International Journal of Geo-Information

MDPI

Article

Simplifying Land Cover-Geoprocessing-Model Migration with a PAMC-LC Containerization Strategy in the Open Web Environment

Huaqiao Xing [1,2,3], Haihang Wang [1,2], Denghai Gao [1,2], Dongyang Hou [4,*] and Huayi Wu [5,6,7]

1 School of Surveying and Geo-Informatics, Shandong Jianzhu University, Jinan 250101, China; xinghuaqiao18@sdjzu.edu.cn (H.X.); 2021165107@stu.sdjzu.edu.cn (H.W.); 2022165110@stu.sdjzu.edu.cn (D.G.)
2 Key Laboratory of Digital Simulation in Spatial Design of Architecture and Urban-Rural, Department of Education of Shandong Province, Jinan 250101, China
3 Department of Civil Engineering, Toronto Metropolitan University, Toronto, ON M5B 2K3, Canada
4 School of Geosciences and Info-Physics, Central South University, Changsha 410083, China
5 State Key Laboratory of Information Engineering in Surveying, Mapping and Remote Sensing, Wuhan University, Wuhan 430079, China; wuhuayi@whu.edu.cn
6 Hubei Luojia Laboratory, Wuhan 430079, China
7 Collaborative Innovation Center of Geospatial Technology, Wuhan 430079, China
* Correspondence: houdongyang1986@csu.edu.cn

Abstract: Land cover and its changes over time are significant for better understanding the Earth's fundamental characteristics and processes, such as global climate change, hydrology, and the carbon cycle. A number of land cover-geoprocessing models have been proposed for land cover-data production with different spatial and temporal resolutions. With the massive growth in land cover data and the increasing demand for efficient model utilization, developing efficient and convenient land cover-geoprocessing models has become a formidable challenge. Although some model-migration methods have been proposed for handling the massive data, the intricacy of land cover-data and -heterogeneity models frequently prevent current strategies from directly meeting demand. In this paper, we propose the PAMC-LC-containerization approach to overcome the difficulties associated with moving existing land cover models in the open web environment. Based on the idea of model migration, we design a standardized model description and hierarchical encapsulation strategy for land cover models, and develop migration and deployment methods. Furthermore, we assess the viability and efficacy of the proposed approach by using coupled workflows for model migration and the introduction of visualization on the Mts-WH dataset and the Google dataset. The experimental results show that the PAMC-LC approach can simplify and streamline the model migration process, with important ramifications for increasing productivity, reusing models, and lowering additional data-transmission costs.

Keywords: land cover; containerization migration; Docker; geoprocessing workflow

Citation: Xing, H.; Wang, H.; Gao, D.; Hou, D.; Wu, H. Simplifying Land Cover-Geoprocessing-Model Migration with a PAMC-LC Containerization Strategy in the Open Web Environment. *ISPRS Int. J. Geo-Inf.* **2024**, *13*, 187. https://doi.org/10.3390/ijgi13060187

Academic Editors: Jamal Jokar Arsanjani and Wolfgang Kainz

Received: 21 April 2024
Revised: 29 May 2024
Accepted: 30 May 2024
Published: 3 June 2024

1. Introduction

Land cover is significant for better understanding the Earth's fundamental characteristics and processes, such as global climate change, hydrology, and the carbon cycle [1]. Land cover-geoprocessing models are of great significance for monitoring dynamic changes in global land cover, climate, and the sustainability of ecosystems, as well as for predicting future changes [2–4].The emergence of "Land Cover 2.0", which represents a new era in land cover geoprocessing, is due to the free and open access to data, prepared analysis data, high-performance computing, and rapidly evolving data processing and analysis capabilities [5]. In recent decades, remarkable advancements have been achieved in land cover-geoprocessing models, spanning data, methodology, computational capabilities, and

practical applications [6]. Specifically, these models encompass a range of sophisticated techniques, including maximum likelihood classification, random forest, support vector machine, neural networks, change-detection models, and more.

Land cover-geoprocessing models often involve complex and resource-intensive workflows, which require significant computing power to process large amounts of remote sensing data [7]. Remote sensing data has often been downloaded locally and analyzed using specialist desktop applications. However, with the emergence of large and accessible remote sensing datasets, this approach has become increasingly time-consuming and inefficient [8]. Moreover, the processing and computation efficiency of conventional methods is limited by device performance, and the transmission of large volumes of land cover data requires high network capacity. Therefore, it is an ideal solution for end-users to migrate their data-processing models and environments in order to directly access the required processing results from the data side, reducing additional data-transmission expenses and improving work efficiency. In this context, the development of network technologies for sharing, reusing, and integrating land cover models in network environments has gained widespread acceptance [9]. Reusing computing algorithms is a top goal in these efforts, not only to prevent redundant implementation and increase utilization efficiency, but also to ultimately acquire more effective and high-quality models. Given the size and quantity of remote sensing datasets, it is essential to move the computing code closer to the data rather than transferring a large amount of data to the processing services. Furthermore, elastic computing environments such as grids and cloud computing also require technology that can seamlessly distribute land cover-processing logic in scalable environments [10]. By embracing these technological improvements, users can handle land cover data effectively and efficiently, supporting a variety of efficient applications like environmental monitoring and land cover planning.

Model migration is a crucial topic in software engineering that deals with moving various software-system components from one platform to another in order to meet evolving business needs, customer expectations, and technical improvements [11]. In recent years, researchers and practitioners have scrutinized these migration issues from multiple perspectives, proposing theoretical frameworks, design patterns, implementation strategies, and evaluation metrics. This research includes ensuring compatibility with the new system, preserving functionality and data, and minimizing downtime during the migration process. For example, the utilization of ONNX's (Open Neural Network Exchange) open format can facilitate the exchange of machine learning models across different frameworks and platforms [12]. The ActiveSpace framework was proposed for transferring binary code to a buffer and compiling it in real-time [13]. Numerous container platforms, such as OpenVZ (OpenVZ Virtualization Environment) [14], LXC (Linux Containers) [15], and Docker [16] fully or partially support container migration. Research in this field mainly concentrates on developing automated tools and processes to expedite code migration, enhance reliability, reduce risk, and increase re-usability [17]. Moreover, standardized methods and technologies are required to simplify human–computer and computer–model interactions, whether supporting service composition or promoting the use of cross-domain and interdisciplinary models. Some research aims to increase the accessibility and interoperability of models by lowering the threshold for model execution, including developing web-based-modeling methods with simple graphical user interfaces, linking different models via application programming interfaces (APIs) [9], and providing models as accessible web services [18]. A recent example is the use of OGMS-WS (OpenGMS Model Service Wrapper) to share environmental models as web services [19]. However, land cover-processing schemes based specifically on virtual containers to replace data transmission have not been fully researched and implemented. Given the complexity of land cover, which involves diverse terrain types, various vegetation species, and the impacts of human activities, it is crucial to properly consider the unique characteristics of spatial heterogeneity, temporal variability, and scale dependency in land cover-information data, as well as the model's functional logic and interface mapping. Although numerous efficient model-migration methods have

evolved in recent years, none of them are capable of concurrently handling heterogeneous model compatibility, interactivity, and portability in various situations.

In order to address the above challenges, this paper presents a novel model-migration strategy known as PAMC-LC (Polyglot Algorithmic Migration Container for Land Cover). First, a standardized model-description interface was designed for the land cover-geoprocessing model to simplify the obstacles in the deployment and use of the model. Second, with regard to the layering strategy of the virtual container, an encapsulation scheme was proposed for the land cover-geoprocessing model addressing two aspects, i.e., runtime-resource encapsulation and runtime-environment encapsulation. Finally, the migration and deployment schemes of the encapsulated model were developed according to the actual needs of the land cover-geoprocessing-model migration. Compared with existing migration solutions, PAMC-LC can readily accommodate multiple model development languages, have both interfaces and visualization interfaces, and provide a mobility solution based on virtual containers.

The remainder of this paper is organized as follows. Section 2 presents the transfer concept and the PAMC-LC method. Section 3 introduces the specific implementation and purpose of model migration through two walkthrough examples. Section 4 presents the evaluations and discussion of the effective service chains with PAMC-LC. Finally, conclusions and future work directions are given in Section 5.

2. Methods

2.1. Breaking the Barriers of Data Transfer with PAMC-LC

Data is frequently considered as isolated entities in the conventional land cover-data service [20]. It is a typical paradigm for land cover geoprocessing using user-provided models or applications at personal workstation. This requires the original land cover data to be transmitted from the data provider to the user's workstation via a wide area network (WAN). Due to the typically large size of land cover data, the time and cost of transmitting this data over a WAN can be significant. Therefore, it is of great importance to provide on-demand processing capabilities on the land cover-data provider's side, allowing users to submit their data-processing models for execution on the server's side and directly obtain the processing results they need. This is essential for lowering extra data-transfer costs and raising productivity.

This paper examines a crucial issue in land cover geoprocessing: how can we efficiently handle the transmission of large-scale land cover data within an open web environment? A typical WPS (Web Processing Services) server requires clients to send their data to the server for computation before returning the processing results, which is the current model for using remote sensing data. Even though this paradigm is appropriate for managing small data sets, it greatly ties up network capacity when processing huge data sets, such as those produced by bulk data processing or intricate and large-storage-demanding land cover data. With the increasing diversity and capabilities of remote sensing imaging methods, as well as the continuous improvement of spatial and temporal resolutions, the complexity and volume of remote sensing images are increasing geometrically. Nevertheless, effective processing and utilization of massive land cover data is a crucial step. Applying this big data characteristic to WPS services is challenging as the necessary remote sensing images can be several hundred megabytes or more in size. A distributed-geographic-information processing system may also require data to be repeatedly transmitted between different servers to achieve service composition. In synchronous mode, when users execute the "Execute" interface, they need to establish a connection with the server, submit data, and wait for a server response, during which the user is blocked. Prolonged waiting times may render the service unfeasible. Although an asynchronous mode can partially address this response issue, users still need to endure long time delays, which cannot be fundamentally resolved even with data compression and the use of raw binary formats.

As a result, rather than replacing data exchange, the ideas of model and environment transfer enable the algorithm code to be shifted closer to the site of the data. As shown

in Figure 1, the idea of model transfer can also be used to process massive and dispersed data sets across nodes, allowing specific parallelization strategies. The cutting-edge state of big data processing also shows that there is still a need to enhance cross-platform code exchange and interoperability.

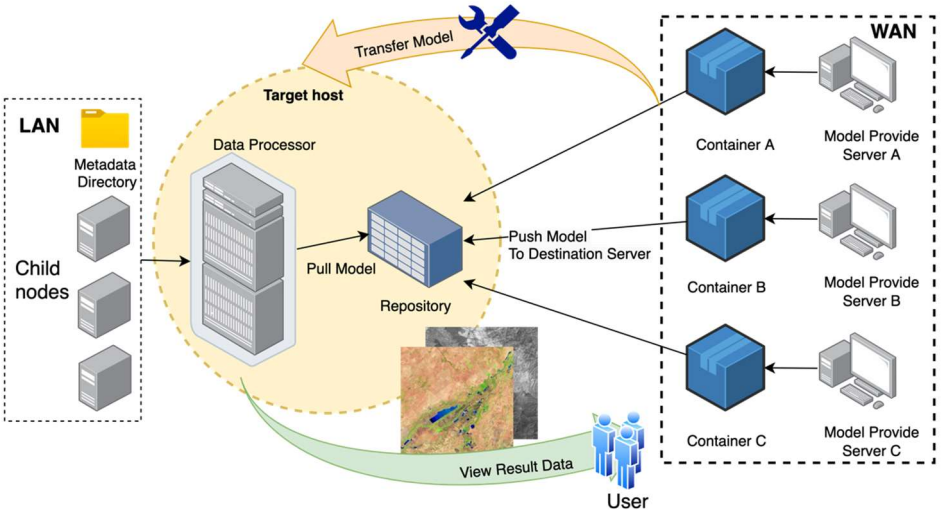


Figure 1. The conceptual framework of land cover-model migration.

This study suggests a model-encapsulation method based on container technology, called Polyglot Algorithmic Migration Container for Land Cover (PAMC-LC), to address the needs of model migration in land cover-processing applications. This approach can handle a variety of difficulties that existing models encounter during migration. The method has three distinct qualities in particular: (1) During the migration process, the code and environment are encapsulated together in a container to ensure the model's correct operation across different systems and avoid conflicts between different models. (2) As land cover-processing applications involve multiple programming languages, the strategy supports mainstream programming languages and provides multiple language interfaces. (3) To enhance the model's usability and availability, the strategy encapsulates the model as a service and provides an HTML interface and API. With this strategy, the goal of "one encapsulation, multiple deployments" can be achieved. Users do not need to be concerned about potential environmental incompatibilities that could lead to model malfunction because the model, code, dependencies, and environment are all housed within the container. In conclusion, the PAMC-LC model-encapsulation approach is an effective strategy for model migration. Subsequent sections will discuss the specifics of this strategy's implementation and application cases.

2.2. Simplifying Model Deployment with Resource Interfaces

To provide a solid foundation for model deployment, it is essential to provide a structured description of the heterogeneous features of the model-runtime environment. There are many languages available for standardizing and describing packaged applications, which are used to direct the deployment of model applications in the process of generating models. For example, the Deployable Software Description (DSD) language is used to describe complex internal and external dependencies of software systems; the Open Software Description (OSD) language can be used to describe software components, versions, and internal structures of packaged software; and the Management Information Format (MIF) language adds many standard fields to support software deployment, which can be used to describe elements of different operating systems. These languages can be used to describe software, but because of their sophisticated syntax and extensive content, they are less user-friendly.

The runtime-environment description document should be created with people in mind when it comes to land cover-processing models. The environment requirements for deployment and execution are made explicit to model users through this interface. Moreover, the runtime-environment description document is also intended for machine interpretation, allowing for the determination of the completeness of the operating environment and the facilitation of subsequent adaptation and deployment processes.

This work proposes a model-resource-description interface to give a uniform and consistent description of encapsulated geographic models in order to complement our suggested strategy. The interface is based on the RESTful service architecture style, where a web service is seen as a collection of all the resources in the project. Each resource corresponds to a unique URL identifier, which can be accessed to perform operations on the resource. The overall design of the interface is shown in Figure 2 and includes several sections such as basic model information, input and output descriptions, runtime descriptions, software and hardware environments, and license information.

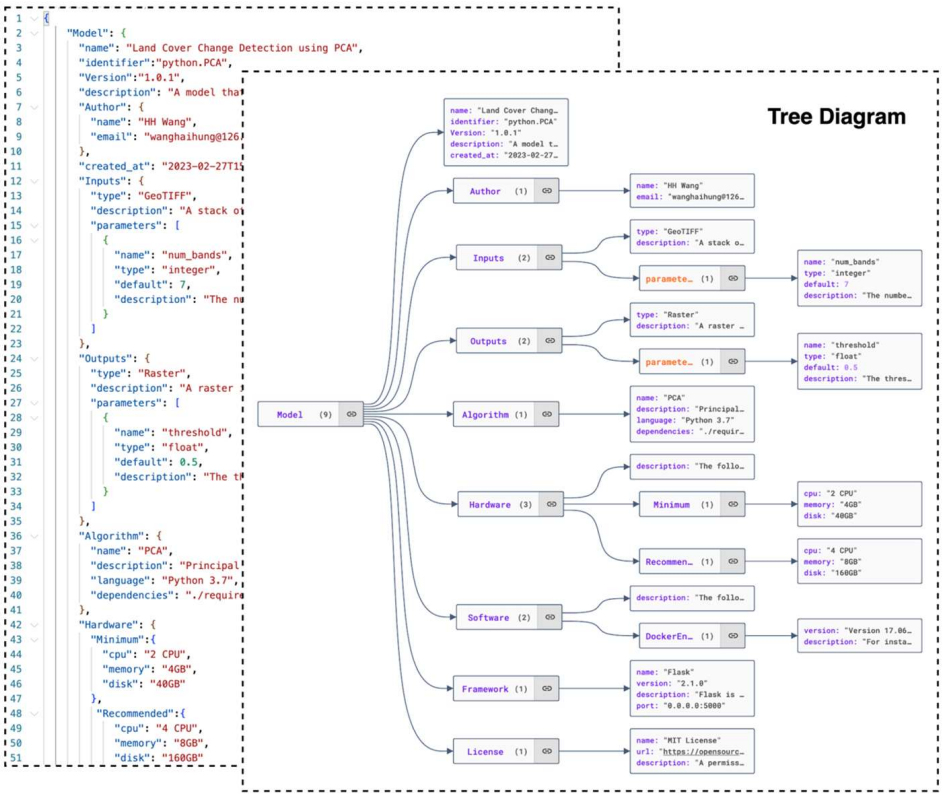


Figure 2. Model-resource-description interface in PAMC-LC.

Model providers can more precisely structure and specify model-deployment information by utilizing the model-resource-description interface. Additionally, model users can use the data in the runtime description document to have a comprehensive knowledge of the environmental requirements for model deployment and operation. The interface uses an approachable JSON format, which improves readability and makes writing and editing easier. The interface also adheres to particular patterns and criteria, making it simple to extend and validate. It is possible to nimbly insert dynamic content, new elements, or characteristics when creating algorithm descriptions.

2.3. Streamlined Model Migration with Container-Based Encapsulation

The core concept of the model-encapsulation strategy is to bundle the model, code, dependencies, and environment in a container, enabling fast and reliable model migration. Specifically, the container consists of two parts: the runtime resources and the runtime

environment. The runtime environment encompasses the operating system, programming language-runtime environment, and other system-level dependencies, whereas the runtime resources include code, the model, and other application-level dependencies. In the model-encapsulation strategy, both the runtime environment and the resources are packaged as an image file and loaded together upon container startup.

2.3.1. Four-Tier Model Deployment: Runtime-Resource Encapsulation

In an effort to deploy land cover models as a usable model service, we sought to develop an encapsulation-based strategy. Based on this concept, this article reorganized the deployment package into four layers: (1) the core deployment resources of the model; (2) the resources for executing the model service; (3) the resources for validating the model; and (4) the resources for configuring and extending the model. As depicted in Figure 3, these layers include fields that provide information on the land cover-processing logic and its specific implementation [21].

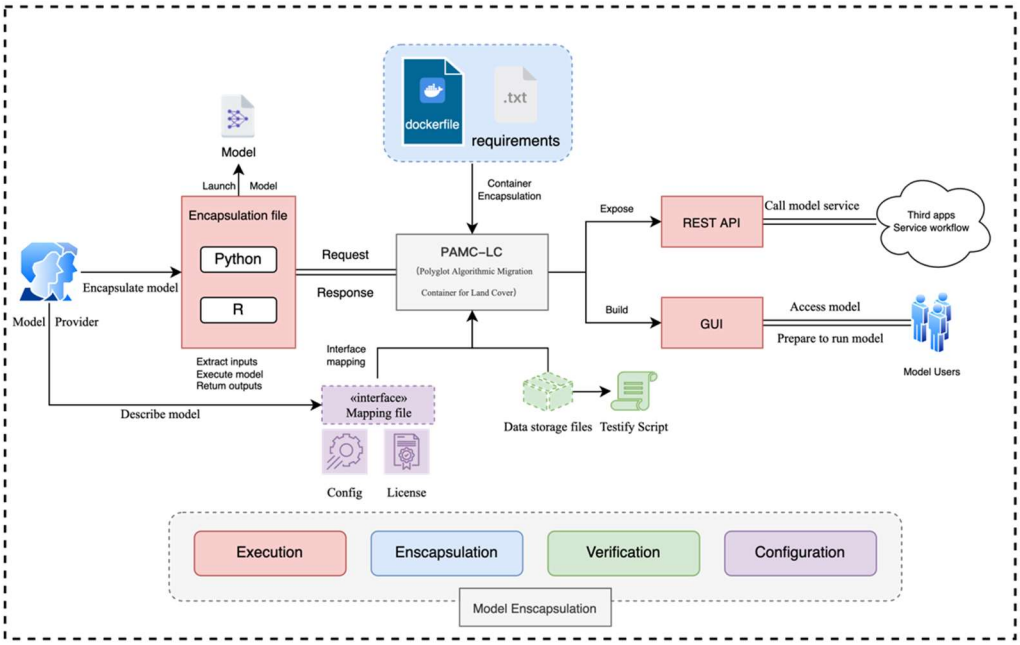


Figure 3. Runtime-resource hierarchical encapsulation structure.

At the level of model service utilities, in addition to the files related to model execution, there is other supplementary information that is provided to help users gain a more comprehensive understanding of the deployed model service. Firstly, the model's core encapsulated resources mainly consist of the image construction files and the necessary dependent environment, which serves as the foundation of model construction. Secondly, the model service-execution resources include the contents that support the operation of the model structure, such as core algorithms, GUI interfaces, and cached data. Additionally, in the execution resources, we can include heterogeneous programming languages such as Python and R. Thirdly, the model-validation resources contain demonstration data and validation scripts to assist in verifying whether the deployed model service can be used. This also helps model service users to gain a clearer understanding of the input/output data of the model service and prepare appropriate data when using specific model services. The validation script file mainly records the input and output file names of the demonstrated model. Finally, the model configuration and extension resources include the model-description interface and configuration files. The model-description interface is used to provide information for the standardized description of the model and the construction of the GUI interface. The configuration file configures the process number for handling requests, asynchronous processing methods, exposed interfaces, timeout duration, and

other contents related to the model. It also includes a license document to explain the licensing copyright of the model service.

2.3.2. Minimizing Heterogeneity: Runtime-Environment Encapsulation

In order to facilitate the re-use and integration of models, it is essential to consider the issue of model heterogeneity. Models for land cover processing are developed by different researchers or research groups using various techniques or methods, and therefore they are typically built on different platforms (e.g., Linux and Windows), written in different programming languages (such as Python, R, Matlab, and Java), and employ different user interfaces (such as command-line, desktop software, executable files, and dynamic link libraries). In order to make it easier for model users to re-use various models, these heterogeneities must be reduced. This process, known as heterogeneous model encapsulation, seeks to lessen the execution variances from the initial design [8].

Researchers or model providers frequently employ numerous programming languages to provide services or applications in the field of practical model development. Typically, all these components must function in a variety of settings, including development, testing, and production. If every component makes use of a different language environment, then managing and maintaining various environments may become exceedingly challenging. Additionally, heterogeneous model encapsulation could be very helpful when working with big, complicated datasets. It is possible that a single model will not be sufficient to account for every trend in the data. A sophisticated model that works more accurately, reliably, and efficiently on fresh, untested data can be produced by integrating many models. To properly manage land cover data, this form of model building necessitates careful consideration of the differences between heterogeneous models. As a result, they must be incorporated into a single framework.

It is possible to accelerate continuous integration and deployment, simplify environment configuration, increase application portability, and reduce resource usage by encapsulating various language environments using Docker and combining them into a single workspace [16]. The numerous language components and their dependencies, which make up the foundational elements of the framework, take up a comparatively high storage proportion of the base image throughout the unification process. Pure algorithms have a nearly nonexistent network-transmission cost. In fact, many similar or identical services require a certain base image to be constructed. By open-sourcing the base image, users can download or save the already-pulled base image in advance, making it easier to use similar or identical services in the future, which can improve execution efficiency. In terms of service composition, this model is extremely important.

Figure 4's illustration of the PAMC-LC multi-language model method shows how it is designed. Each layer in the Docker container is incrementally modified, serving as building blocks for constructing the image. This modular design allows different images to share a common base image, enabling us to encapsulate different programming languages in different layers. To accommodate different development languages, we use layers to construct the container during the building process. An image is composed of one or more layers, and when these layers are "stacked" together, they form the image we see. Layers in an image are read-only, meaning we cannot modify the data contained in these layers. The data in the image layers can only be read, and when a container is started based on an image, a new read–write layer is added on top of the read-only layers. This read–write layer does not belong to the image but to the current container, and all operations within the container are performed in this read–write layer. The data we see in the container is the result of combining the read–write and read-only layers, and these two layers together are referred to as the "container layer". By constructing different layers in this way, we can encapsulate multiple programming language environments. By packaging several languages and dependency libraries in different levels, we may even add or delete current algorithms in the "container layer". By ensuring that each layer only contains the installation files required for the associated language or dependency library and excluding all other files,

this method considerably reduces the size of the picture. It significantly reduces the size of the image by simply sending incremental or differential data to the image.

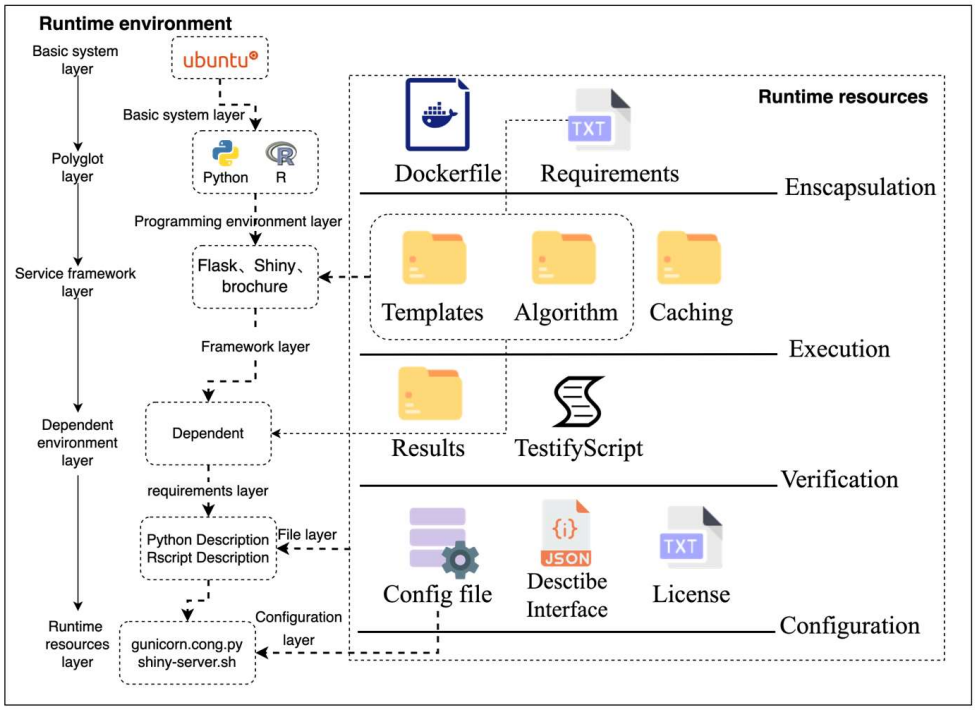


Figure 4. Runtime-environment hierarchical encapsulation structure.

2.4. Moving and Deployment Strategy for Encapsulated Model

Data interoperability refers to the ability of data to be correctly interpreted and used between different systems or organizations [22]. The foundation of model applications is data exchange and manipulation, but they are often challenged by technical and conceptual difficulties. Conceptual challenges involve resolving the different representations of data and knowledge by model builders and providing different levels of model interoperability. Technical challenges involve implementing "dialogue" between models, automating data exchange, collaboratively executing models, and ensuring the repeatability and reproducibility of model configuration and processing [23]. Datasets should be interoperable during the data exchange process so that they can be converted and prepared for use as input by different models. Furthermore, additional computations are needed to adapt data to different model requirements, such as the spatial scope, resolution, and format of the data [24]. Data interaction between models should come first in the collaboration and "dialogue" processes of models, then should come automated or semi-automatic deployment of algorithms or models, and finally reliable transmission and storage capabilities should be guaranteed [25].

First, in order to handle and store our model-container images throughout the migration phase of our models, we must pre-create a private image repository on the target host server. Private image repositories have several advantages over public image repositories: (1) They save network bandwidth and improve Docker deployment speed. We do not need to download images from the public image repository every time, but rather only need to download them from the private repository. (2) They boost safety. In general, we only put our application-packaged pictures in the private repository rather than the public image repository. Thus, image tampering or leakage can be avoided.

Secondly, data in Docker containers is non-persistent. Data volumes are one or more directories that are specifically designated in containers, bypassing the union file system and providing various useful features for persistent or shared data. Therefore, data files can be shared between containers and hosts. Data stored on data volumes continues to

exist after the container's lifecycle and can be shared and accessed by other containers. Each deployment of a container may result in residual and unnecessary containers causing resource waste. When a data volume is attached to a container, this feature should not be used. An elegant way to reduce the risk of data loss is to use a processing-separated data volume. Sharing and replicating volume data provides decoupling between data and services. In addition, by setting up distributed regions on different nodes, data and applications can be strategically placed in multiple regions. This can reduce failure points, indirectly increasing application and data availability.

The main goal of the model-deployment phase is to arrange a collection of containers. A YAML configuration file that specifies the Docker container ensemble overrides the runtime settings of the containers. A tool called Compose makes it possible to build and run multi-container Docker applications. It permits setting up a YAML file to configure an application's services. All services can then be created and launched from the configuration using a single command, as seen in Figure 5b. In this architecture, the client may communicate through an API gateway instead of through the service directly in order to connect to each service. With greater flexibility and reduced risk, the development team can deploy updates, add new features, or embrace other technology stacks without disrupting the entire program. To preserve changes made in the container, administrators need to commit these changes to an image. Docker will then create additional layers on top of the existing image. Automatic update-deployment scripts are shown in Figure 5c. Image startup and deployment times are relatively short, enabling new developers to quickly join the project, as they only need to understand the single service that provides the functionality they are working on, rather than the entire system. A single service provided by the image model can be integrated as a component into other systems, and if the usage of that functionality is excessively high, then the service can be scaled outwards to handle high loads. Therefore, this service architecture can be utilized for horizontal scaling, and the system will not suffer from a single point of failure.

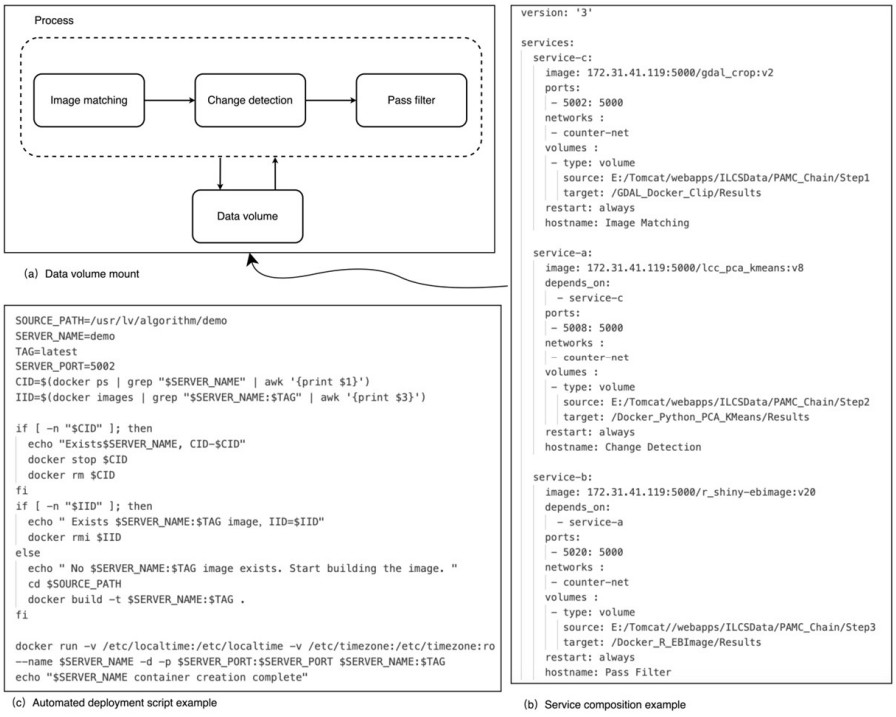


Figure 5. Model-deployment script and Compose service combination pseudocode.

3. Experiment

3.1. Containerization Model Transfer and Verification

3.1.1. Enhancing Sharing and Security

To ensure the security of our private image repository, we must consider two aspects: first, the security of the image repository itself. For example, when using it, appropriate security certificates must be configured and communication must take place using the HTTPS protocol. The second concern is the security of the image's pulling procedure, which includes authentication, looking for image signatures, and scanning while pulling images. To manage and store our model-container images, we have employed Harbor, an open-source, cloud-native, image repository project. Harbor offers additional functionalities, such as security scanning, role-based access control, and log collection, among others. Its structure is illustrated in Figure 6.

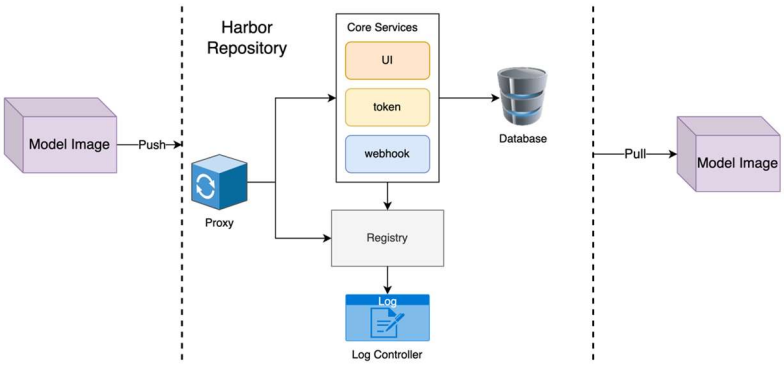


Figure 6. Harbor's private-mirror-warehouse structure.

During this stage, three services for storage and management were deployed into the Harbor image repository of the target domain. Table 1 includes details about these three services.

Table 1. Three services providing information for processing task.

ID	Service-A	Service-B	Service-C
Name	Change Detection	Pass Filter	Image Matching
Function	PCA/K-Means	Pass/Filter/Morphing	Matching/Crop
Language	Python	R	Python
Storage	172.31.41.119:5000/lcc_pca_kmeans:v6	172.31.41.119:5000/r_shiny-ebimage:v20	172.31.41.119:5000/gdal_crop:v2
Description	Identifies changes or differences that occur in the data	A signal processing method that can filter out unwanted frequency components and retain frequency components of interest	Matches the image according to the coordinate system and crop out the common area

The Change Detection process consists of PCA and K-Means algorithms and encompasses three essential stages: the generation of difference images and feature vector space (EVS), the construction of the feature vector space (FVS), and clustering of the feature vector space and change map. The Pass Filter module includes high-pass and low-pass filters, as well as erosion and dilation operations. In the Image Matching stage, images are matched based on coordinate systems, and a common region is extracted by cropping, ensuring smooth execution of the Change Detection process.

3.1.2. Achieving Automated Model Deployment and Interoperability

The use of automated deployment has grown in popularity during the past few years. Firstly, it can significantly improve the repeatability of software builds, which is essential

for ensuring consistent application deployment across platforms and hosts. Secondly, automated deployment can reduce the time required to deploy and launch applications, especially in cloud-based environments. Finally, automated deployment can also achieve faster and more reliable software-change testing and deployment, thereby improving the efficiency of continuous integration processes. During the deployment phase of our model, we built automated scripts to update and deploy its migrated container, and the automated update-deployment script is shown in Figure 5c.

To facilitate adapting and deploying models in computing nodes, we suggested a model-resource-description interface in Section 2.2. Data volumes and bind mounts are the two main methods used for data exchange in containers. Whenever Docker uses data volumes, it creates one or more directories on the host computer and uses those as data volumes. These data volumes have higher portability and maintainability and can share data between containers. At the same time, data volumes can also be used for data persistence, ensuring that data still exists after the container is deleted. When using bind mounts, data in the container can interact with any directory or file on the host machine, but this method has certain limitations and is not conducive to sharing data between containers. Therefore, employing data volumes helps to ensure that data is transported from one model to another when exchanging data between multiple models. As an illustration, a user starts the Service-A container, create some data, and save it to the/data directory. Next, a user starts the Service-B container, mount the same data volume to it, and then read the data they created in the Service-A container from the Service-B container. These steps can be used to mount the data volume to various containers in order to provide data sharing and durability.

3.1.3. Registering and Accessing Model Resources through a Portal Website

In this paper, we have developed a model-description interface and a template file for generating dynamic interfaces to models based on migration strategies. The model-description interface (implemented by the model provider) is used by the deployment process to transform computing nodes into usable model services and establish connections with model service users, as shown in Figure 7. This interface can be incorporated as a web component in third-party applications or accessed as a service through an API.

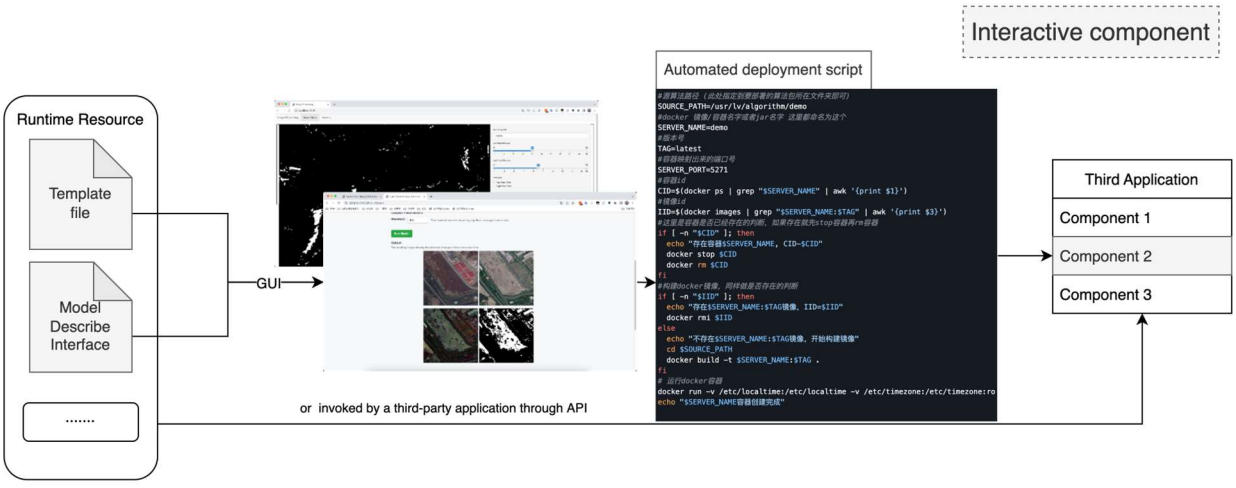


Figure 7. Integration of the virtual container as a separate component.

Stateless and loosely linked application programs with any interacting components are required for the container's application programs. Container configurations can be changed in the event of a failure without affecting the availability of the entire system. This approach offers the following benefits: (1) Rapid service deployment and dismantling; components can be added, removed, and managed in real-time within third-party applications, and the application process within the Docker container can be launched instantly due to the

absence of any startup procedures. When the application is no longer needed, the container can be dismantled, leaving no trace on the host operating system. (2) It enables the model to become an independent, reusable entity, and facilitates the creation of tightly coupled and loosely coupled systems. It can be used for the fast execution of workflows to build different models, making the models more easily reusable and adaptable to different application scenarios [26].

In this section, we demonstrate the utilization of a component-based GUI for land cover-change detection in the study area, facilitated by image-processing services. We are assuming successful deployment of the Change Detection and Pass Filter models on the target host, developed using Python and R programming languages, respectively, and provided externally via distinct RESTful frameworks. In addition, the Multi-temp Scene Wuhan (Mts-WH) dataset is stored on the target server, comprising two large, high-resolution remote sensing images obtained from the IKONOS sensor, with dimensions of 7200 $\times$ 6000 pixels, covering the Hanyang district of Wuhan, China. The images were acquired in February 2002 and June 2009, and after GS (Gram–Schmidt) algorithm fusion, they possess a 1 m resolution and include four bands (blue, green, red, and near-infrared) [27,28].

As shown in Figure 8, users of computing resources can directly launch the model image in the deployed node without additional configuration by utilizing existing computer nodes. The model-resource-description interface describes basic information regarding the model container, and registers it in the computing resource repository. Subsequently, the UI provided to external users can be loaded into the system as a component via the portal website, enabling the construction of available model services. The model's built-in GUI provides users with inputs, executions, outputs, and information descriptions, and generates a Change Detection execution case via visual operations that invoke this model service. In this case, we use two services: Change Detection and Pass Filter. Change Detection is based on the Python language, provided by the Flask framework, and includes PCA and K-Means algorithms; Pass Filter is based on the R language, provided by the shiny-server, and includes high-pass filters, low-pass filters, morphological erosion, and morphological dilation. It is worth noting that we can use a combination of erosion and dilation to perform opening and closing operations on the image. Firstly, we upload the prepared 2002 and 2009 images of Hanyang district, set the number of image bands, and after clicking "Run Model" the computed results will be displayed directly on the page, including original and processed images. Subsequently, using image-processing tools provided by Pass Filter, we can optimize the resulting image and remove spurious patches in the Change Detection results.

As depicted in Figure 9, the data and results obtained via model calculations are illustrated. Figure 9a,b, respectively represents the high-resolution remote sensing images of the Han Yang district in 2002 and 2009. Figure 9c displays the change-intensity image obtained through gray mapping of the difference image. In the difference image, brighter pixels represent areas of higher difference, corresponding to regions with more significant changes in the remote sensing images. The black areas in the Change Detection result indicate unchanged regions, whereas the white areas denote the changing areas. Notably, sparse villages represented by scattered houses gradually disappeared in 2009, giving way to the rapid expansion of industrial and residential areas, as shown in the Figure 9d. Most of the unused land in 2002 was converted into other land-use categories. Therefore, it can be inferred that after seven years of development, the western side of Han Yang has transformed into a residential area, while the eastern side has developed into an industrial area [27]. The results obtained from PAMC-LC are consistent with the development situation of the study area.

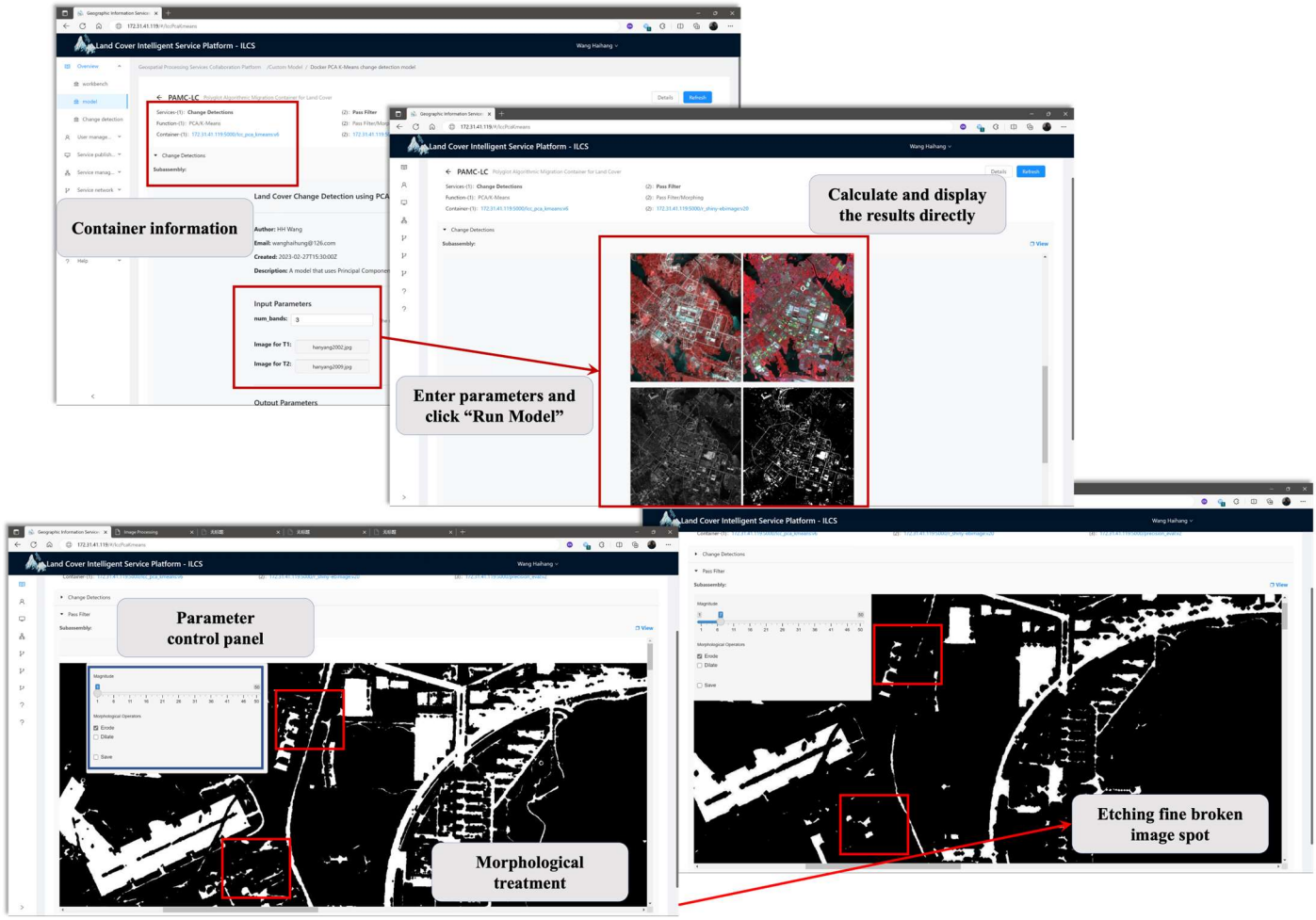
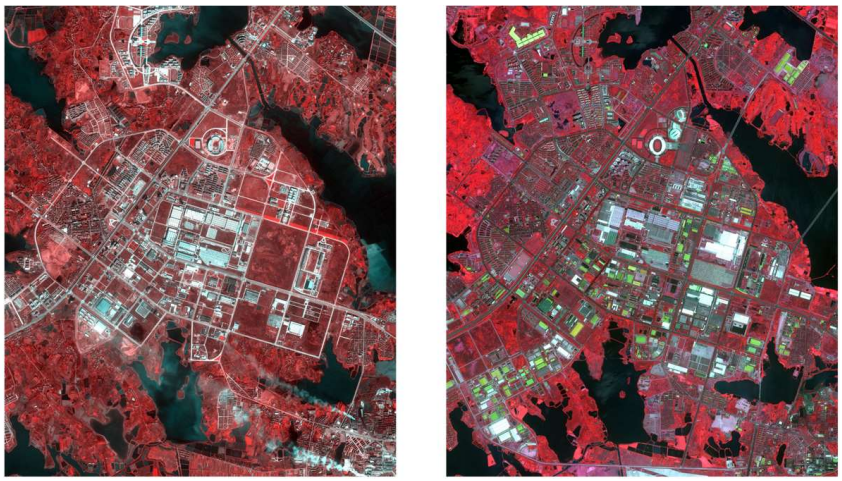
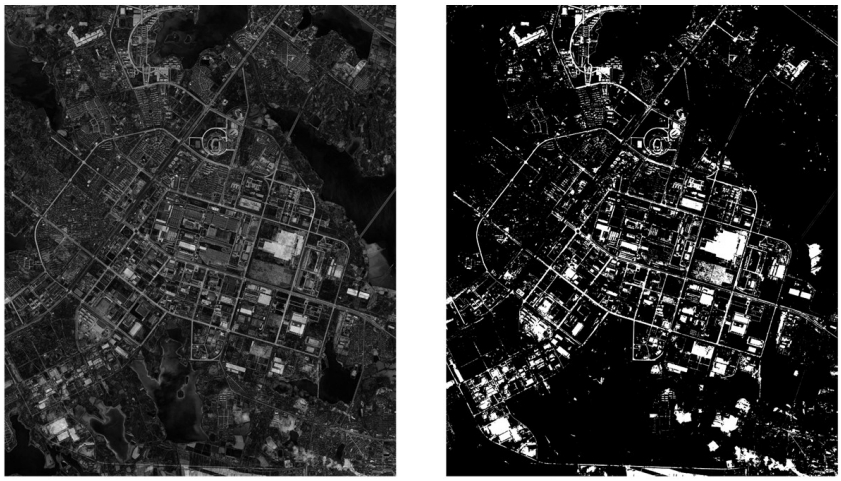


Figure 8. Using the migration model for change detection and image processing.



(a). Image of Hanyang District in 2002 (b). Image of Hanyang District in 2009

Figure 9. *Cont.*



(c). Changing intensity image by detection (d). Change detection results

Figure 9. Change detection results from PAMC-LC.

3.2. Building Effective Service Chains with PAMC-LC

To validate the capabilities and practicality of the proposed strategy, we developed a container-based service chain. Testing and validating models against specific requirements can significantly improve the quality of the models. Integrating a model's environment can be challenging as it requires considerable scientific and programming efforts [29] Even if individual components are tested separately and executed correctly, the integrated output may still produce unexpected results due to different assumptions in the overall process [30]. The purpose of this part is to provide an example application of the change-detection model in order to illustrate the advantages of the PAMC-LC approach. Figure 10 illustrates the design process of the model transfer and composition using this approach.

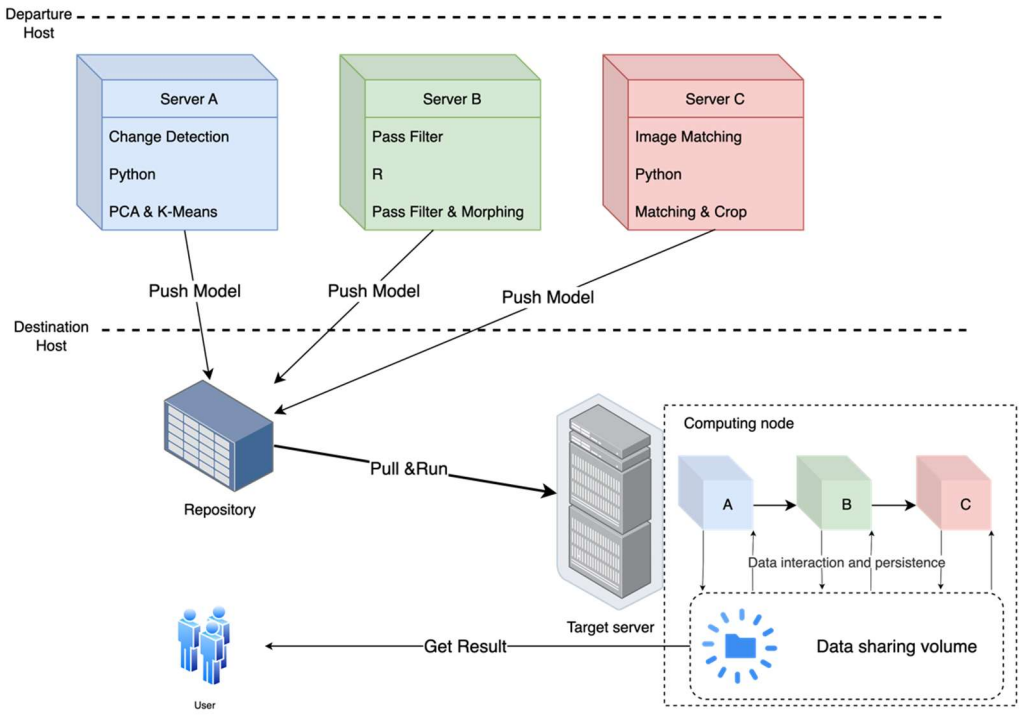


Figure 10. Change Detection-model-migration process.

In this instance, we utilized the Change Detection dataset for high-resolution satellite images (Google Data Set) to verify the efficacy of our proposed strategy in constructing

a service chain [31]. The dataset comprises satellite images collected between 2006 and 2019, covering the suburbs of Guangzhou, China. Nineteen pairs of images, which exhibit seasonal changes, were collected using Google Earth services through BIGEMAP software. These images consist of three spectral bands—red, green, and blue—with a spatial resolution of 0.55 m and sizes ranging from 1006 × 1168 pixels to 4936 × 5224 pixels. To validate the feasibility of integrating the transfer model into a service chain, a visual execution demonstration of change detection was developed.

Suppose we have three source servers, A, B, and C, each storing models for change detection, filtering, and accuracy evaluation, which were developed using different programming languages. In order to validate the effectiveness of PAMC-LC in different scenarios, we want to construct a service chain through the interface of these models and estimate the land cover changes that occurred in the study area at a certain time. To achieve this goal, we deployed three atomic services, namely Image Matching, Change Detection, and Pass Filter, on the target host. These services were developed using different programming languages and frameworks and encapsulated in cross-platform containers.

We have deployed three services on the target host based on the PAMC-LC strategy, enabling the application to access these services via an API. The model-resource-description interface provides basic information about the model container, and based on this interface, we have developed a visual workflow to verify the combinatorial capability of multi-programming language models. As shown in Figure 11a–c, we need to manually enter parameters for configuration. In (a), we upload two phases of images that need change detection, we select a Change Detection model in (b), and in (c) we then set the erode parameter to "5", and the workflow is built as in Figure 11d. After clicking "Execute", the workflow starts to execute. At this point, the containers are called for the computation tasks, and the results are displayed as in Figure 11e upon completion. The output results include the original images of the two periods, the matched images after region matching, the Change Detection results, and the post-processed images. Through these steps, we have successfully established a service chain containing multiple atomic services to estimate the land cover changes in the study area. This model migration and deployment strategy can be applied in different scenarios and has high scalability and flexibility.

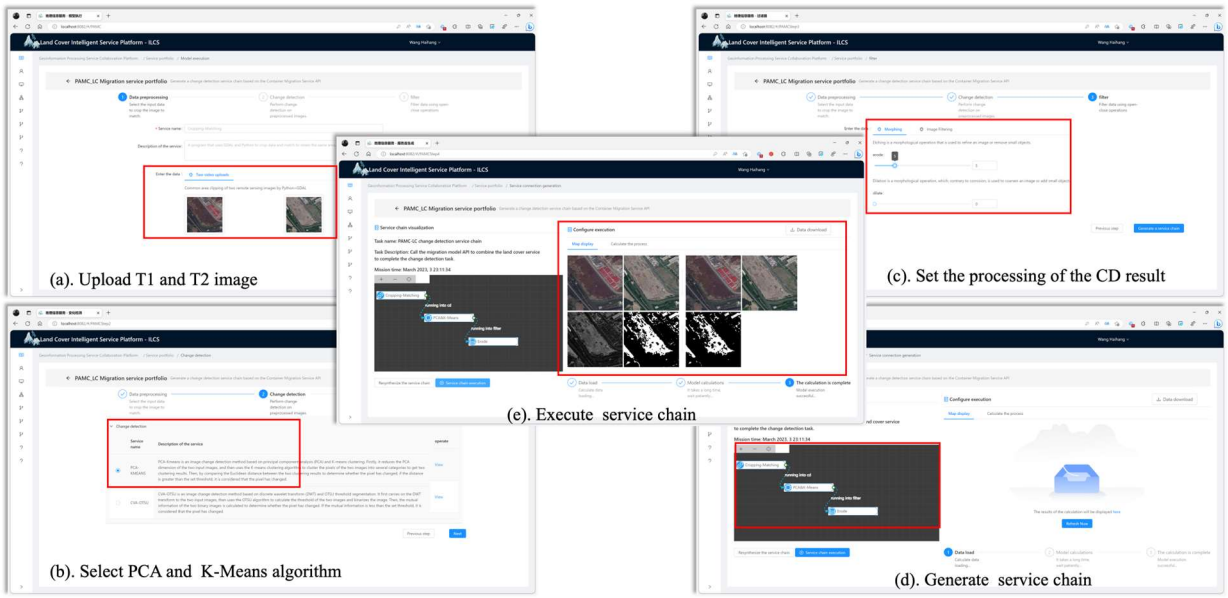


Figure 11. Using service chain for land cover processing (Steps is displayed in red boxes).

The Google Data Set contains 40 high-resolution satellite images of the suburbs of Guangzhou that were taken between 2006 and 2019. We processed these images using the suggested workflow. Some of the processed results are shown in Figure 12. Our experimen-

tal results demonstrate the efficacy of the PAMC-LC model in handling high-resolution satellite image data, providing strong support for future related research. The processed images reveal significant changes in land cover, such as urban expansion, deforestation, and agricultural development. These changes have important implications for urban planning, environmental management, and resource conservation. Moreover, the workflow-based change-detection approach exhibits high scalability and adaptability, making it suitable for processing large volumes of satellite image data from different regions and time periods.

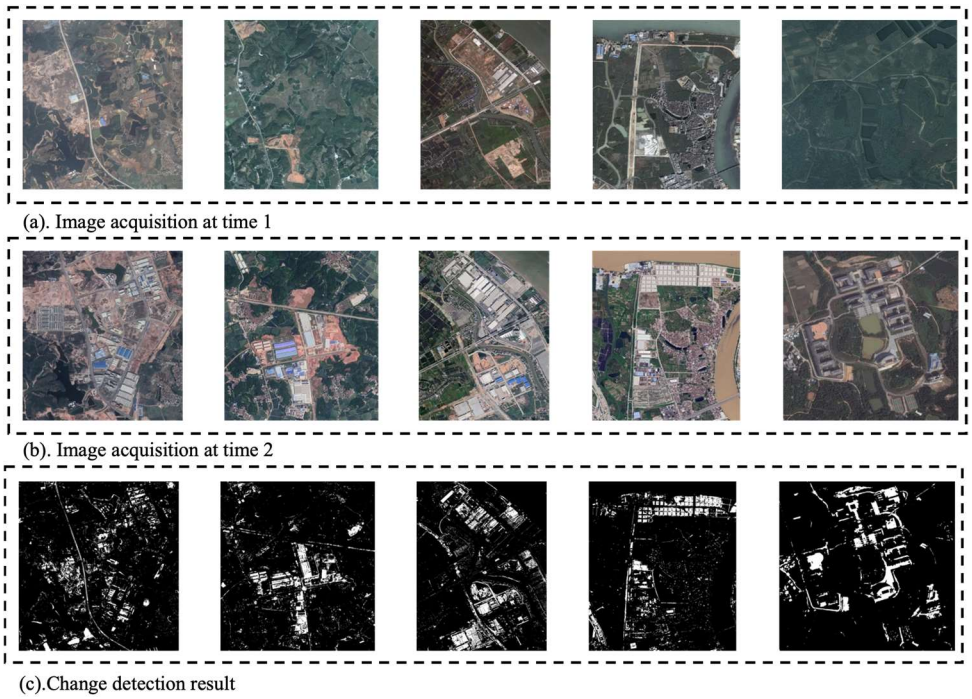


Figure 12. Partial processing result of Google Data Set.

In order to verify the reliability of the results, we selected enough samples for accuracy assessment. During the evaluation process, we randomly selected two change test results. The first study area selected 453 "change" samples and 374 "no change" samples, and the second study area selected 442 and 557 corresponding samples. Accuracy assessment showed that the overall accuracy for each of these two regions was 0.88 and 0.86, respectively, and the Kappa coefficient for each region was 0.75 and 0.72, respectively.

4. Discussion

To assess the effectiveness of the PAMC-LC strategy in reducing data migration costs, we convened a seminar with five professionals in the field and collected results on service usability and time under different modes of operation. Our investigation focused primarily on the level of difficulty in using containerized services for change detection compared to the original processing services, as well as the time required to obtain change detection results. The original services provide web services by the same algorithm and are placed on different servers. Unlike PAMC-LC, these services are not containerized. Figure 13 illustrates the efficiency and network stability of using containerized services for change detection, with Figure 13a showing the average time required for image processing and Figure 13b displaying the number of network interruptions during operation, with each member conducting five different change-detection experiments on images.

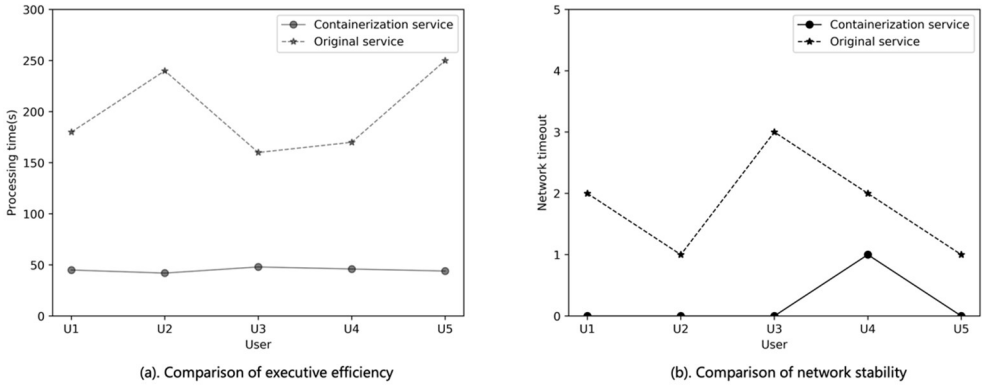


Figure 13. Testing by different users.

The data must be communicated at least four times, from the user to the first processing service, to the last processing service, and then back to the user, assuming that we have three processing services and that the current algorithm requires processing services to perform change detection. The amount of data that is transmitted over the network should rise by two, in theory. The migrated services are more reliable and efficient, as evidenced by the investigation results in Figure 13a. Furthermore, the image must also be transferred at least twenty times between the three services and the user if the original services are used for five tests. This causes 1–3 network oscillations and necessitates that the user re-upload the data.

In the case of service composition, it has been found that loading the model onto the data side yields faster results in all of the considered scenarios. This is due to the smaller size of the transmitted model (see Table 2) and the size of the results returned to the application being smaller than the original data size, resulting in faster transmission times. The data to be processed on the user's host comprises 40 remote sensing images with a total size of 1.42 GB. When processed with the original services, a total of 4.38 GB of data is transmitted, far exceeding the 1.58 GB of data transmitted with the PAMC-LC strategy. Additionally, multiple data transmissions could exacerbate network turbulence and disruptions and possibly jeopardize data security. In order to minimize repetitive data transmissions and significantly lower network, storage, and time costs in the case of enormous data volumes, users can migrate models to the data side using containers in the mirror-model-migration process. In the example, the returned data was only saved as binary images, so the size was not large. However, for many other services, the size of returned data may be comparable to that of the submitted data. Therefore, it is difficult to repeatedly transmit such vast amounts of data through the network.

Table 2. Comparison between original service and PAMC-LC.

	Original Service				**PAMC-LC**	
Server	**IM**	**CD**	**PF**	**Client**	**Data**	**Client**
Bytes received theoretically	1.42 G	1.42 G	1.42 G	123.8 M	1.46 G	123.8 M
Bytes sent theoretically	1.42 G	1.42 G	123.8 M	1.42 G	123.8 M	1.46 G
Total transfer	4.38 G				1.58 G	
Timeout possibility	Higher				Lower	
Transmission	4 times				2 times	
Data security	No				Yes	

The core algorithm script, which only uses a few kilobytes of memory, is the only part that is really executed because the basic image and its dependent surroundings take up the majority of storage space in a mirror image. If the basic image can be deployed in advance

on the target host, then the amount of model data transmitted through the network can be greatly reduced. In fact, the PAMC-LC runtime-environment encapsulation strategy minimizes the volume of the mirror image when it is transferred from one host to another. Docker only transfers the layers that are missing from the target host after comparing the hash values of each image layer on the local and target sites. The mirror image's size can be minimized while still keeping its integrity in this fashion. However, using a typical processing service could be more effective when the volume of data to be processed is considerably less than the size of the service image.

We simulated low bandwidths ranging from 3 Mbps to 30 Mbps to test our system on a wide area network. As shown in Figure 14, the cost of data transmission increases significantly when the amount of data reaches around 103 Mb (1 Gb), and at this point, the size of the processing model container also reaches the level of gigabytes. In this evaluation process, the time required to move 1.5 GB of data to the service side is approximately the same as the time required to transfer the mirror model to the data side. For data sizes smaller than 1.5 GB, moving data is more efficient, whereas for data sizes larger than 1.5 GB, deploying code on the data side is more efficient. As a result, different processing methods may be appropriate for various application situations. When processing large amounts of data, it may be more appropriate to upload the processing service to the data side; however, when sharing data between devices or when processing tasks with smaller data volumes, it may be more appropriate to upload the data to the processing service side.

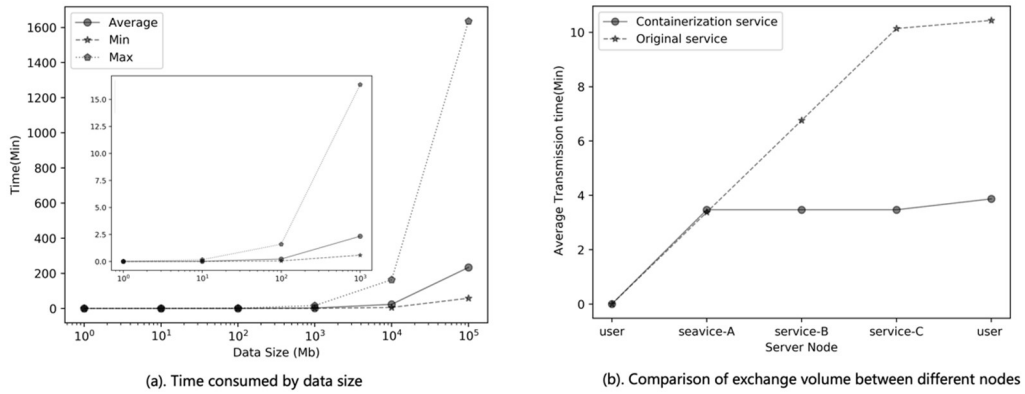


Figure 14. Data exchange efficiency.

5. Conclusions

Multi-Language Containerization Strategy: Our paper proposes a multi-language containerization strategy to facilitate the migration of land cover models. We developed a workflow based on this strategy, demonstrating and verifying land cover-change detection. Through Docker and our encapsulation method, we achieved the seamless migration of Python and R models, along with their dependent environments and service-oriented models.

Efficiency in Data-Processing Tasks: Our results, evaluated using the Mts-WH dataset and Google dataset, highlight the superiority of virtual containers for massive data-processing tasks. By deploying processing models directly to the data side, we effectively managed model components, reducing the overhead associated with system administration and maintenance.

Usability and Flexibility: Through exercise examples, we validated the usability of our suggested model. By creating individual services as standalone elements, while supporting specific objectives, we enable the development of both tightly connected and loosely coupled systems. This approach also facilitates the combination of disparate models to create new ones, enhancing re-usability and adaptability across various application contexts.

Efficiency in Change Detection: Our method for detecting changes in land cover, utilizing 40 remote sensing images, showcased significant improvements in data-transmission efficiency. The PAMC-LC technique reduced data-transmission volume by 63.92% com-

pared to using the original processing service. Moreover, it enhanced change-detection job efficiency by three to four times by avoiding the time cost of data transfer. Consequently, this technique alleviates strain on local computer, storage, and network resources.

Impact and Future Directions: By providing a stable and portable runtime environment for models, our approach has the potential to revolutionize model migration and re-use in massive land cover-data processing. Future research will focus on enhancing virtual container and cross-platform compatibility to further improve web-based land cover-geoprocessing efficiency. At the same time, in the future, more complex model configurations will be added based on the original strategy to provide deep learning, transfer learning and other needs.

Author Contributions: Conceptualization, Huaqiao Xing and Huayi Wu; Methodology, Haihang Wang and Dongyang Hou; Software, Huaqiao Xing, Haihang Wang and Denghai Gao; Formal analysis, Huaqiao Xing and Denghai Gao; Investigation, Haihang Wang, Denghai Gao and Dongyang Hou; Resources, Huaqiao Xing and Dongyang Hou; Data curation, Haihang Wang and Dongyang Hou; Writing—original draft, Huaqiao Xing and Haihang Wang; Writing—review & editing, Huaqiao Xing; Visualization, Denghai Gao and Dongyang Hou; Supervision, Huaqiao Xing and Huayi Wu; Project administration, Huayi Wu; Funding acquisition, Huaqiao Xing. All authors have read and agreed to the published version of the manuscript.

Funding: This research was jointly funded by the Shandong Provincial Natural Science Foundation (No. ZR2022YQ36), the Youth Innovation Team Project of Higher School in Shandong Province (No. 2022KJ201), the Program of China Scholarship Council (No. 202209995003), and the Jinan City and University Integration Development Project (JNSX2023065).

Data Availability Statement: The Mts-WH Dataset is available on the Sensing Intelligence, Geoscience and Machine learning lab resource catalog as "Multi-temporal Scene WuHan Dataset" [http://sigma.whu.edu.cn/resource.php] (accessed on 15 April 2023).The Google dataset is derived from the following resources available in the public domain: [https://github.com/daifeng2016/Change-Detection-Dataset-for-High-Resolution-Satellite-Imagery] (accessed on 21 April 2023).

Acknowledgments: The authors would like to thank the editors and the anonymous reviewer whose constructive comments will help to improve the presentation of this paper.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Foley, J.; Defries, R.; Asner, G.; Barford, C.; Bonan, G.; Carpenter, S.; Chapin, F.S., III; Coe, M.; Daily, G.; Gibbs, H.; et al. Global Consequences of Land Use. *Science* **2005**, *309*, 570–574. [CrossRef]
2. Meyer, W.B.; Turner, B.L. Human Population Growth and Global Land-Use/Cover Change. *Annu. Rev. Ecol. Syst.* **1992**, *23*, 39–61. [CrossRef]
3. Zhao, M.; Xu, T. Research on the Environmental Impacts of Land Use and Land Cover Change. *Res. Soil Water Conserv.* **2005**, *12*, 43–45.
4. Zhao, Q.; Wen, Z.; Chen, S.; Ding, S.; Zhang, M. Quantifying Land Use/Land Cover and Landscape Pattern Changes and Impacts on Ecosystem Services. *Int. J. Environ. Res. Public Health* **2020**, *17*, 126. [CrossRef]
5. Wulder, M.A.; Coops, N.C.; Roy, D.P.; White, J.C.; Hermosilla, T. Land Cover 2.0. *Int. J. Remote Sens.* **2018**, *39*, 4254–4284. [CrossRef]
6. Xing, H.; Zhu, L.; Feng, Y.; Wang, W.; Hou, D.; Meng, F.; Ni, Y. An Adaptive Change Threshold Selection Method Based on Land Cover Posterior Probability and Spatial Neighborhood Information. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2021**, *14*, 11608–11621. [CrossRef]
7. Venter, Z.S.; Sydenham, M.A.K. Continental-Scale Land Cover Mapping at 10 m Resolution Over Europe (ELC10). *Remote Sens.* **2021**, *13*, 2301. [CrossRef]
8. Xing, H.; Chen, J.; Wu, H.; Hou, D. A Web Service-Oriented Geoprocessing System for Supporting Intelligent Land Cover Change Detection. *ISPRS Int. Geo-Inf.* **2019**, *8*, 50. [CrossRef]
9. Laniak, G.F.; Olchin, G.; Goodall, J.; Voinov, A.; Hill, M.; Glynn, P.; Whelan, G.; Geller, G.; Quinn, N.; Blind, M.; et al. Integrated Environmental Modeling: A Vision and Roadmap for the Future. *Environ. Model. Softw.* **2013**, *39*, 3–23. [CrossRef]
10. Yue, P.; Zhou, H.; Gong, J.; Hu, L. Geoprocessing in Cloud Computing Platforms—A Comparative Analysis. *Int. J. Digit. Earth* **2013**, *6*, 404–425. [CrossRef]

11. Williams, J.R.; Paige, R.F.; Polack, F.A.C. Searching for Model Migration Strategies. In *6th International Workshop on Models and Evolution, Proceedings of the ACM/IEEE 15th International Conference on Model Driven Engineering Languages and Systems, Innsbruck Austria, 1–5 October 2012*; Association for Computing Machinery: New York, NY, USA, 2012; pp. 39–44.
12. Danopoulos, D.; Kachris, C.; Soudris, D. Utilizing Cloud FPGAs towards the Open Neural Network Standard. *Sustain. Comput. Inform. Syst.* **2021**, *30*, 100520. [CrossRef]
13. Docan, C.; Parashar, M.; Cummings, J.; Klasky, S. Moving the Code to the Data-Dynamic Code Deployment Using Activespaces. In Proceedings of the 2011 IEEE International Parallel & Distributed Processing Symposium, Anchorage, AK, USA, 16–20 May 2011; IEEE: New York, NY, USA, 2011; pp. 758–769.
14. Romero, F.; Hacker, T.J. Live Migration of Parallel Applications with Openvz. In Proceedings of the 2011 IEEE Workshops of International Conference on Advanced Information Networking and Applications, Singapore, 22–25 March 2011; IEEE: New York, NY, USA, 2011; pp. 526–531.
15. Qiu, Y.; Lung, C.-H.; Ajila, S.; Srivastava, P. Experimental Evaluation of LXC Container Migration for Cloudlets Using Multipath TCP. *Comput. Netw.* **2019**, *164*, 106900. [CrossRef]
16. Boettiger, C. An Introduction to Docker for Reproducible Research, with Examples from the R Environment. *SIGOPS Oper. Syst. Rev.* **2015**, *49*, 71–79. [CrossRef]
17. Xing, H.; Hou, D.; Wang, S.; Yu, M.; Meng, F. O-LCMapping: A Google Earth Engine-Based Web Toolkit for Supporting Online Land Cover Classification. *Earth Sci. Inf.* **2021**, *14*, 529–541. [CrossRef]
18. Li, Y. Towards Fast Prototyping of Cloud-Based Environmental Decision Support Systems for Environmental Scientists Using R Shiny and Docker. *Environ. Model. Softw.* **2020**, *132*, 104797. [CrossRef]
19. Qiao, X.; Li, Z.; Zhang, F.; Ames, D.P.; Chen, M.; James Nelson, E.; Khattar, R. A Container-Based Approach for Sharing Environmental Models as Web Services. *Int. J. Digit. Earth* **2021**, *14*, 1067–1086. [CrossRef]
20. Xing, H.; Zhu, L.; Chen, B.; Zhang, L.; Hou, D.; Fang, W. A Novel Change Detection Method Using Remotely Sensed Image Time Series Value and Shape Based Dynamic Time Warping. *Geocarto. Int.* **2021**, *37*, 9607–9624. [CrossRef]
21. Müller, M.; Bernard, L.; Kadner, D. Moving Code—Sharing Geoprocessing Logic on the Web. *ISPRS J. Photogramm. Remote Sens.* **2017**, *83*, 193–203. [CrossRef]
22. Kadadi, A.; Agrawal, R.; Nyamful, C.; Atiq, R. Challenges of Data Integration and Interoperability in Big Data. In Proceedings of the 2014 IEEE International Conference on Big Data (Big Data), Washington, DC, USA, 27–30 October 2014; pp. 38–40.
23. Knapen, R.; Janssen, S.; Roosenschoon, O.; Verweij, P.; de Winter, W.; Uiterwijk, M.; Wien, J.-E. Evaluating OpenMI as a Model Integration Platform across Disciplines. *Environ. Model. Softw.* **2013**, *39*, 274–282. [CrossRef]
24. Moore, R.V.; Tindall, C.I. An Overview of the Open Modelling Interface and Environment (the OpenMI). *Environ. Sci. Policy* **2005**, *8*, 279–286. [CrossRef]
25. Xing, H.; Liu, C.; Li, R.; Wang, H.; Zhang, J.; Wu, H. Domain Constraints-Driven Automatic Service Composition for Online Land Cover Geoprocessing. *ISPRS Int. J. Geo-Inf.* **2022**, *11*, 629. [CrossRef]
26. Belete, G.F.; Voinov, A.; Laniak, G.F. An Overview of the Model Integration Process: From Pre-Integration Assessment to Testing. *Environ. Model. Softw.* **2017**, *87*, 49–63. [CrossRef]
27. Wu, C.; Zhang, L.; Zhang, L. A Scene Change Detection Framework for Multi-Temporal Very High Resolution Remote Sensing Images. *Signal Process.* **2016**, *124*, 184–197. [CrossRef]
28. Wu, C.; Zhang, L.; Du, B. Kernel Slow Feature Analysis for Scene Change Detection. *IEEE Trans. Geosci. Remote Sens.* **2017**, *55*, 2367–2384. [CrossRef]
29. Bruggeman, J.; Bolding, K. A General Framework for Aquatic Biogeochemical Models. *Environ. Model. Softw.* **2014**, *61*, 249–265. [CrossRef]
30. Voinov, A.; Cerco, C. Model Integration and the Role of Data. *Environ. Model. Softw.* **2010**, *25*, 965–969. [CrossRef]
31. Peng, D.; Bruzzone, L.; Zhang, Y.; Guan, H.; Ding, H.; Huang, X. SemiCDNet: A Semisupervised Convolutional Neural Network for Change Detection in High Resolution Remote-Sensing Images. *IEEE Trans. Geosci. Remote Sens.* **2021**, *59*, 5891–5906. [CrossRef]