1
1.3 Open and Run Jupyter Notebooks - Download SUMMA Input - Execute SUMMA	Easy	1
Total Score		**3**

Table A.3. An example of reproducible approach for Approach-4: Using Singularity

Developer Work

Steps	Level of Difficulty	Scores
1. Create a Reproducible Approach for SUMMA, pySUMMA, and Modeling Workflows		**13**
1.1 Create SUMMA and pySUMMA Definition file	Difficult	10
1.2 Create SUMMA and pySUMMA Singularity Image	Easy	1
1.3 Create Jupyter Kernel file	Easy	1
1.4 Upload SUMMA and pySUMMA Singularity image on HPC	Easy	1
2.Create a Reproducible Approach of Modeling Workflows		**12**
2.1 Create Jupyter Notebooks	Difficult	10
2.2 Share Jupyter Notebooks	Easy	1
2.3 Share SUMMA Input	Easy	1
Total Score		**25**

User Work

Steps	Level of Difficulty	Scores
1.Reproduce SUMMA, pySUMMA, and Modeling Workflows		**5**
1.1 Install Singularity	Easy	1
1.2 Install Anaconda and Jupyter	Easy	1
1.3 Download Jupyter Notebooks	Easy	1
1.4 Create Jupyter Kernel	Easy	1
1.5 Open and Run Jupyter Notebooks - Download Singularity image - Create Jupyter Kernel - Download SUMMA input - Execute SUMMA	Easy	1
Total Score		**5**

Table A.4. An example of reproducible approach for Approach-5, 8 and 9: Using Sciunit

Developer Work

Steps	Level of Difficulty	Scores
1. Create a Reproducible Approach for SUMMA, pySUMMA, and Modeling Workflows		**17**
1.1 Create Python or Shell Script to Execute SUMMA	Difficult	10
1.2 Create Jupyter Notebook to encapsulate Sciunit Workflow	Medium	5
1.3 Create SUMMA Sciunit Container	Easy	1
1.4 Share SUMMA Sciunit Container and Jupyter Notebooks	Easy	1
Total Score		**17**

User Work

Steps	Level of Difficulty	Scores
1.Reproduce SUMMA, pySUMMA, and Modeling Workflows		**2**
1.1 Download Sciunit Container and Jupyter Notebooks	Easy	1
1.2 Open and Run Jupyter Notebooks - Install Sciunit - Execute Sciunit	Easy	1
Total Score		**2**

Table A.5. An example of reproducible approach for Approach-6 and 7: Using CJH and CJW

Developer Work

Steps	Level of Difficulty	Scores
1. Create a Reproducible Approach for SUMMA and pySUMMA		**18**
1.1 Git Clone CJH or CJW Dockerfile Repository	Easy	1
1.2 Create Dockerfile for SUMMA and pySUMMA	Difficult	10
1.3 Pull Request to add Dockerfile into CJH or CJW	Easy	1
1.4 Review Dockerfile by CJH or CJW Development Team	Medium	5
1.5 Share the SUMMA and pySUMMA Docker image on CJH or CJW	Easy	1
2.Create a Reproducible Approach of Modeling Workflows		**12**
2.1 Create Jupyter Notebooks	Difficult	10
2.2 Share Jupyter Notebooks	Easy	1
2.3 Share SUMMA Input	Easy	1
Total Score		**30**

User Work

Steps	Level of Difficulty	Scores
1.Reproduce Modeling Workflows		**2**
1.1 Log in CJH or CJW	Easy	1
1.2 Open and Run Jupyter Notebooks - Download SUMMA input - Execute SUMMA * To use XSEDE from CJW, we need additional work - Create Singularity image to upload into XSEDE - Create Python code to interact between CJW and XSEDE for SUMMA modeling	Easy	1
Total Score		**2**

Table A.6. An example of reproducible approach for Approach 10: Using Binder

Developer Work

Steps	Level of Difficulty	Scores
1. Create a Reproducible Approach for SUMMA, pySUMMA, and Modeling Workflows		**24**
1.1 Create SUMMA and pySUMMA Configuration file for Binder	Difficult	10
1.2 Create Jupyter Notebooks	Difficult	10
1.3 Share SUMMA and pySUMMA Configuration file and Jupyter Notebooks on Online Repositories	Easy	1
1.4 Create Binder for SUMMA and pySUMMA	Easy	1
1.5 Share Binder Link	Easy	1
1.6 Share SUMMA input	Easy	1
Total Score		**24**

User Work

Steps	Level of Difficulty	Scores
1.Reproduce SUMMA, pySUMMA, and Modeling Workflows		**2**
1.1 Click Binder Link	Easy	1
1.2 Open and Run Jupyter Notebooks - Download SUMMA input - Execute SUMMA	Easy	1
Total Score		**2**

Table A.7. An example of reproducible approach for Approach 11: Using a HPC Cluster

Developer Work

Steps	Level of Difficulty	Scores
1. Create a Reproducible Approach for SUMMA, pySUMMA, and Modeling Workflows		**13**
1.1 Create SUMMA and pySUMMA Definition file	Difficult	10
1.2 Create SUMMA and pySUMMA Singularity Image	Easy	1
1.3 Create Jupyter Kernel file	Easy	1
1.4 Upload SUMMA and pySUMMA Singularity image on HPC	Easy	1
2. Create a Reproducible Approach of Modeling Workflows		**12**
2.1 Create Jupyter Notebooks	Difficult	10
2.2 Share Jupyter Notebooks	Easy	1
2.3 Share SUMMA Input	Easy	1
Total Score		**25**

User Work

Steps	Level of Difficulty	Scores
1.Reproduce SUMMA, pySUMMA, and Modeling Workflows		**4**
1.1 Log in HPC	Easy	1
1.2 Download Jupyter Notebooks	Easy	1
1.3 Create Jupyter Kernel	Easy	1
1.4 Open and Run Jupyter Notebooks - Download Singularity image - Create Jupyter Kernel - Download SUMMA input - Execute SUMMA * Don't need to install Singularity in HPC because generally Singularity is preconfigured.	Easy	1
Total Score		**4**

Table A.8. The size of reproducible artifacts for Approach-1: Compiling Model Software

Dependencies	Size (MB)
Ubuntu	8,870
Anaconda	6,382
SUMMA	28
pySUMMA	3,169
Subtotal (w/o Ubuntu)	9,578
Total	**18,448**

Table A.9. The size of reproducible artifacts for Approach 2: Containerizing Model Software only with Docker

Dependencies	Size (MB)
Ubuntu	8,870
Anaconda	6,382
Docker tool	386
Docker image (SUMMA)	615
pySUMMA	3,169
Subtotal (w/o Ubuntu)	10,551
Total	**19,421**

Table A.10. The size of reproducible artifacts for Approach 3: Containerizing All Software with Docker

Dependencies	Size (MB)
Ubuntu	8,870
Docker tool	386
Docker image (SUMMA, Anaconda, pySUMMA)	5,770
Subtotal (w/o Ubuntu)	6,156
Total	**15,026**

Table A.11. The size of reproducible artifacts for Approach-4: Using Singularity

Dependencies	Size (MB)
Ubuntu	8,870
Singularity tool	1,005
Singularity image (SUMMA, Anaconda, pySUMMA)	2,900
Subtotal (w/o Ubuntu)	3,905
Total	**12,775**

Table A.12. The size of reproducible artifacts for Approach-5, 8 and 9: Using Sciunit

Dependencies	Size (MB)
Ubuntu	8,870
Sciunit tool	6
Sciunit image (SUMMA, Anaconda, pySUMMA)	306
Subtotal (w/o Ubuntu)	312
Total	**9,182**

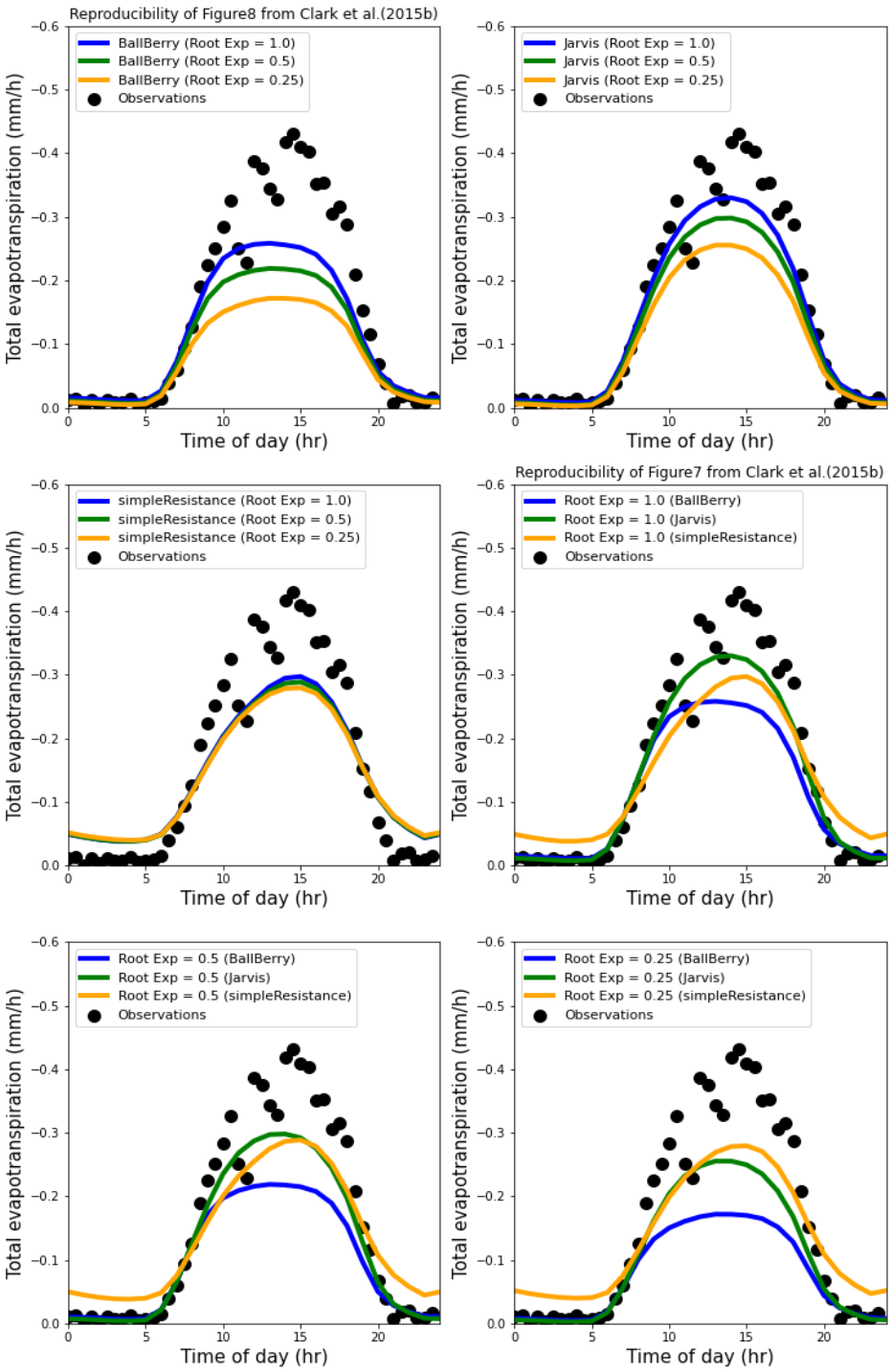

Figure A.1 Reproduced and additional figures of figure 7 and 8 from Clark et al. (2015b) showing the impact of the three different stomatal resistance parameterizations on total evapotranspiration based on Scenario 1 and 2

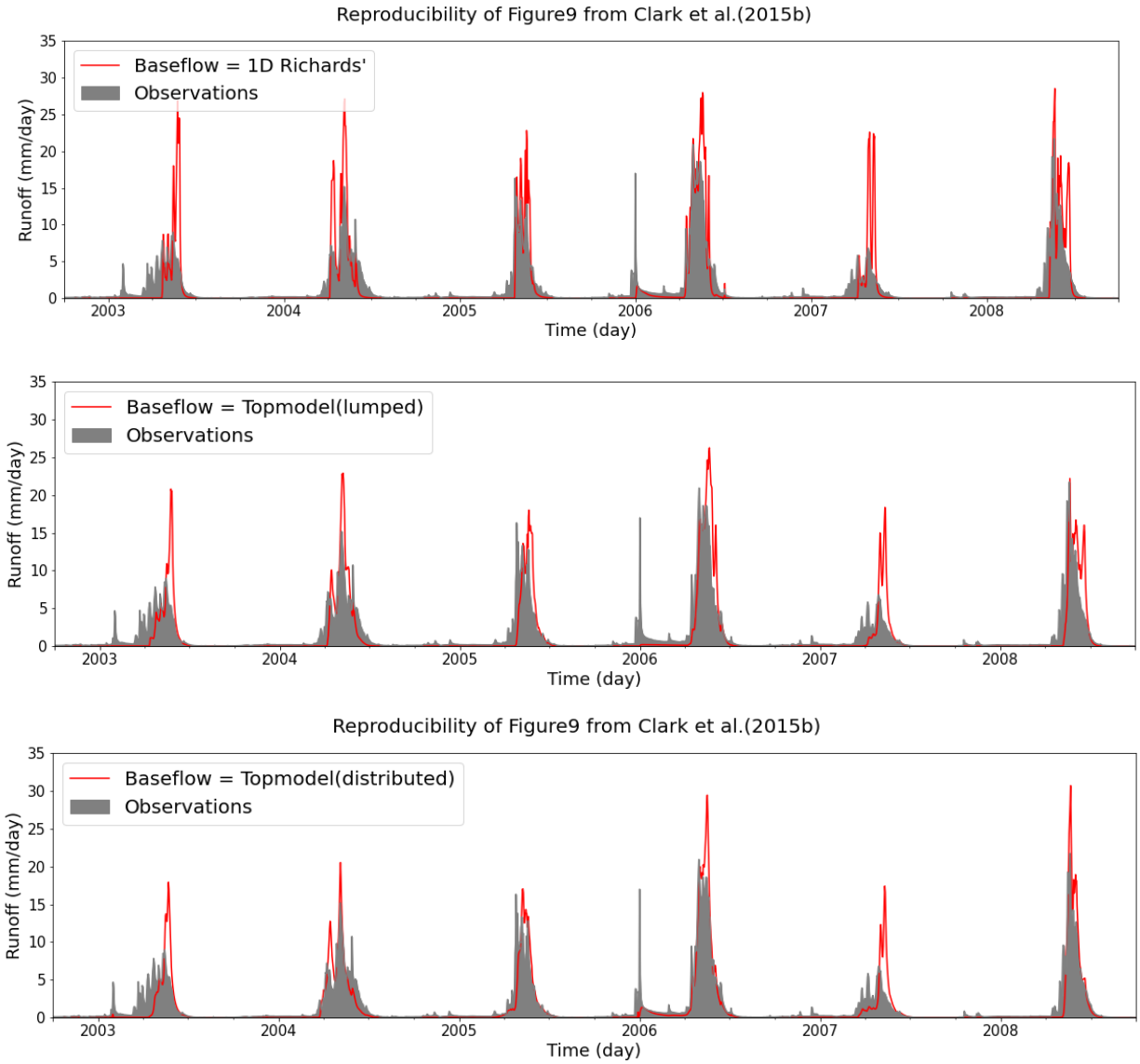

Figure A.2 Reproduced and additional figures of figure 9 from Clark et al. (2015b) showing the impact of the different lateral flux parameterizations on simulations of runoff based on Scenario 3 and 4