Comparing containerization-based approaches for reproducible computational modeling of environmental systems

Young-Don Choi,[a,b] Binata Roy,[a] Jared Nguyen,[c] Raza Ahmad,[d] Iman Maghami,[a] Ayman Nassar,[e] Zhiyu Li,[f] Anthony M. Castronova,[g] Tanu Malik,[d] Shaowen Wang[f], Jonathan L. Goodall[a,*]

[a] Department of Civil and Environmental Engineering, University of Virginia, Charlottesville, Virginia, USA

[b] AI research Laboratory, R&D Management Department, K-water Research Institute, South Korea

[c] Department of Computer Science, University of Virginia, Charlottesville, Virginia, USA

[d] College of Computing and Digital Media, DePaul University, Chicago, IL 60604, USA

[e] Department of Civil and Environmental Engineering, Utah Water Research Laboratory, Utah State University, Logan, Utah, USA

[f] Department of Geography & Geographic Information Science, University of Illinois at Urbana-Champaign, IL, USA

[g] Consortium of Universities for the Advancement of Hydrological Science, Inc, Cambridge, Massachusetts, USA

* To whom correspondence should be addressed (E-mail: goodall@virginia.edu; Address: University of Virginia, Department of Engineering System and Environment, University of Virginia, 151 Engineers Way, P.O. Box 400747, Charlottesville, VA, 22904, USA; Tel: (434) 243-5019

Choi, Y.D., Roy, B., Nguyen, J., Ahmad, R., Maghami, I., Nassar, A., Li, Z., Castronova, A.M., Malik, T., Wang, S. and Goodall, J.L., 2023. Comparing containerization-based approaches for reproducible computational modeling of environmental systems. Environmental Modelling & Software, p.105760

Abstract

Creating online data repositories that follow findable, accessible, interoperable, and reusable (FAIR) principles has been a significant focus in the research community to address the reproducibility crisis facing many computational fields, including environmental modeling. However, less work has focused on another reproducibility challenge: capturing modeling software and computational environments needed to reproduce complex modeling workflows. Containerization technology offers an opportunity to address this need, and there are a growing number of strategies being put forth that leverage containerization to improve the reproducibility of environmental modeling. This research compares ten such approaches using a hydrologic model application as a case study. For each approach, we use both quantitative and qualitative metrics for comparing the different strategies. Based on the results, we discuss challenges and opportunities for containerization in environmental modelling and recommend best practices across both research and educational use cases for when and how to apply the different containerization-based strategies.

Keywords: *Container Technology, Cloud Computing, Reproducibility, Cyberinfrastructure, Hydrologic Modeling, Jupyter Notebooks*

1. Introduction

The rapid advancement of computing offers both opportunities and challenges for reproducibility in computational research (de Lusignan & van Weel, 2006). On the one hand, new tools and technologies have made possible complex physical modeling (Kerandi et al., 2018), deep learning (Shen, 2018), and interdisciplinary modeling (Laniak et al., 2013; Vogel et al., 2015). Additionally, with the possible exception of non-deterministic modeling approaches that rely on unique random seeds, there is some level of confidence that if the same input data and model software are executed on 'identical machine', it will result in the same output, even when the modeling software is very complicated (Sacks et al., 1989). On the other hand, creating "identical machines" including both hardware and software on a machine is very difficult in practice. When these computational models are moved to a new machine, modelers often experience difficulties reproducing the same model results (Baker, 2016; Essawy et al., 2020; Hothorn & Leisch, 2011; Wilson et al., 2017). This is because the way software is packaged, installed, and executed on specific hardware to create 'identical machines' is often very complicated and difficult, even when these steps are well documented (Garijo et al., 2013).

The rapid evolution of software versions is one key factor that makes computational reproducibility so challenging (Epskamp, 2019; Yuan et al., 2018), especially open-source software commonly used in many scientific communities. Slight differences in the computational environment, including but not limited to software dependencies, can result in unexpected errors in re-executing models (Stagge et al., 2019) and can significantly influence the model 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, in fact, required for reproducible work: sharing computational environments and automated workflows (Beaulieu-Jones & Greene, 2017; Chuah et al., 2020; Essawy et al., 2020; Kim et al., 2018).

To overcome what might be called the "reproducibility gap," researchers have presented not only high-level guidelines and principles (Choi et al., 2021; Essawy et al., 2020; Gil et al., 2016; Wilkinson et al., 2016) but also developed various software tools for specific tasks required for computational reproducibility (Kurtzer et al., 2017; Merkel, 2014; That et al., 2017). For example, while online repositories that follow FAIR (Findable, Accessible, Interoperable, Reusable) guiding principles (Wilkinson et al., 2016) continue to mature, it has led not only to a growing demand for sharing well-documented data, source code, software, and workflows, but also with software for automatically encapsulating computational environments and workflows using containerization and literate programming (Kery et al., 2018; Knuth, 1984).

These new tools and concepts open the door to applying FAIR guiding principles that are inclusive of not just data but also modeling software. For example, Bast (2019) suggested that source code management and containerization tools are needed to reproduce the computational environments that underly computational models, while Goble et al. (2020) suggested the FAIR principles are required for end-to-end workflows to describe the execution of a computational process such as data collection, data preparation, data analysis, and data visualization. Researchers are beginning to create the cyberinfrastructure needed for such approaches. Reproducibility of computational environments and automated workflows have been shown to be critical to filling the computational reproducibility gaps in practice (Piccolo & Frampton, 2016; Rosenberg et al., 2020; Sandve et al., 2013). In hydrology, Hutton et al. (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).

To discuss reproducible computational modeling, it is first important to define and agree to the main software components used in environmental modeling studies. In this paper, we consider three main software components: 1) the core model software (i.e., what could be thought of as the model engine), 2) secondary software needed to support the modeling application (i.e., software dependencies and support tools), and 3) modeling workflows that capture the end-to-end modeling application (i.e., from data preparation to analysis and visualization of the model output). The core model software is often developed using a compiled programming language and is optimized for computational performance. The secondary software needed to support the modeling application might include a Graphical User Interface (GUI) or an Application Programming Interface (API) for creating and analyzing input and output files associated with the core model software. Many such model APIs now exist for different environmental models (Choi et al., 2021; Lampert & Wu, 2015; B. McDonnell et al., 2020; Volk & Turner, 2019) and offer a powerful way of programmatically creating and interacting with the so-called model instances (Morsy et al., 2017). Finally, modeling workflows are important to capture the entire end-to-end process, using model APIs and scripting to link the entire end-to-end process from raw data to publication-ready figures.

Despite progress in understanding and creating more reproducible modeling studies across fields, many challenges remain (Reinecke et al., 2022). We argue that a significant reason for these remaining challenges is the required level of human expertise to install and configure complicated computational modeling setups. Outside of a few well-maintained and often commercial or government-backed organizations, many model developers might understand the specific requirements for reproducing their environmental models on other computers, but documenting this procedure with enough detail for others to follow can be challenging. The challenge resulting from the increased complexity of software systems is not unique to environmental modeling or even scientific modeling more generally; it is, in fact, a common problem in software engineering. To address this challenge, computer scientists and software engineers have developed sophisticated containerization tools to encapsulate complex software as a virtual machine or environment (Bentaleb et al., 2022). Therefore, this paper aims to compare various local and remote computational approaches to advance reproducible environmental modeling (Hut et al., 2017; Choi et al., 2021).

While these containerization tools offer an important opportunity, it can be challenging for environmental modelers to know how best to utilize them for different modeling use cases and applications. Many containerization approaches exist, and the options for using these approaches across different computational environments (e.g., the researcher's personal computer to remote cloud-computing environments) make the advantages and disadvantages of leveraging containerization difficult to discern. Thus, the goal of this paper is to compare different local and remote computational approaches for advancing reproducible environmental modeling. We evaluate ten approaches, in total, using a hydrologic modeling case study leveraging the Structure for Unifying Multiple Modeling Alternative (SUMMA) (Clark et al., 2015a) modeling framework and a set of quantitative and qualitative metrics. We discuss the benefits and weaknesses of each approach and summarize recommended best practices for using the approaches to achieve different modeling objectives. Finally, we discuss remaining knowledge gaps in creating reproducible computational models that require future research and development.

2. Methodology

2.1 The Computational Reproducibility Approaches

The ten computational reproducibility approaches considered in this study are given in Table 1. The approaches are first categorized as using primarily local or remote computational resources for execution. For the local approaches, a virtual machine (VM) in Virtual Box with specifications typical of a personal computer was used for model execution. For the remote approaches, each approach leveraged JupyterHub but was implemented using different cyberinfrastructures: Consortium of Universities for the Advancement of Hydrologic Science, Inc. (CUAHSI) JupyterHub, CyberGIS-Jupyter for Water (Yin et al., 2019), and BinderHub to build a Jupyter instance from a code repository.

Table 1. Approaches for computational reproducibility through containerization considered in the study.

Approach No.	**Local and Remote Computational Environments**		**Combination of Software Tools and Modeling Workflows**		
			1) Core Model Software	**2) Secondary Software**	**3) Modeling Workflow**
1	**LOCAL**	Virtual Box	GNU Make	Conda Virtual Environment	Jupyter Notebook
2			Docker		
3			Docker		
4			Singularity		
5			Sciunit		
6	**REMOTE**	CUAHSI JupyterHub	Docker		Jupyter Notebook
7		CyberGIS-Jupyter for Water	Docker		
8		CUAHSI JupyterHub	Sciunit		
9		CyberGIS-Jupyter for Water	Sciunit		
10		Binder	Docker		Jupyter Notebook

The second categorization is based on what components of the end-to-end modeling workflow are containerized and using which containerization technology. As stated earlier, we consider a computational model consisting of three primary software components: 1) the core model software, 2) secondary software, and 3) a modeling workflow. Approaches for computational reproducibility through containerization may address one or more of these components. Likewise, different containerization technologies exist including Docker, Singularity, and Sciunit. We did not test all combinations of component containerization using different technologies, but rather we focused on logical combinations, which based on our judgement, would most likely be used by environmental modelers. Finally, in the first two cases we used GNU Make and a Python-based Virtual Environment (Conda VE) to include approaches commonly used by modelers for comparison purposes. While not directly using containerization, these approaches represent ways for achieving more portable and reproducible environmental modeling applications and represent a meaningful base case for comparison.

The first five approaches in Table 1 all leverage local computing resources, which means a VM on the modeler's own workstation for running the end-to-end modeling workflow. Approach 1 represents a standard approach commonly used by modelers (Peckham et al., 2013) in that the model software is compiled using GNU Make and the secondary software, written in Python, is encapsulated in a Conda VE. Approach 2 introduces formal containerization tools rather than only encapsulation, but only for the core model software component using Docker as the containerization solution. Approach 3 builds on Approach 2 by using containerization for not only the core model software, but also the secondary software supporting the model, again using Docker as the containerization tool. Approach 4 further builds on Approach 3 by keeping the same containerization strategy but switching the containerization tool from Docker to Singularity. Finally, Approach 5 also builds from Approach 3 but uses Sciunit rather than Docker or Singularity as the containerization tool. Thus, across these five approaches, we begin from a standard approach without direct use of containerization technologies and build to an end-to-end modeling workflow leveraging three containerization technologies: Docker, Singularity, and Sciunit.

Approaches 6-10 make use of remote computational resources to compute the same end-to-end modeling workflow. Approach 6 uses the CUAHSI JupyterHub (hereafter CJH), a cloud computing environment on the Google Cloud Platform specifically designed to support research and education in the water sciences. Approach 7 uses the CyberGIS-Jupyter for Water platform (hereafter CJW), a CyberGIS-Jupyter instance tailored to support data-intensive and reproducible research in the environmental modeling community built on the Jetstream computational resource (Yin et al., 2018). In both approaches, Docker is used as a containerization technology. Approaches 8 and 9 again use CJH and CJW, respectively, but with Sciunit in place of Docker as the containerization tool. Singularity is not typically used in JupyterHub environments (Prasad et al., 2020), so it was not considered for these approaches. Approach 10 uses a containerization approach called Binder that allows users to create a custom JupyterHub instance from a code repository using Docker as the containerization technology (Jupyter Project et al., 2018). Further detail about the specific procedures and characteristics of each approach is presented in the following subsections.

2.1.1 Local Approaches

For the five local approaches, we used Virtual Box to create a consistent Linux virtual environment (Ubuntu 20.04 LDT) with a Windows operating system and a single-core processor (Table 2). We considered this to be a typical personal computer used by modelers, although we acknowledge many modelers would have access to workstations with higher end computational and memory resources.

Table 2. Specification of the base local computational environment

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

Figure 1 shows the steps required to complete the five local approaches from the perspective of a developer, that is the person setting up the modeling workflow, and the user, that is the person executing the workflow for a given input dataset. These steps are used to evaluate each approach across a set of metrics which are described later in the paper.

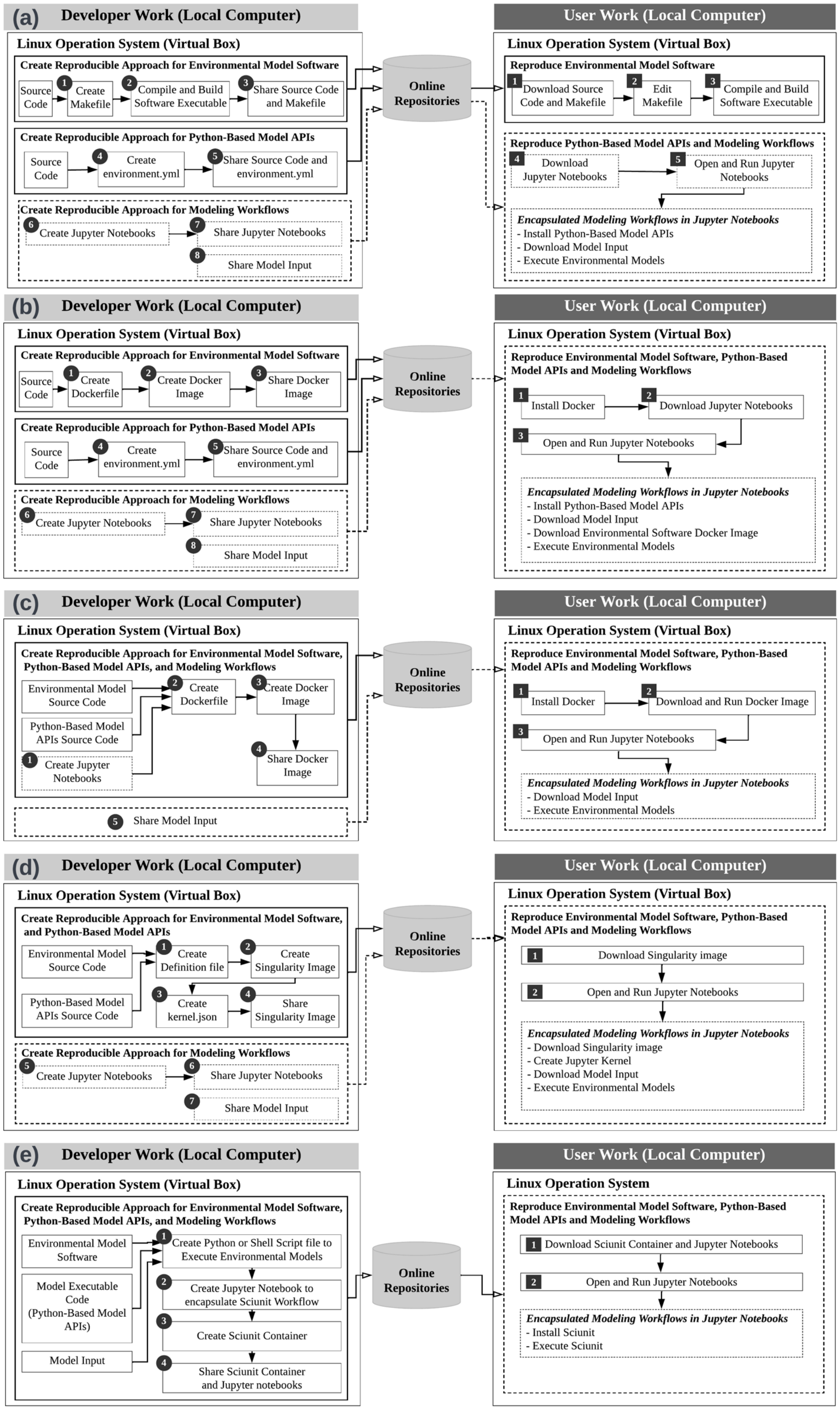

(a)
Developer Work (Local Computer)
Linux Operation System (Virtual Box)
Create Reproducible Approach for Environmental Model Software
Source Code
1 Create Makefile
2 Compile and Build Software Executable
3 Share Source Code and Makefile
Create Reproducible Approach for Python-Based Model APIs
Source Code
4 Create environment.yml
5 Share Source Code and environment.yml
Create Reproducible Approach for Modeling Workflows
6 Create Jupyter Notebooks
7 Share Jupyter Notebooks
8 Share Model Input
Online Repositories
User Work (Local Computer)
Linux Operation System (Virtual Box)
Reproduce Environmental Model Software
1 Download Source Code and Makefile
2 Edit Makefile
3 Compile and Build Software Executable
Reproduce Python-Based Model APIs and Modeling Workflows
4 Download Jupyter Notebooks
5 Open and Run Jupyter Notebooks
Encapsulated Modeling Workflows in Jupyter Notebooks
- Install Python-Based Model APIs
- Download Model Input
- Execute Environmental Models
(b)
Developer Work (Local Computer)
Linux Operation System (Virtual Box)
Create Reproducible Approach for Environmental Model Software
Source Code
1 Create Dockerfile
2 Create Docker Image
3 Share Docker Image
Create Reproducible Approach for Python-Based Model APIs
Source Code
4 Create environment.yml
5 Share Source Code and environment.yml
Create Reproducible Approach for Modeling Workflows
6 Create Jupyter Notebooks
7 Share Jupyter Notebooks
8 Share Model Input
Online Repositories
User Work (Local Computer)
Linux Operation System (Virtual Box)
Reproduce Environmental Model Software, Python-Based Model APIs and Modeling Workflows
1 Install Docker
2 Download Jupyter Notebooks
3 Open and Run Jupyter Notebooks
Encapsulated Modeling Workflows in Jupyter Notebooks
- Install Python-Based Model APIs
- Download Model Input
- Download Environmental Software Docker Image
- Execute Environmental Models
(c)
Developer Work (Local Computer)
Linux Operation System (Virtual Box)
Create Reproducible Approach for Environmental Model Software, Python-Based Model APIs, and Modeling Workflows
Environmental Model Source Code
Python-Based Model APIs Source Code
1 Create Jupyter Notebooks
2 Create Dockerfile
3 Create Docker Image
4 Share Docker Image
5 Share Model Input
Online Repositories
User Work (Local Computer)
Linux Operation System (Virtual Box)
Reproduce Environmental Model Software, Python-Based Model APIs and Modeling Workflows
1 Install Docker
2 Download and Run Docker Image
3 Open and Run Jupyter Notebooks
Encapsulated Modeling Workflows in Jupyter Notebooks
- Download Model Input
- Execute Environmental Models
(d)
Developer Work (Local Computer)
Linux Operation System (Virtual Box)
Create Reproducible Approach for Environmental Model Software, and Python-Based Model APIs
Environmental Model Source Code
Python-Based Model APIs Source Code
1 Create Definition file
2 Create Singularity Image
3 Create kernel.json
4 Share Singularity Image
Create Reproducible Approach for Modeling Workflows
5 Create Jupyter Notebooks
6 Share Jupyter Notebooks
7 Share Model Input
Online Repositories
User Work (Local Computer)
Linux Operation System (Virtual Box)
Reproduce Environmental Model Software, Python-Based Model APIs and Modeling Workflows
1 Download Singularity image
2 Open and Run Jupyter Notebooks
Encapsulated Modeling Workflows in Jupyter Notebooks
- Download Singularity image
- Create Jupyter Kernel
- Download Model Input
- Execute Environmental Models
(e)
Developer Work (Local Computer)
Linux Operation System (Virtual Box)
Create Reproducible Approach for Environmental Model Software, Python-Based Model APIs, and Modeling Workflows
Environmental Model Software
Model Executable Code (Python-Based Model APIs)
Model Input
1 Create Python or Shell Script file to Execute Environmental Models
2 Create Jupyter Notebook to encapsulate Sciunit Workflow
3 Create Sciunit Container
4 Share Sciunit Container and Jupyter notebooks
Online Repositories
User Work (Local Computer)
Linux Operation System
Reproduce Environmental Model Software, Python-Based Model APIs and Modeling Workflows
1 Download Sciunit Container and Jupyter Notebooks
2 Open and Run Jupyter Notebooks
Encapsulated Modeling Workflows in Jupyter Notebooks
- Install Sciunit
- Execute Sciunit

Figure 1: The steps required for the five local approaches from the developer and user perspectives.

As shown in Figure 1a, to set up Approach 1 the developer must complete the following steps: 1) create a Makefile; 2) compile and build the core software executable; and 3) share the source code and Makefile on an online repository such as GitHub or HydroShare. Next, the Conda VE is created to support the secondary software for the modeling workflow. Finally, the Jupyter notebooks that document the end-to-end modeling steps are created and shared. Once the end-to-end model workflow has been captured, the developer's job is complete, and a user can reproduce the modeling study. The steps the user must take to execute Approach 1 are 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. Once these steps are complete, the user must download the Jupyter notebooks that document, the end-to-end workflow, including installing the required software, downloading model input data, and executing the environmental model. Compared to the developer work, the user work is simpler because the Jupyter notebooks document the workflow, and the user's task is focused mainly on compiling the core model software and installing secondary software.

Figure 1b shows the procedure for Approach 2 where the developer must 1) create a Dockerfile, which has instructions to download and build software, 2) create a Docker image from the Dockerfile, and 3) share the Docker image on an online repository such as the DockerHub. This process is often not a linear sequence of steps, but an interactive process where creating the Docker image is time-consuming involving testing and verification before the Docker image is finally shared. Once this process is complete, however, the user only needs to install Docker using the simple command "*sudo apt install docker.io*" to get the core model operating correctly. The user must still obtain and run the Jupyter notebooks representing the end-to-end workflow, including installing required secondary software and input files, before executing the model.

For Approach 3 (Figure 1c), the developer’s first step is creating Jupyter notebooks to containerize workflows into a Docker container. Next, the developer must create a Dockerfile that includes the commands needed to containerize the core environmental model software, Python-based model API, and modeling workflows. In this approach, users only need to install Docker and run the Docker image because the Docker image has the required dependencies. Then user can open and run the Jupyter notebooks to reproduce the end-to-end workflow.

In Approach 4 (Figure 1d), the developer will first create a Definition file to create a Singularity image that includes a dependency list. Next, the developer must make a “kernel.json” file to link a Jupyter kernel with the Singularity image and Jupyter notebooks. Next, the developer can share the Singularity image through online repositories including Singularity Hub. Developers must also create and share Jupyter notebooks and the model input for the modeling workflows. After the developer’s work is complete, the user needs to download the Jupyter notebooks first, then open and run the Jupyter notebooks. The Jupyter notebooks handle the rest of the workflow including downloading the Singularity image of the core environmental model software, creating the Jupyter kernel to establish a link between the Singularity image and Jupyter notebooks, downloading the model input data, and executing the environmental model.

Finally, in Approach 5 (Figure 1e), the developer first creates a Jupyter notebook to encapsulate workflows using Sciunit (Essawy et al., 2018). Next, the developer creates a Sciunit container using the programming code and the Jupyter notebook. After that step is complete, the developer can share the Sciunit container and the Jupyter notebook. Users then can download the Sciunit container and Jupyter notebook and only need to open and run the corresponding Jupyter notebook. Unlike other approaches, users do not need to download the model input as the Sciunit container includes the model input and all the software dependencies.

2.1.2 Remote Approaches

Figure 2 illustrates the steps required to complete the five remote approaches from the perspective of the developer and the user. As with the five local approaches, these steps are used to evaluate each approach across a set of metrics that are described later in the paper.

As Figure 2a shows, for Approaches 6 and 7 the developer must create a Dockerfile, similar to Approaches 2 and 3. The user may use GitHub to add a new Dockerfile as a pull request to the CJH or CJW GitHub repository. 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 the modeling environments are preconfigured and shared through the environmental profiles of CJH or Jupyter kernels of CJW. Figure 2b shows the general procedure of Approaches 8 and 9 that follow the same steps as Approach 5 (Figure 1e) using Sciunit, so they are not explained further here.

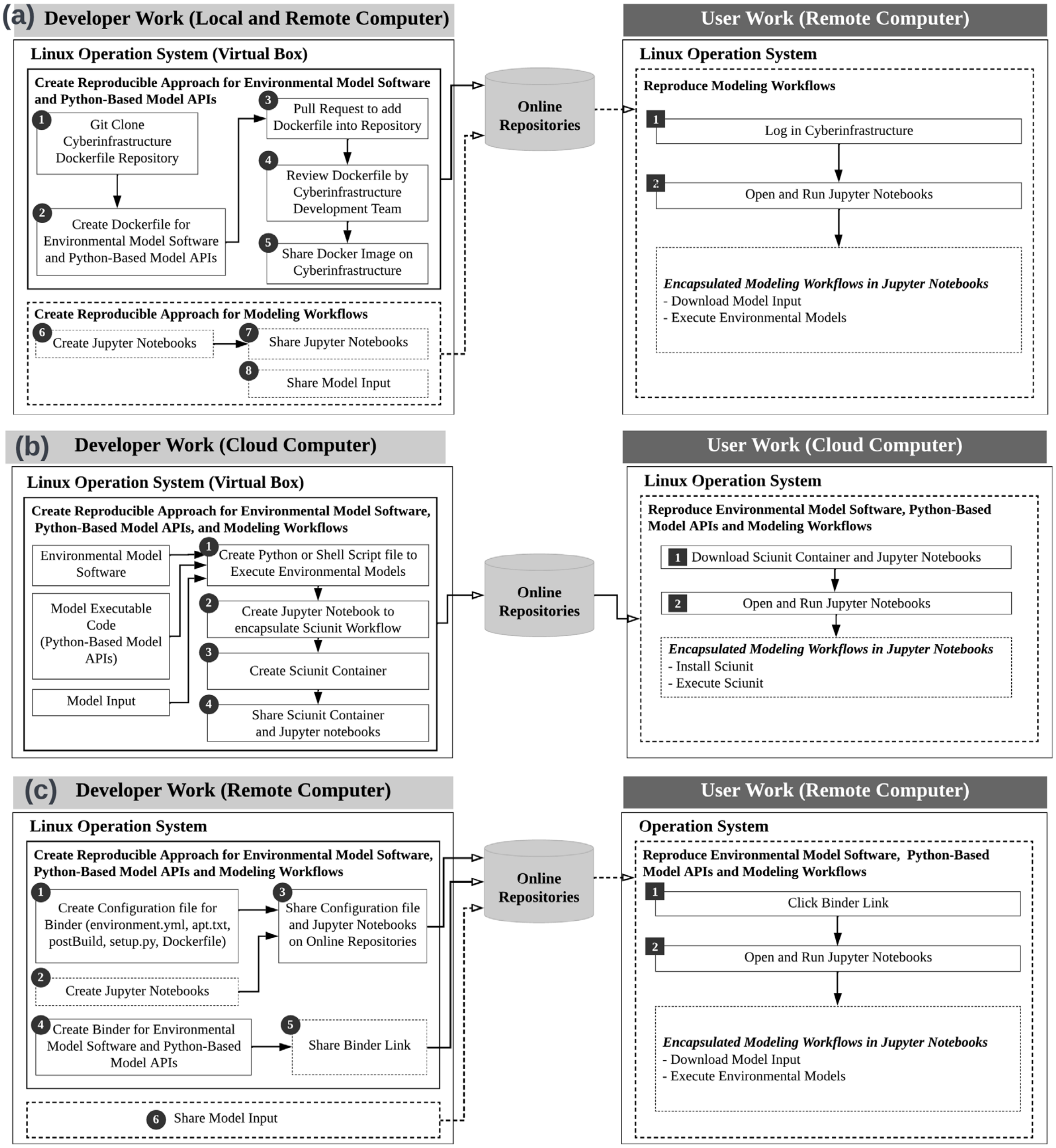

(a) Developer Work (Local and Remote Computer)
Linux Operation System (Virtual Box)
Create Reproducible Approach for Environmental Model Software and Python-Based Model APIs
1 Git Clone Cyberinfrastructure Dockerfile Repository
2 Create Dockerfile for Environmental Model Software and Python-Based Model APIs
3 Pull Request to add Dockerfile into Repository
4 Review Dockerfile by Cyberinfrastructure Development Team
5 Share Docker Image on Cyberinfrastructure
Create Reproducible Approach for Modeling Workflows
6 Create Jupyter Notebooks
7 Share Jupyter Notebooks
8 Share Model Input
Online Repositories
User Work (Remote Computer)
Linux Operation System
Reproduce Modeling Workflows
1 Log in Cyberinfrastructure
2 Open and Run Jupyter Notebooks
Encapsulated Modeling Workflows in Jupyter Notebooks
- Download Model Input
- Execute Environmental Models
(b) Developer Work (Cloud Computer)
Linux Operation System (Virtual Box)
Create Reproducible Approach for Environmental Model Software, Python-Based Model APIs, and Modeling Workflows
Environmental Model Software
Model Executable Code (Python-Based Model APIs)
Model Input
1 Create Python or Shell Script file to Execute Environmental Models
2 Create Jupyter Notebook to encapsulate Sciunit Workflow
3 Create Sciunit Container
4 Share Sciunit Container and Jupyter notebooks
Online Repositories
User Work (Cloud Computer)
Linux Operation System
Reproduce Environmental Model Software, Python-Based Model APIs and Modeling Workflows
1 Download Sciunit Container and Jupyter Notebooks
2 Open and Run Jupyter Notebooks
Encapsulated Modeling Workflows in Jupyter Notebooks
- Install Sciunit
- Execute Sciunit
(c) Developer Work (Remote Computer)
Linux Operation System
Create Reproducible Approach for Environmental Model Software, Python-Based Model APIs and Modeling Workflows
1 Create Configuration file for Binder (environment.yml, apt.txt, postBuild, setup.py, Dockerfile)
2 Create Jupyter Notebooks
3 Share Configuration file and Jupyter Notebooks on Online Repositories
4 Create Binder for Environmental Model Software and Python-Based Model APIs
5 Share Binder Link
6 Share Model Input
Online Repositories
User Work (Remote Computer)
Operation System
Reproduce Environmental Model Software, Python-Based Model APIs and Modeling Workflows
1 Click Binder Link
2 Open and Run Jupyter Notebooks
Encapsulated Modeling Workflows in Jupyter Notebooks
- Download Model Input
- Execute Environmental Models

Figure 2. The steps required for the five remote approaches from the developer and user perspectives.

Figure 2c shows the general procedure of Approach 10. 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 workflow. Then, the developer shares the configuration files and the Jupyter notebooks through an online repository such as GitHub, Figshare, Zenodo, or HydroShare. After that, the developer uses MyBinder 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.

2.2 Evaluation of the Approaches

We evaluated the ten approaches (five local and five remote) against a set of quantitative and qualitative metrics using a hydrologic modeling study as an example application. In this example application, we used the SUMMA (Clark et al., 2015a) hydrologic model as the core model software, pySUMMA (Choi et al., 2021) and other Python packages as the secondary software, and Jupyter notebooks to orchestrate the end-to-end modeling workflow. These three components are described in further detail in the following subsection. We then describe the quantitative and qualitative criteria used to evaluate the ten approaches.

2.2.1 Modeling Application used for the Evaluation

SUMMA was selected for the evaluation because it represents a typical numerical computational model used in environmental studies. It is, in fact, more of a modeling framework since it 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 Fortran, and we used the Fortran compiler ‘gfortran’ to compile the 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 file format used by SUMMA. 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 be compiled. 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 compile SUMMA but convenient for working with SUMMA input and output files, includes pySUMMA, a Python-based SUMMA model API. pySUMMA allows 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 its pySUMMA GitHub repository 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 install the pySUMMA environment with dependencies on a new machine.

Finally, for modeling workflows, we used Jupyter notebooks to create modeling workflows through a mix of formatted text, mathematical equations, and executable code with in-line visualizations. We created Jupyter notebooks for each of the ten reproducible approaches described earlier to encapsulate reproducible artifacts and modeling workflows. These notebooks are available as products of this research as described in the Data and Software Availability section of this paper.

We used hydrologic modeling experiments described in Clark et al. (2015b) in our evaluation. Based on these experiments, we created four scenarios (Table 3) using two datasets to reproduce Figures 7, 8, and 9 in Clark et al. (2015b). The first scenario is a single simulation for 15 months using the S*imple Resistance* method, as the stomatal resistance parameterization in SUMMA. The second scenario includes nine ensemble simulations for analyzing the impact on ET using 1) three different stomatal resistance parameterizations, *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. The first and second scenarios aim to reproduce Figures 7 and 8 in Clark et al. (2015b) (included here as Figure A.1). The third scenario is a single simulation for 75 months to analyze the impact of using the *1d Richards* method (Celia et al., 1990), which is one of the lateral flow parameterizations in SUMMA, on runoff. The fourth and final scenario is three ensemble simulations to analyze the impact of using three different lateral flow parameterizations: *1d Richards, Lumped Topmodel,* and *Distributed Topmodel* (Duan & Miller, 1997) on runoff. From the third and fourth scenarios, our aim is to reproduce Figure 9 in Clark et al. (2015b) (included here as Figure A.2).

Table 3. SUMMA simulation scenarios for evaluating the ten reproducible approaches.

Scenario	Descriptions
(a) Scenario 1	□ A single simulation (simple resistance method) □ Simulation periods: 2006-07-01 ~ 2007-09-30 (15 months)
(b) 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)
(c) 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)

2.2.2 Quantitative Performance Metrics

The following quantitative measures were used to evaluate the ten approaches. 1) Competency considers the level of effort in reproducing each step in the approach 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 the storage requirements for storing and sharing each approach, another important factor in the adoption of reproducible approaches (Craig & Victoria., 2020; Kovács, 2017). 3) The computational time measures the wall time required to execute the approach, which can vary significantly across approaches and impact the usability of the approach (Kozhirbayev & Sinnott, 2017).

For the competency metric, we evaluated the level of skill required to complete each step of the approach from both the developer and user perspectives. We defined three levels: Minimal, Moderate, and Substantial. Minimal means basic skills are required including downloading, setting up, and running the code without any changes in the basic workflow. Moderate means additional skills are needed including editing and creating simple codes in the existing workflow. Finally, Substantial means requiring expertise in coding and re-configuring the existing workflow.

In order to given numerical scores to these categories, we scored 'Minimal Skill' as an integer between 1 and 3, 'Moderate Skill' between 4 and 6, and 'Substantial Skill' between 7 and 9. As this was done for each step in an approach and an overall 'total score' for the approach was calculated as simply the sum of all steps in that approach. Since this scoring can be subjective, we had six experts, all co-authors of this paper and knowledgeable of the modeling steps as both users and developers, complete the evaluation independently and report the range of scores in the results section. The Appendix (Table A.1 – A.7) includes the questionaries used to obtain the competency scores for the ten approaches.

For the size metric, we measured how much space is used to store all digital artifacts associated with the reproducible approach. We only considered the size metric for the five local approaches and not the five remote approaches because the size of the preconfigured computational artifacts in a remote environment will be determined by the specific technical implementation in that remote environment and will have less impact on the end user. Finally, for the computational time metric, we measured the execution time across all ten reproducible approaches. In this performance metric, we measured the wall time required to run the end-to-end workflow for the approach.

2.2.3 Qualitative Performance Metrics

In terms of qualitative performance metrics, we first describe the strengths and weaknesses of each approach through our experience implementing each approach from both the developer and user perspectives. We then considered two broad use cases for environmental models: 1) education and 2) research. Based on the strengths and weaknesses and with these two use cases in mind, we present recommendations for best practices when using each of the ten approaches.

3. Results

3.1 Quantitative Evaluation

3.1.1 Required Competency

The resulting competency metric scores, grouped by developer and user work, are shown in Figure 3. The boxplot depicts the range of the scores across the experts who rated the competency needed to complete each step of each approach. The total score for developer work was consistently higher than the user work, indicating that the developer work requires greater competency or effort than the user work. This is expected as the competency was defined around coding and computing skills rather than modeling skills. Interestingly, there was less variability in scores when evaluating the user’s work compared to the developer’s work, meaning there was more agreement among those who completed the evaluation about the competency required for the user steps. To help visualize the results for each approach, the median scores are depicted in a spider plot (Figure 4) to show how each approach ranks across the developer and user competency metric scores.

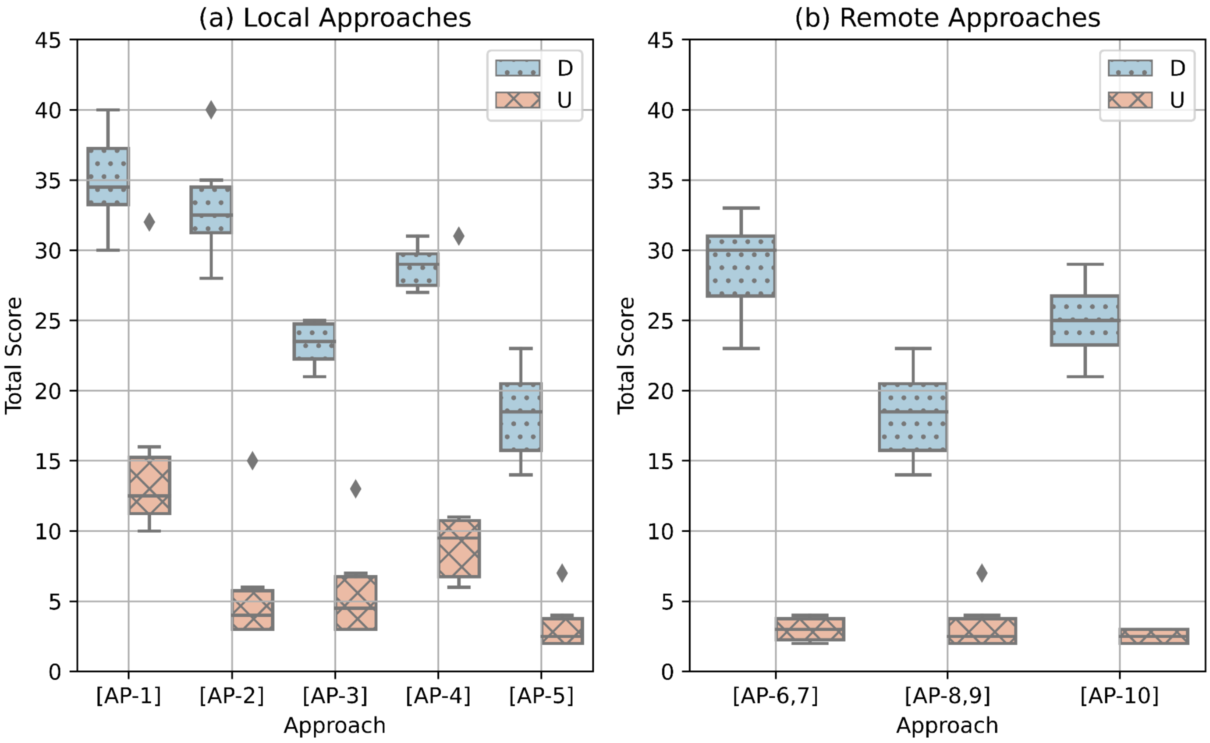


Figure 3: Competency metric scores for each approach for the steps completed by the Developer (D) and User (U). The box plots represent the range of scores across the six individuals who rated the approach.

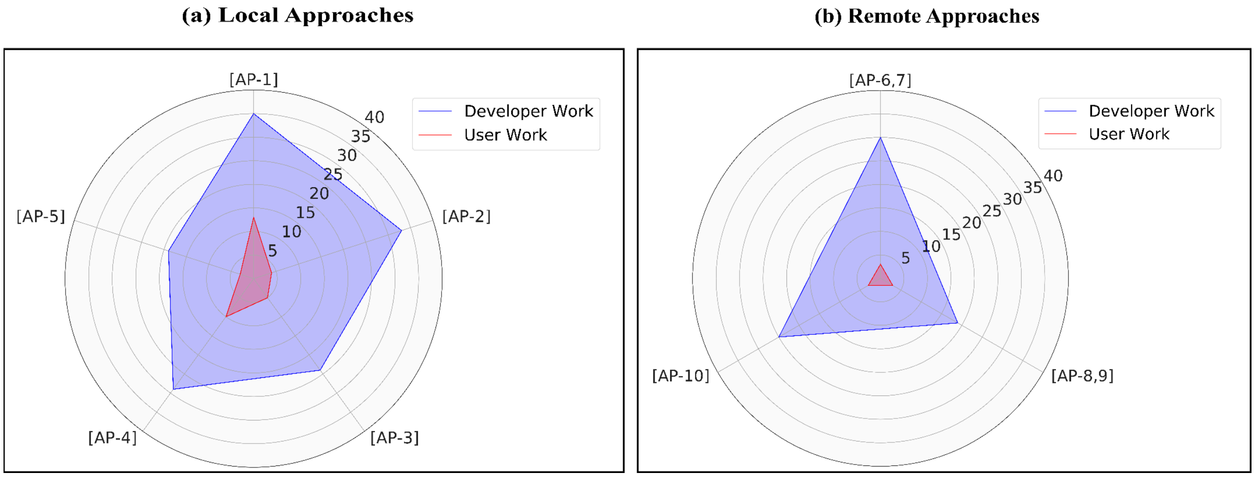


Figure 4: The median competency metric scores for (a) local and (b) remote reproducible approaches for the developer and user work.

In terms of developer work, Approach 1 (Score = 35) was scored as the most complicated approach among the local approaches. In this approach, the developer needs to reproduce every step individually as it does not use containerization technology. In terms of user work, most approaches have a low score compared to Approach 1. Approach 5 was scored as the simplest approach from both the developer (Score = 19) and user perspectives (Score = 3). For the remote approaches and in terms of developer work, approaches 6 and 7 were judged to require a high level of competency. Because Dockerfile must be programmed, it requires a considerable amount of knowledge about the Docker platform and its API, it can be a complex task for the model developers. Furthermore, for cloud environments like CJH and CJW, developers themselves cannot install new models until they are reviewed by the larger CJH and CJW development teams. Approaches 8 and 9 were judged to be the simplest approaches considering developer (Score = 19) and user work (Score = 3). These approaches used Sciunit which can containerize modeling environment and workflow into a container by recording steps in the model execution code created with no additional work. This allows users to easily reproduce published results using Sciunit containers and commands in a Jupyter notebook. Approaches 6, 7, and 10 are the simpler in part because dependencies for the environmental modeling code are preconfigured into containers.

3.1.2 Size of Reproducible Artifacts

Figure 5 shows the sizes of the digital artifacts for the five local reproducible approaches. Approach 5 is the most lightweight and it is ten times smaller than Approach 4, which is the second most lightweight. This is because Sciunit only encapsulates dependencies that are used during modeling workflows, compared to other containerization tools that containerize additional software and Python libraries that may be stated for example in a Dockerfile but not directly used in the workflow. Sciunit further uses content-based deduplication to determine redundant file blocks across files used in a workflow (That et al., 2017; Yuan et al., 2018). In addition, Approach 4 is lighter weight than Approaches 1-3 because Singularity utilizes a flatter structure for files in an image, meaning all files of an image are combined into a single image format and compressed. In contrast, Docker uses a concept of layering of files in which files are shared across multiple images. The layered file system on disk, however, is not compressed as in Singularity and thus the result in Figure 5. We note that currently we have a single model run and layering does not offer much space saving but can do so if the developer is using multiple containers. We also anticipate that in that case a compressed file system will offer more savings than layering and the overall result trend will be the same. This concept used in Docker is not helpful for a single model software run, but it will help when researchers want to use multiple commands with layered images. Finally, approach 1 and 2 do not use container tools and thus do not take advantage of file system saving methods such as compression and deduplication. The sizes required for each of the dependencies are given in detail in Table A.8.

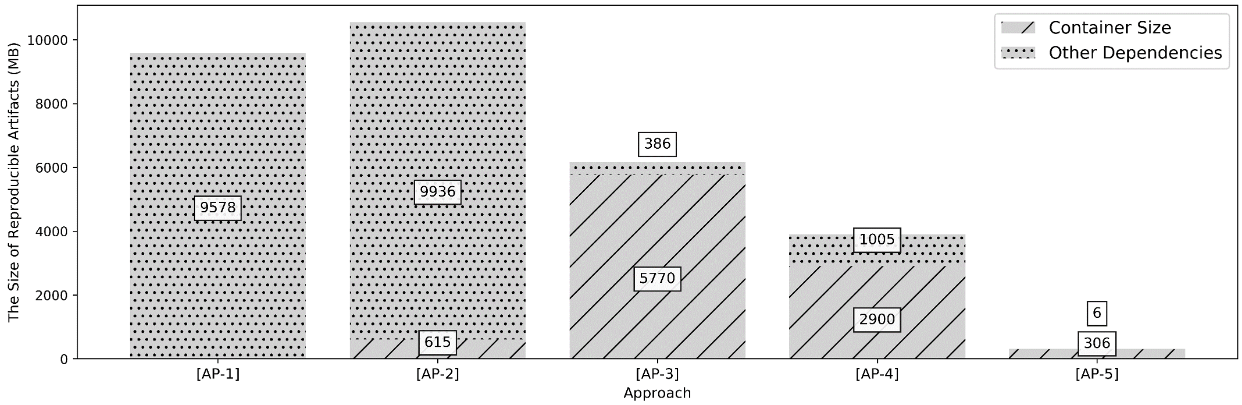
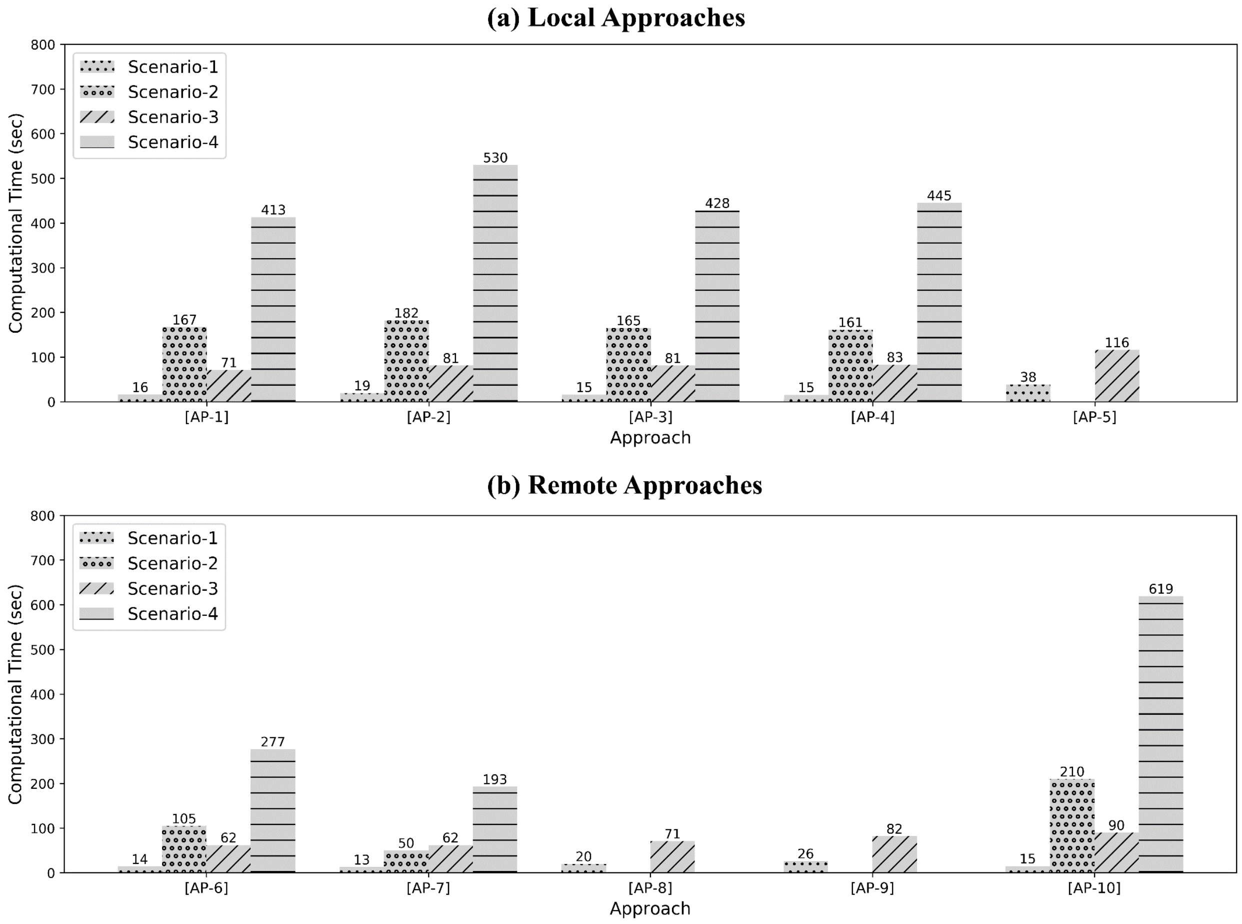


Figure 5. Comparison of the size for reproducible artifacts for the local reproducible approaches.

3.1.3 Workflow Runtime

Figure 6 shows the results of the workflow runtime comparison for the approaches using (a) local and (b) remote compute resources across the four modeling scenarios described in Table 3. When we compare the five local approaches; it shows that Approach 5 is slightly slower than the other approaches. However, the overall computing time is similar across the five local reproducible approaches. For the remote approaches, Approach 7 was the fastest approach even though the approach requires additional time to submit jobs between CJW to distributed HPC resources and retrieve model output from such resources 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 ones. Due to how the model runs were setup using Dask, a Python library for parallel computing (Rocklin, 2015), and because Dask automatically allocates multiple cores for ensemble simulations, the Sciunit encapsulation of the ensemble simulations (Scenarios 2 and 4) were not configured to take advantage of the multiple cores. Hence, the runtime for Scenarios 2 and 4 were excluded from the figure 6(a). From the performance test of computing time, for data-intensive modeling such as the simulation of fully distributed models and Contiguous United States (CONUS) scale models, we can see the value of using remote environments that can access HPC resource.


(a) Local Approaches

(b) Remote Approaches

Figure 6. Comparison of computing time in the (a) local and (b) remote reproducible approaches.

Overall, if we summarize the result of quantitative performance, we find that the local reproducible approaches require more competency in coding and computing skills, more computational time, and more space compared to remote ones. For both local and remote approaches, the developer work requires a remarkably higher level of effort than the user work. If we compare different containerization tools across the local and remote approaches, Docker was the heaviest weight and was judged to require greater competency. On the other hand, Sciunit was the most lightweight and required less competency to use compared to other containerization tools. Finally, Singularity excelled as a containerization approach for parallel computing. It is worth noting that the performance of containerization-based approaches can vary based on the type (e.g., hydrologic model SUMMA vs. atmospheric model Weather Research and Forecasting WRF) and scale of the environmental model (e.g., local vs. global). For different types of models, the challenges lie in model compilation difficulty and the developers' competency. In terms of modeling scale, factors such as model complexity and watershed scale come into play, with the performance of approaches being influenced by the memory size of the computer used.

3.2 Qualitative Evaluation

3.2.1 Strengths and Weaknesses of Approaches

The strengths and weaknesses of the five local reproducible approaches judged through this research are presented in Table 4. For Approach 1 (Table 4a), a strength is that the GNU Make tool is a common method to share model software and GNU Make itself is important because within each containerization tool GNU Make must be run to build the SUMMA executable. However, this approach is still difficult for many model users as it requires a higher level of computational competency. Therefore, having the developer with the skills required to complete this step and then sharing a containerized version of the model software reduces the burden on the model user.

Table 4. Qualitative evaluation of the strengths, weaknesses, and recommended usages for the local approaches.

(a) **Approach 1: Compiling the Core Model Software**

Strengths	□ [D, U] GNU Make itself is important because this tool has to use in 10 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

(b) **Approach 2: Containerizing the Core Model Software with Docker**

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

(c) **Approach 3: Containerizing All Software with Docker**

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

(d) **Approach 4: Containerizing All Software with Singularity**

Strengths	□ [D] Easy to convert Docker images to Singularity images using docker2singularity library □ [U] Lightweight than other reproducible approaches except Sciunit
Weaknesses	□ [U] Niche usage comparing to Docker
Recommended usages	□ [Research] Models requiring HPC

(e) **Approach 5: Containerizing All Software and Workflow with Sciunit**

Strengths	□ [D, U] The simplest complexity for reproducibility in both developer and user perspective □ [U] The most lightweight in ten reproducible approaches □ [D, U] Easy to share Sciunit containers in 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 □ [U] Does not encapsulate automatic allocation of parallel computing such as Dask
Recommended usages	□ [Education] Offline education □ [Research] Reliable reproducibility as Sciunit can containerize all reproducible artifacts into a container without significant memory use

[D] = Developer; [U] = User

Table 4b presents the results of the qualitative performance evaluation for Approach 2. This approach uses Docker to containerize only the core model software; therefore, users can easily reproduce SUMMA using Docker from DockerHub. In addition, users can install and apply new Python libraries as model APIs without any limitations. After downloading the SUMMA Docker image and installing pySUMMA within a Conda VE, users can execute SUMMA using the "*docker*" option in the pySUMMA "*run*" method. Even if users have not downloaded the SUMMA Docker image on the local computer, pySUMMA can automatically download it from DockerHub. However, sometimes when users create the Conda VE, unexpected errors may occur, requiring the user 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 4c presents the results of the qualitative performance evaluation for Approach 3. This approach containerizes every dependency into a Docker image; therefore, the procedure is stable and consistent in that it is unlikely that errors will occur across users. However, there is a limitation when attempting to install new software or dependencies because users must work inside a Docker image, even if users can install new dependencies, they are temporary. Therefore, this approach is helpful for offline education for practicing and reproducing published results on local computers (public or personal computers) but is less well suited for use cases that require the extension of past work.

Table 4d presents the qualitative performance evaluation results for Approach 4. It is lightweight compared to other reproducibility approaches, except for Sciunit. Currently, Singularity is less widely used than Docker, so sometimes researchers themselves need to create Singularity definition files. In this scenario, we recommend researchers try to find a Dockerfile first and then use the docker2singularity library to convert the Dockerfile into a Singularity definition file.

Finally, Table 4e presents the qualitative performance evaluation results for Approach 5. Sciunit has many advantages, such as being the most simple and lightweight of the ten reproducible approaches considered in this study. In addition, Sciunit is efficient in terms of memory use for encapsulating modeling environments, workflows, and data into one container. Due to its easy installation, Sciunit is helpful as an educational setting where instructors can share reproducible computational materials and students are asked to containerize their own analyses. Thus, it is a powerful tool for reliable reproducible research without requiring continuous version control. However, Sciunit is still in active development as a research project and, in our experience for complicated software with the GRASS GIS system, a dependency of the workflows, it was unable to automatically encapsulate the system. In other cases, Sciunit, being efficient in what it encapsulates by monitoring what software is used in a workflow, may exclude related software not directly used but potentially helpful when extending a workflow (e.g., plotting routines helpful to visualize model output but not directly used in the encapsulated workflow). This is most often a benefit, producing a highly optimized container, but requires the modeler to carefully consider and include all software calls that may be useful in later reuse of the container.

Table 5 highlights the strengths and weaknesses for the five remote reproducibility approaches. Table 5a includes the qualitative performance evaluation results for Approaches 6 and 7. These approaches allow users to use preconfigured modeling environments; therefore, users can use environmental models straightforwardly without additional software installation. In addition, CJW supports distributed HPC resources for parallel computing. Also, CJH supports a custom Conda VE to permanently install Python or other libraries, like Approach 2. However, there is a limitation with installing new model software because it requires an administrator for installation into CJH and CJW due to security concerns. 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 compute-intensive problem solving (CJW). Table 5b presents the qualitative performance evaluation results for Approaches 8 and 9. Considering the main usage of Sciunit, qualitative performance test results are the same as Approach 5, except for offline use.

Table 5. Qualitative evaluation of the strengths, weaknesses, and recommended usages for the five remote approaches.

(a) Approaches 6 and 7: Using CJH and CJW with Docker	
Strengths	☐ [U] The lowest complexity for users, possible to use preconfigured modeling environments ☐ [U] Possible to use distributed HPC resources for scalable modelling work (CJW) ☐ [U] Possible to install custom Conda VE (CUAHSI JupyterHub)
Weaknesses	☐ [U] Impossible to install particular model software or package 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] Computation- and/or data-intensive problem solving (CJW)
(b) Approaches 8 and 9: Using CJH and CJW with Sciunit	
Strengths	☐ [D, U] The lowest complexity for reproducibility in both developer and user perspective ☐ [U] The most lightweight in 10 reproducible approaches ☐ [D, U] Easy to share Sciunit containers in 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 ☐ [U] Does not encapsulate automatic allocation of parallel computing such as Dask
Recommended usages	☐ [Research] Reliable reproducibility as Sciunit can containerize all reproducible artifacts into a container without significant memory use
(c) Approach 10: Using Binder with Docker	
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

[D] = Developer; [U] = User

Finally, Table 5c presents the qualitative performance test results for Approach 10. This approach allows developers to share modeling environments online with users with a single click. Also, users can add new software or libraries, though users first need to understand how to edit Binder configuration files. Despite these conveniences, MyBinder has a limitation in persistent sessions because it supports these online modeling environments for free. Therefore, if users have no activity 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. 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 cyberinfrastructures are developed using Docker.

3.2.2 Recommended Approaches for Common Use Cases

Summarizing the qualitative metrics while also drawing on insights from the quantitative evaluation, we recommend best practices for leveraging containerization and computing environments to achieve reproducible environmental modeling objectives. These recommendations are provided in Tables 6 and 7 for the local and remote approaches, respectively. We considered common use cases in environmental modeling around two broad categories: education and research. Traditionally, we conduct environmental modeling through classes and workshops for educational purposes in an “offline” manner that requires installing software on local computers. However, many educational institutions are transitioning to remote or “online” compute environments (Prasad et al., 2020). Therefore, we divide the objectives of education into online and offline. For environmental modeling research, we can generally divide the steps required to perform computational modeling into 1) model installation, 2) model application, and 3) data analysis for data-intensive computations using model-associated files (Addor et al., 2020).

In the case of local approaches (Table 6), Approaches 3 and 5 are recommended for educational use cases. One reason for this recommendation is because these approaches have low required competency scores, suggesting they are less complexity to install and configure. Of these two approaches, Approach 3 may prove a better choice as Docker containerizes every dependency into Docker images. However, if users want a more lightweight approach to distribute containerized images without considering version control, Approach 5 that uses Sciunit may be a preferred choice. For research purposes, especially for model development, Approach 1 is the only approach to efficiently build new or modified model software source code. Other approaches can only create a container image using existing model software source code for reproducibility. For the purpose of model application in research use cases, Approach 2 is recommended because it has the flexibility to install and apply new Python libraries for various analyses and visualizations. For data analysis, remote approaches are preferred to local approaches because of the space and time required for such data-intensive computations within a local environment.

In the case of remote approaches (Table 7), for online education purposes, approaches 6 and 7 are recommended approaches because they offer the lowest required competency scores for users. These environments support easy sharing via HydroShare and preconfigured modeling environments. Sciunit also has the lowest required competency; however, because Sciunit needs to encapsulate dependencies and workflows together, sometimes creating Sciunit containers can be more difficult compared to other approaches because they can only create a container image using existing model software source code for reproducibility. For research purposes, especially for model development, remote approaches are not recommended because source code changes to the core model software are difficult to make in the remote approaches. If the research application is primarily about performing model runs instead of making changes to the core model software, then Approach 6 is recommended among the remote approaches because it has the flexibility to install and apply new Python libraries for various analyses and visualizations. For the purpose of computationally intensive problem solving, Approach 7 is recommended because it takes advantage of multiple cores and processors for parallel computing, resulting in a lower runtime.

Table 6. Recommended best practices for reproducible approaches on local environments.

Objectives		**Best Practices**
(a) Education	(1) Online (Class or Workshop)	-
	(2) Offline (Class or Workshop)	☐ Containerizing All Software with Docker (AP-3) and Sciunit (AP-5) → The first (AP-5, score:3) and second (AP-3, score:5) lowest complexity for users, a more stable approach (AP-3), and the most lightweight artifacts (AP-5)
(b) Research	(3) Model Installation	☐ Compiling Model Software (AP-1) → The only approach to build new or modified model software source code
	(4) Model Application	☐ Containerizing Core Model Software with Docker (AP-2) → Lower complexity than others (AP-2, score:4), flexibility to install and apply new Python libraries for various analysis and visualization
	(5) Computation- and/or Data-Intensive Problem Solving	-

Table 7. Recommended best practices for reproducible approaches on remote environments.

Objectives		**Best Practices**
(a) Education	(1) Online (Class or Workshop)	☐CJH and CJW with Docker (AP-6 and 7) and Binder with Docker (AP-10) → The lowest complexity for users (score:3), a flexible approach, and easy sharing
	(2) Offline (Class or Workshop)	-
(b) Research	(3) Model Installation	-
	(4) Model Application	☐ CJH with Docker (AP-6) → Lower complexity than others (AP-6, score:3), flexibility to install and apply new Python libraries for various analysis and visualization
	(5) Computation- and/or Data-Intensive Problem Solving	☐ CJW with Docker (AP-7) → The first fastest computational time, possible to use multiple cores for parallel computing

4. Discussion

4.1 Containerization as a Means for Promoting Open and Collaborative Environmental Modeling

Containerization-based reproducible approaches are specifically designed to meet the demanding needs of collaborative model development across varied software and hardware environments. By adopting containerization, researchers can create container images that encapsulate models, dependencies, and software configurations, ensuring a unified and reproducible development environment. This allows multiple researchers or institutions to work within the same software environment and produce consistent results, regardless of their individual setups. Containerization also facilitates model portability across different hardware environments, including local workstations, high-performance computing clusters, and cloud infrastructure. It offers flexibility for collaborators to utilize their preferred hardware setups while maintaining compatibility and consistency. Additionally, collaborative model development entails other aspects such as iterative refining of the model, use of interactive development environments, and efficient sharing of containers. Some use cases corresponding to these aspects have been explored in Ahmad et al. (2022) that demonstrated some necessary extensions to containers. Notably, 'Sciunit-export' enables a seamless transition between Sciunit and other virtual environments like Virtual Env and Conda, further enhancing collaborative model development practices.

In the context of environmental modeling, the role of open-source software and open data in promoting the adoption of reproducible approaches and facilitating collaboration among researchers in the field of environmental modeling is paramount. Open availability of software and data reduces duplicated efforts and fosters higher quality science, improves transparency, and encourages a stronger science-policy boundary (Pfenninger et al., 2017). While there are valid reasons for not openly sharing data and code, such as ethical and security concerns, potential exposure of flawed code or data, additional workloads, and institutional or personal inertia (Pfenninger et al., 2017), it is crucial to understand the practicalities and importance of open code and data. Open practices can be supported through measures like changing attitudes, requesting data-code-workflow-environments during manuscript review process, initiating intellectual property rights/licenses, assigning digital object identifiers (DOIs), and establishing distribution channels for proper recognition. Adhering to these guidelines improves the reproducibility of modeling results, enabling others to verify and build upon the existing work. In the context of containerization, standardized practices enhance interoperability among different containerized models, facilitating their seamless integration into larger scientific workflows and promoting collaboration among researchers. Notably, there is currently a strong momentum for open-source data and software across various scientific domains, including geology, energy, climate modeling, and environmental modeling, extending beyond computer science and data science (Knoth & Nüst, 2017; Fiore et al., 2019; Pfenninger et al., 2017; Morsy et al., 2017; Essawy et al., 2018; Choi et al., 2021).

4.2 Software Licensing and Security Challenges Associated with Containerization

Containerization approaches like Docker, Singularity, and Sciunit run in the Linux operating system, which is Free and Open-Source Software (FOSS). If there is proprietary and licensed software in the Linux operating system, we can consider three specific limitations or challenges. The first challenge is the possibility of containerizing the software. To containerize the software, installation of software inside a container requires using containerization configuration files, such as a dockerfile for Docker and a definition file for Singularity, inside a container. The second challenge is the possibility of process-based containerization such as Sciunit. Sciunit extracts executed codes to efficiently containerize the software. However, there is a possibility of access limitations to the software to protect the software. Finally, a limitation is the uses allowed by the software license and how the software license is implemented. There are many types of licenses and many implementations of licenses including a distributed offline licensed key, a network license key, a subscription-based license, etc. For sharing reproducible approaches, every approach requires an agreement or permission from the owners of the software for any users or concurrent users. Furthermore, there can be specific limitations or challenges such as license compliance, cost, technical compatibility, version updates, dependencies etc. For example, the SUMMA model used in this research is freely accessible under an open-source license, facilitating its use and modification without licensing restrictions. In contrast, proprietary software like TUFLOW (Two-dimensional Unsteady FLOW) may require users to obtain a license and potentially pay fees for certain usage contexts or commercial purposes.

Related to license challenges are security challenges especially when using remote approaches such as CJW and CJH with Docker, Singularity, and Sciunit for environmental modeling. Some of these challenges include data leakage, network security, and malicious containers. To ensure data privacy and integrity, cyberinfrastructures generally use strong authentication to prevent unauthorized access, update containers and underlying software regularly, and verify the authenticity and integrity of containers before deployment. For example, we need to use verified ‘Hydroshare ID’ to use the functionality of CJW and CJH platforms. Also, developers are unable to install new models in CJH and CJW independently, without undergoing a review process conducted by the larger CJH and CJW development teams, as mentioned earlier. Additionally, both CJH and CJW undergo regular maintenance and security measures to safeguard against potential security threats. By proactively maintaining security, remote environments aim to protect the integrity and privacy of the data and models hosted on their platforms, providing a secure and reliable environment for researchers and users.

4.3 Opportunities for Future Research

4.3.1 Advancing Sciunit for Environmental Modeling

Sciunit was shown to be lightweight and time-efficient in the reproducible approaches considered in this study. Sciunit is a tailored environment for geoscience modeling that is still in active development. In its current implementation, Sciunit containerizes the workflow software including software and data dependencies into a single container. Other containerization approaches, such as Docker and Singularity, are aimed at a more general audience but do allow for the separation of the computational modeling environment from the workflow itself. This separation allows for more flexibility in applying different data processing workflows based on containerized computational environments. Sciunit developers are working on adding functionality that could allow a user to create a Docker image from a Sciunit container (Chuah et al., 2020). Exploring such approaches to combine lightweight tailored containerization tools that are specific to domains like environmental modeling, alongside industry standard containerization approaches like Docker, could provide a power approach for bring containerization technology to environmental modeling.

4.3.2 Opportunities for Hybrid Containerization

Integrating or developing a hybrid approach that combines the strengths of multiple containerization tools, like that just described between Sciunit and Docker, is another promising research direction worth exploring in the future. Such approaches can leverage the advantages of different containerization technologies to address specific needs and challenges in environmental modeling and resulting in a flexible and efficient approach for managing and executing containerized applications. By combining tools like Docker, Singularity, Sciunit, and Binder, researchers can potentially benefit from a wider range of features and capabilities. For example, Docker is a widely used interface and offers broad community support, while Singularity focuses on high-performance computing and compatibility with existing HPC systems. Sciunit provides a lightweight, user-friendly framework for creating and sharing scientific models and assessments for environmental research, and Binder facilitates the creation of interactive and reproducible computational environments. A hybrid approach could involve using Docker or Singularity as the base containerization technology and integrating Sciunit and Binder to enhance model accessibility, reproducibility, and collaboration. This combination can enable researchers to package and distribute models using Docker or Singularity, while leveraging the interactive and reproducible features of Sciunit and Binder for easier model evaluation and sharing. Recently there has been significant attention given to such an effort. Youngdahl et al. (2018) demonstrated the use of an automatic hybrid containerization tool called 'Sciunit-Popper' for simplifying the sharing, porting, and reproducing of distributive and iterative experiments. Brown et al. (2019) utilized a hybrid 'Docker- Kubernetes' containerization approach by initially using Docker for deploying GUI/GPU instances and later transitioning to Kubernetes for scalability, deployment, and portability. However, the application of such approaches in environmental modeling is still limited, presenting an opportunity for further research and exploration in the field. In our ongoing work, we are investigating the potential of running distributed applications using a scheduler like Kubernetes, which will be a part of our future endeavors.

4.3.3 Automating Containerization and Model Execution using ML and AI

Another challenge requiring future research is automating containerization and model execution into end-to-end workflows with appropriate resource allocation, scaling, workload balancing, and performance monitoring. Such automation reduces manual effort, automates decision-making, and improves efficiency. In the current study, even though we presented guidelines for the best practices for different modeling use cases, optimization of containerization and model execution processes, potentially automating some aspects of model configuration and setup are still challenging. Recently, the integration of machine learning (ML) and artificial intelligence (AI) has shown the potential for code completion using tools like Github copilot and large language generative models such as ChatGPT (Ouyang et al., 2022). While these tools have the potential to reduce work and speed up the time required to build end-to-end workflows, research is needed to explore the opportunities and limitations of ML and AI-based automation in environmental modeling, given the unique challenges and the importance of process understanding.

4.3.4 Real-time Software Reconfiguration of Containerized Workflows

Approaches 6 and 7 use a Jupyter interface, which has become a widely used tool for providing access to preconfigured modeling environments (Prasad et al., 2020). However, such configurations that rely on Jupyter interfaces can have challenges associated with allowing users to install new software. Environmental modeling, because of the diversity of models used within the community, would benefit from approaches that allow for easy configuration of the software environment behind the Jupyter interface. The "udocker" tool, which is a tool for using Docker without privileges (Gomes et al., 2018), could be a solution for allowing users to add new model software to a Docker image to customize the environment for a particular modeling application. Binder, included in Approach 10, is also a powerful tool to provide customization of remote modeling environments with Jupyter interfaces. Using an implementation of Binder like MyBinder is possible now, but being a general environment, it has limitations for environmental modeling.

As stated earlier, in its current implementation if users have no activity for 10 min, the MyBinder user session is automatically shut down. MyBinder sessions on BinderHub are open to anybody, anywhere, and anytime for free. Therefore, some time limits for BinderHub user session resources are inevitable to prevent misuse of resources. It is possible to automatically save a session when it is shut down, which is a partial solution. Building a cyberinfrastructure system to support environmental modeling that combines BinderHub with more persistent data and compute resource to support reproducible environmental modeling seems like an especially promising future research direction.

4.3.5 Education and Training for Reproducible Environmental Modeling

Common across all of the discussed approaches, education and training plays a crucial role in promoting awareness and effective implementation of reproducibility approaches in environmental modeling. Part of this education is about the importance and challenges associated with reproducibility specifically in the context of environmental modeling. Reproducibility of computational models has long been cited as a challenge due to factors such as model complexity, size, lack of incentives, focus on novelty, etc. (De Vos et al., 2011). Additionally, the sharing of open data-code, and well-documented workflows is still optional in the review and publication process for environmental modeling (Stagge et al., 2019). For one reason, many model developers and users are either unaware or lack the skills to implement these approaches, which benefit greatly from a strong knowledge of containerization techniques and computational skills (Stagge et al., 2019). Scientific cyberinfrastructures like the ones discussed in this paper, HydroShare, CyberGIS for Water, and CUAHSI Jupyterhub, along with many others are working to overcome these challenges and lower the barrier to reproducibility. Research is continuing to highlight the significance of reproducibility in environmental modeling and explore various techniques and methodologies to ensure the production of reproducible results (Morsy et al., 2017; Essawy et al., 2018; Choi et al., 2021). There are also a growing number of opportunities for technical training and demonstration of containerization tools and concepts through conferences, workshops, and training sessions. These efforts aim to ensure that environmental model developers and users are aware of reproducibility approaches and can effectively implement them in their work, thereby promoting reproducibility in environmental modeling and related fields.

5. Conclusions

Reproducibility is the cornerstone of science as it allows for accumulating knowledge by building on prior work. However, many have highlighted the difficulties in achieving reproducible computational research. For environmental modeling, knowledge gaps in achieving reproducible computational modeling remain in understanding how to effectively use modern software tools and practices to achieve this desired outcome. To this aim, we explored ten approaches for achieving reproducible modeling goals using a combination of different containerization tools on both local and remote computational environments contrasting developer and user efforts. We assessed the ten approaches using a hydrologic modeling application against both quantitative and qualitative metrics. Based on this evaluation, our goal is to establish guidelines for the best practices for different modeling use cases common in the environmental modeling community.

For use cases where the objective is to develop new environmental models and it is important to be able to recompile model source code on a frequent basis, Approach 1 as that uses GNU Make and Conda Virtual Environments may be sufficient, or it may be effective to apply Approach 5 using Sciunit to containerize the end-to-end modeling workflow for easier reproducibility and portability. For cases where a given model is applied for a specific system without changes to the core model source code, we recommend approaches where the core model software is containerized and used locally (e.g., Approach 2) or where users interact with the model through a JuypyterHub environment, like CUAHSI JupyterHub (Approach 6), assuming the Dockerized core model software can be uploaded into the JupyterHub environment and made accessible to end users. Of the ten approaches considered, the CyberGIS-Jupyter for Water platform (Approach 7) is recommended for computationally intensive applications given that the platform provides access to high performance and high throughput computational resources, which resulted in relatively low runtimes in our scenarios. For educational use cases, the recommended methods are those that take advantage of remote environments with Jupyter interfaces, like CUAHSI JupyterHub and CyberGIS-Jupyter for Water or use Binder (Approaches 6, 7, and 10), assuming making changes to the core model software are not part of the learning objective. For cases where the educational objective includes having students edit or extend the core model software, then using Sciunit for containerizing the software (Approach 5) is recommended because it offers a low required competency compared to more general containerization approaches.

While this study considers ten approaches for reproducible environmental modeling, this is not an exhaustive list and new approaches continue to be introduced. Given our review of these approaches and considering their relative strengths and weaknesses, we can suggest possible directions for future research and development. Although the trend of environmental modeling appears to be moving to remote or cloud computational environments, providing deployment flexibility of such environments for environmental models remains a challenge. We are encouraged by approaches like Binder that allow for on-demand creation of remote virtual environments. A Binder-based environment for environmental modeling that allows for more persistent sessions and larger data storage solutions could be powerful. Furthermore, containerization approaches like Sciunit that are tailored for geoscience modeling use cases provide benefits that larger, less tailored containerization technologies (e.g., Docker or Singularity) cannot provide. However, merging of tailored and industry-standard containerization strategies as hybrid approaches may be able to harness the strengths of both approaches and provide solutions for environmental modelers seeking to create more reproducible environmental studies. Ultimately, these approaches can help to lower the barrier to fostering a "culture of reproducibility" (Rosenberg et al., 2020) that supports open and collaborative environmental modeling.

Data and Software Availability

The data and computational environments used in this study are available as ten HydroShare resources and three GitHub repositories. We published all data and computational environments with persistent digital object identifiers (DOI) on HydroShare and shared them by a collection resource (HS-1) in HydroShare (Choi et al., 2022). This collection resource provides the links for all HydroShare resources as "Collection Contents" and three GitHub repositories as "Related Resource Reference." The ten HydroShare resources consist of one collection resource, two composite resources for SUMMA model inputs (HS-2, HS-3), one composite resource for the Virtual Box image used across the five local approaches (HS-4, Approaches 1~5), four composite resources for Jupyter notebooks used in the four remote approaches (HS-5~8, Approaches 6~9), and one composite resource for a Jupyter notebook used to create Figures 3-6 using performance results (HS-9) and one composite resource for the Singularity image (HS-10). In addition, three GitHub repositories were created to share Approach 10 and to show how to create a Docker and a Singularity image for Approach-2, 3, and 4.

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

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

Github copilot: https://github.com/features/copilot/

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

Sciunit: http://sciunit.run

Singularity Hub: https://singularityhub.com

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

Acknowledgments

This work was supported by the United States National Science Foundation under collaborative grants ICER-1928369, ICER-1928315, OAC-1664061, OAC-1664018, and OAC-1664119. We acknowledge the work of the Reprobench and HydroShare teams that made this research possible.

We also thank Natalie Thompson, and the consultants of the University of Virginia Graduate Writing Lab for their helpful feedback in preparing the manuscript.

References

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

Ahmad, R., Choi, Y. D., Goodall, J. L., Tarboton, D., Nassar, A., & Malik, T. (2022). Improving reproducibility of geoscience models with Sciunit. *Recent Advancement in Geoinformatics and Data Science*, *2558*(07). https://doi.org/10.1130/2022.2558(07)

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

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

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

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

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

Bentaleb, O., Belloum, A. S. Z., Sebaa, A., & El-Maouhab, A. (2022). Containerization technologies: taxonomies, applications and challenges. *Journal of Supercomputing*, *78*(1), 1144–1181. https://doi.org/10.1007/s11227-021-03914-1

Brown, M., Renambot, L., Long, L., Bargo, T., & Johnson, A. E. (2019). COMPaaS DLV: Composable infrastructure for deep learning in an academic research environment. *Proceedings - International Conference on Network Protocols, ICNP*, *2019-October*, 1–2. https://doi.org/10.1109/ICNP.2019.8888070

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

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*, *135*. https://doi.org/10.1016/j.envsoft.2020.104888

Choi, Y., Goodall, J., Nguyen, J., Ahmad, R., Malik, T., Li, Z., et al. (2022). Comparing Approaches to Achieve Reproducible Computational Modeling for Hydrological and Environmental Systems, HydroShare.

Chuah, J., Deeds, M., Malik, T., Choi, Y., & Goodall, J. L. (2020). Documenting computing environments for reproducible experiments. *Advances in Parallel Computing*, *36*(September), 756–765. https://doi.org/10.3233/APC200106

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

Clark, Martyn P, Nijssen, B., Lundquist, J. D., Kavetski, D., Rupp, D. E., Woods, R. A., et al. (2015). Water Resources Research, 2515–2542. https://doi.org/10.1002/2015WR017198.A

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

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

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 & Software*, 104753. https://doi.org/https://doi.org/10.1016/j.envsoft.2020.104753

Fiore, S., Elia, D., Palazzo, C., Dranca, A., Antonio, F., Williams, D. N., et al. (2019). Towards an Open (Data) Science Analytics-Hub for Reproducible Multi-Model Climate Analysis at Scale. *Proceedings - 2018 IEEE International Conference on Big Data, Big Data 2018*, 3226–3234. https://doi.org/10.1109/BigData.2018.8622205

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*, *8*(11). 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. Toward the Geoscience Paper of the Future: Best practices for documenting and sharing research from data to software to provenance, 3 Earth and Space Science § (2016). John Wiley & Sons, Ltd. 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

Hothorn, T., & Leisch, F. (2011). Case studies in reproducibility. *Briefings in Bioinformatics*, *12*(3), 288–300. https://doi.org/10.1093/bib/bbq084

Hut, R. W., van de Giesen, N. C., & Drost, N. (2017, May). 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 our. *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

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), 113–120. 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

Knoth, C., & Nüst, D. (2017). Reproducibility and practical adoption of GEOBIA with open-source software in Docker containers. *Remote Sensing*, *9*(3). https://doi.org/10.3390/rs9030290

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

Kovács, Á. (2017). Comparison of different linux containers. In *2017 40th International Conference on Telecommunications and Signal Processing, TSP 2017* (Vol. 2017-Janua). 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

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 & Software*, *39*, 3–23. https://doi.org/10.1016/j.envsoft.2012.09.006

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

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

Merkel, D. (2014). Docker: lightweight Linux containers for consistent development and deployment. *Linux Journal*, *2014*(239), 2. https://doi.org/10.1097/01.NND.0000320699.47006.a3

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 and Software*, *93*, 13–28. https://doi.org/10.1016/j.envsoft.2017.02.028

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C. L., Mishkin, P., et al. (2022). Training language models to follow instructions with human feedback, (NeurIPS). Retrieved from http://arxiv.org/abs/2203.02155

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 & Geosciences*, *53*, 3–12. https://doi.org/10.1016/j.cageo.2012.04.002

Pfenninger, S., DeCarolis, J., Hirth, L., Quoilin, S., & Staffell, I. (2017). The importance of open data and software: Is energy research lagging behind? *Energy Policy*, *101*(November 2016), 211–215. https://doi.org/10.1016/j.enpol.2016.11.046

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

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

Reinecke, R., Trautmann, T., Wagener, T., & Schüler, K. (2022). The critical need to foster computational reproducibility. *Environmental Research Letters*, *17*(4), 41005. https://doi.org/10.1088/1748-9326/ac5cf8

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), 01820002. https://doi.org/10.1061/(ASCE)WR.1943-5452.0001215

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

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

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

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*, 1–12. https://doi.org/10.1038/sdata.2019.30

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* (pp. 374–383). Institute of Electrical and Electronics Engineers Inc. https://doi.org/10.1109/eScience.2017.51

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

De Vos, M. G., Janssen, S. J. C., Van Bussel, L. G. J., Kromdijk, J., Van Vliet, J., & Top, J. L. (2011). Are environmental models transparent and reproducible enough? *MODSIM 2011 - 19th International Congress on Modelling and Simulation - Sustaining Our Future: Understanding and Living with Uncertainty*, (December), 2954–2961.

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

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

Yin, D., Liu, Y., Hu, H., Terstriep, J., Hong, X., Padmanabhan, A., & Wang, S. (2018). CyberGIS-Jupyter for reproducible and scalable geospatial analytics. *Concurrency Computation* . https://doi.org/10.1002/cpe.5040

Youngdahl, A., Yuan, Z., Hai, D., That, T., Malik, T., & Jimenez, I. (2018). Semantically Organized Containers for Reproducible Research Containers, 1–4.

Yuan, Z., Ton That, D., Kothari, S., Fils, G., & Malik, T. (2018). Utilizing Provenance in Reusable Research Objects. *Informatics*, *5*(1), 14. https://doi.org/10.3390/informatics5010014