# OpenGeoLab: Synergizing service-oriented resources and reproducible workflows for geographic modeling

Peilong Ma[a,b,c,1], Minshuo Zhou[a,b,c,1], Wanhao Li[a,b,c], Wei Xie[a,b,c], Tianyu Sheng[a,b,c], Yongning Wen[a,b,c], Songshan Yue[a,b,c], Guonian Lv[a,b,c], Min Chen[a,b,c,*]

[a]*State Key Laboratory of Climate System Prediction and Risk Management, Nanjing Normal University, Nanjing, 210023, Jiangsu, China*
[b]*Key Laboratory of Virtual Geographic Environment (Ministry of Education of PRC), Nanjing Normal University, Nanjing, 210023, Jiangsu, China*
[c]*State Key Laboratory Cultivation Base of Geographical Environment Evolution, Nanjing, 210023, Jiangsu, China*

## Abstract

Geographic modeling increasingly relies on integrating heterogeneous computational resources, spanning spatiotemporal datasets, preprocessing algorithms, and simulation models. However, the utilization of these resources is hindered by infrastructure fragmentation, where datasets and models reside in isolated platforms with incompatible interfaces. Researchers are forced to function as human middleware, devoting substantial effort to technical coordination. While service-oriented computing, integrated cloud platforms, and containerization technologies have individually advanced resource access and reproducibility, they have evolved as architecturally disconnected paradigms. This paper addresses this gap by proposing a cross-cutting infrastructure pattern for geographic modeling and presenting OpenGeoLab as its reference implementation. The platform establishes a "clean client" environment that synergizes service-oriented resource orchestration with reproducible workflows. Distributed modeling resources are aggregated through standardized, machine-readable service interfaces, while specification-driven graphical interfaces translate visual configurations into executable Python scripts within deterministic container environments, bridging usability and reproducibility. We demonstrate the framework's utility by leveraging multi-temporal remote sensing data across diverse applications, from urban expansion simulation to rooftop solar potential assessment. These applications represent different analytical paradigms of temporal raster prediction and spatial vector analysis, demonstrating that the unified workflow democratizes access to advanced geographic modeling while ensuring reproducibility. Beyond providing an operational platform, we outline a trajectory for modeling infrastructure to evolve from ad-hoc tool collections toward integrated ecosystems capable of supporting future AI-driven modeling automation and AI-assisted scientific discovery.

*Keywords:* Cyberinfrastructure, Geographic Modeling, Web Services, Jupyter, Reproducible Research

*Corresponding author: Min Chen. Email: chenmin0902@163.com
[1]These authors contributed equally to this work.

## 1. Introduction

Geographic modeling has evolved into a data-intensive discipline critical for understanding complex human-environment interactions (Zhou, 2025; Lü et al., 2025). Sophisticated workflows increasingly require integrating heterogeneous computational resources, spanning multi-source datasets, preprocessing algorithms, and simulation models (Liu et al., 2022; Ma et al., 2022; Palomino et al., 2017). However, the practical utilization of these resources is impeded by severe infrastructure fragmentation (Wang et al., 2025). Resources typically reside in isolated systems characterized by incompatible interfaces and disparate authentication protocols (Zhu et al., 2023b). Consequently, researchers are compelled to rely on manual, ad-hoc orchestration, devoting substantial effort to routine technical tasks (Lehmann et al., 2014; Longley and Chen, 2025). This fragmentation impedes research efficiency and creates barriers to reproducibility, as stitched workflows are difficult to document, share, and re-execute across different computing environments (Ma et al., 2025b).

To address these infrastructure challenges, the research community has advanced solutions spanning resource accessibility (Wang et al., 2024), computational scalability (Cavallaro et al., 2022), and execution consistency (Choi et al., 2023; Han et al., 2015). Foundational efforts in service-oriented computing (SOC), driven by OGC specifications (Bröring et al., 2012) and OpenGMS (Wen et al., 2013; Xu et al., 2024a), sought to decouple model execution from local environments. By abstracting resources as web services, this approach eliminates the need for complex local software management (Chen et al., 2020). Advancements in integrated cloud platforms, such as Google Earth Engine (GEE) and CyberGIS, extended this concept by coupling large-scale data repositories with elastic computing resources, thereby resolving the bottlenecks of data transfer and format conversion (Gorelick et al., 2017; Kang et al., 2020). To ensure transparency and generalizability, the reproducible computing paradigm has emerged, centering on the Jupyter ecosystem and Docker containerization. This approach prioritizes the encapsulation of code, data, and runtime environments into self-contained artifacts, ensuring analyses remain verifiable across heterogeneous infrastructures (Kluyver et al., 2016; Boettiger, 2015).

Despite these parallel advancements, these paradigms have matured in architectural isolation, resulting in a disjointed ecosystem where service access, data coupling, and environment reproducibility operate as fragmented layers (Martin et al., 2017). The challenge has shifted from the availability of resources to their orchestration. Researchers are relegated to functioning as human middleware, manually bridging the gaps between service protocols, execution environments, and heterogeneous data formats (Ranatunga et al., 2025). This lack of coordination imposes severe integration overhead, amplifying cognitive load while stifling cross-disciplinary collaboration (Mehmood et al., 2025). Furthermore, this human-dependent workflow prevents the emergence of automated scientific discovery, which requires end-to-end machine-readable coherence across resources, processes, and outputs. Current interoperability gaps inhibit the automated composition and execution of complex modeling workflows (Ballatore et al., 2014; Veenendaal et al., 2016). What remains absent is not specialized tools for isolated aspects of modeling process, but a unified infrastructure pattern that architecturally synthesizes resource access, workflow synthesis, and task execution (Yang et al., 2010; Xia et al., 2018).

To operationalize this vision, this paper proposes a unified infrastructure pattern for geographic modeling and presents OpenGeoLab as its reference implementation. This architectural synthesizes service-oriented resource orchestration, interactive workflow synthesis, and deterministic execution into a coherent ecosystem. Adopting a "clean client" philosophy, the system utilizes a lightweight, middleware-driven Jupyter environment to aggregate distributed resources, such as those within the OpenGMS ecosystem, without requiring local dependency management. Users interact through specification-driven graphical interfaces that bridge the usability-reproducibility gap by transparently translating visual configurations into executable Python scripts. Our contribution extends beyond the development of a specific platform to articulating a design pattern for the post-desktop era. This work demonstrates how architectural coordination can democratize access to computational resources while laying the machine-readable foundations requisite for future automated scientific discovery.

The remainder of this paper is organized as follows. Section 2 reviews related works, specifically examining the architectural limitations within current service-oriented, cloud-based, and reproducible computing paradigms. Section 3 details the system architecture and key implementation techniques of OpenGeoLab, describing how distributed resource access and execution environments are coordinated. Section 4 validates the proposed infrastructure through two case studies to demonstrate how the unified workflow reduces technical barriers while ensuring reproducibility. Section 5 discusses the broader implications of this approach, critically assessing the architecture's generalizability and limitations while charting directions for future automated scientific discovery. Finally, Section 6 offers concluding remarks.

## 2. Related Works

### *2.1. Geographic Modeling Resource Sharing and Servitization*

The paradigm of geographic resource sharing has evolved from static artifact repositories to dynamic service ecosystems (Chen et al., 2021; Zhu et al., 2024; Li et al., 2013). Initial efforts focused on metadata-driven discovery, establishing centralized repositories where researchers could catalogue and retrieve shared resources. Prominent initiatives, such as the Community Surface Dynamics Modeling System (CSDMS) Model Repository (Overeem et al., 2013) and HydroShare (Tarboton et al., 2024), successfully standardized the publication of Earth surface models and hydrological resources. While these platforms significantly improved resource visibility, early iterations primarily functioned as storage archives. Consequently, the burden of execution, including dependency resolution and software compilation, remained with the end-user's local environment.

To mitigate local deployment constraints, the community advanced toward SOC. By encapsulating models and algorithms as network-accessible web services, approaches based on OGC web processing service standards and the OpenGMS effectively decoupled model execution from local hardware (Chen et al., 2020; Ma et al., 2025a). Other platforms, including newer iterations of CSDMS (Overeem et al., 2013) and HydroShare (Tarboton et al., 2024), similarly introduced web-based execution capabilities to allow users to configure models through browser interfaces. However, a critical architectural limitation persists as these capabilities are typically

implemented as isolated systems. Access is mediated through separate, dedicated web portals, where each platform enforces distinct authentication mechanisms and interface protocols. This portal-centric model solves the accessibility problem for individual tools but introduces interoperability barriers (Koo and Kim, 2022; Oladosu et al., 2022). Researchers attempting to compose workflows spanning diverse resources are forced to manually navigate multiple platforms and transfer data between isolated systems, creating friction that hinders the development of integrated, cross-domain modeling workflows.

### *2.2. Integrated Cloud Platforms and Data-Computation Coupling*

While service-oriented architectures addressed the execution of individual models, they did not inherently resolve the data transfer bottlenecks associated with large-scale geospatial analysis. Integrated cloud platforms emerged by tightly coupling massive data repositories with elastic processing infrastructure within unified environments. GEE pioneered this model by delivering a planetary-scale analysis environment where multi-petabyte raster collections are processed through parallelized pipelines (Gorelick et al., 2017). This integration of data catalogs and processing algorithms successfully demonstrated the potential of cloud-native geospatial computing. However, GEE adopts the architecture primarily tailored for raster-based remote sensing workflows. This design optimization restricts the direct integration of external, process-based simulation models or custom executables that do not match its specific parallel processing paradigms (Crego et al., 2022), thereby limiting its applicability for broader geographic modeling tasks that involve complex vector interactions or scientific modeling codes.

CyberGIS represents a complementary approach by bridging high-performance computing access with geospatial analysis capabilities (Kang et al., 2020; Wang, 2010). By providing middleware that connects interactive Python environments with supercomputing resources, CyberGIS enables researchers to execute computationally intensive tasks that exceed standard cloud capacities. Unlike the specialized execution model of GEE, CyberGIS offers greater flexibility by connecting to diverse external data repositories and supporting custom code execution. Nevertheless, significant usability barriers persist. These platforms typically function as high-code environments, requiring researchers to possess advanced programming proficiency to manually locate resources and configure analysis parameters via scripts (Hou et al., 2025). The lack of interactive, low-code mechanisms to assist in workflow construction means that the cognitive load of technical coordination still falls heavily on the researcher (Bucchiarone et al., 2025), limiting accessibility for domain experts who lack extensive software engineering training.

### *2.3. Computational Notebooks and Reproducible Environments*

While cloud platforms optimized data-intensive execution, the challenge of creating transparent, shareable workflow artifacts persisted. Computational notebooks emerged as a standard for scientific reproducibility by enabling researchers to combine executable code, narrative documentation, and visualization results within a single document (Siddik et al., 2025; Samuel and Mietchen, 2024; Ben Guebila et al., 2022). The Jupyter ecosystem, in particular, has become the dominant interface for

this literate computing paradigm (Kluyver et al., 2016; Perkel, 2018). Extensions such as JupyterHub and MyBinder have further expanded this capability to support multi-user collaboration and on-demand cloud sessions (Beg et al., 2021; Perkel, 2018), allowing researchers to publish complete analytical workflows that peers can inspect and re-execute.

Sharing code alone is insufficient for reproducibility, as consistent execution necessitates an identical software environment (Zhu et al., 2023a). While initiatives like Whole Tale (Brinckman et al., 2019) leverage Docker to package these environments, they shift the burden of dependency resolution to the user. In the geospatial domain, this manual configuration is fragile due to the coupling between core libraries like GDAL and PROJ. Furthermore, although platforms like ModelWhale(`https://www.modelwhale.com`) simplify environment provisioning, they prioritize generic data science workflows, lacking the specialized orchestration capabilities needed to integrate complex geographic simulation models.

### *2.4. Integrated Geographic Modeling Infrastructure*

The review of existing paradigms reveals a critical architectural gap. While service-oriented architectures, cloud platforms, and containerization technologies have independently advanced remote execution, data integration, and environment reproducibility, they remain functionally disconnected. Currently, constructing complete workflows requires researchers to manually bridge these disparate systems. Researchers must navigate service portals to locate resources, switch to separate notebook environments for analysis development, and configure container images to ensure consistency. This requirement for manual coordination forces domain scientists to act as system integrators, managing authentication protocols and data transfers across disconnected components. Such fragmentation imposes significant cost that constrains the scalability of research and impedes the transition from data discovery to reproducible publication (Hernandez and Colom, 2025).

OpenGeoLab responds to this challenge by implementing a unified infrastructure pattern, where service-oriented resource access, interactive code synthesis, and containerized execution function as interdependent architectural components. Rather than requiring users to assemble these tools, the platform consolidates them within a cohesive JupyterLab workspace. The system aggregates distributed resources, such as those from the OpenGMS ecosystem, and embeds service discovery directly within the analysis interface. By employing interactive configuration forms that automatically generate executable Python code, the platform provides accessible entry points for domain scientists while preserving the computational transparency required for peer review. Furthermore, pre-configured Docker environments tailored to geospatial dependencies eliminate the manual resolution of conflicts that impede workflow sharing. This architectural integration transforms the modeling process from a disjointed assembly of tools into a continuous workflow where resource discovery, analysis configuration, and environment management operate as a unified computational ecosystem.

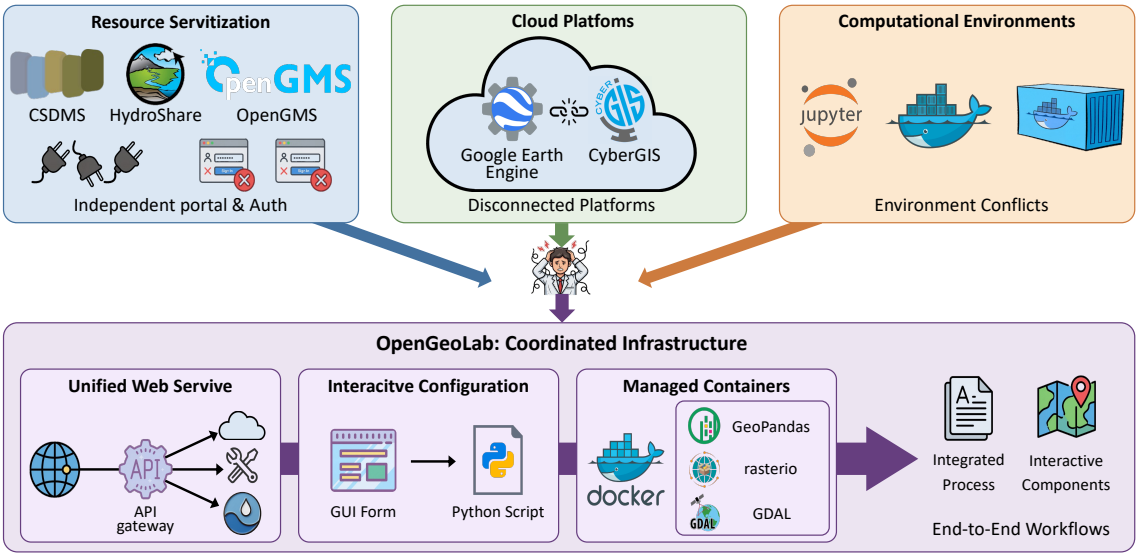


Figure 1. **Conceptual framework illustrating the transition from fragmented toolchains to a unified geographic modeling infrastructure**. The upper panel depicts the current landscape of "architectural isolation," where resource repositories, cloud platforms, and computational environments function as disconnected silos with incompatible interfaces. The lower panel presents the OpenGeoLab solution, which architecturally coordinates three interdependent components: unified web service gateways, interactive code synthesis mechanisms, and managed containerized environments. This integration resolves underlying dependency conflicts to enable seamless, end-to-end reproducible workflows.

## 3. OpenGeoLab: System Design and Implementation

### *3.1. System Overview and Architecture*

OpenGeoLab implements the unified infrastructure approach through a three-tier architecture that coordinates user interaction, logic orchestration, and resource execution (Figure 2). This layered design ensures the separation of concerns between visual configuration, middleware integration, and physical computation.

The presentation layer functions as the primary user workspace, implemented as a customized JupyterLab interface with domain-specific extensions. As illustrated in Figure 2, this layer integrates standard notebook programming capabilities with graphical model configuration panels. Users interact with these panels to define parameters visually, which are then translated into executable scripts within the notebook cells. This design maintains a "clean client" environment where the frontend handles interaction and visualization but delegates heavy computational tasks to the underlying layers.

The middleware layer serves as the system's integration hub and service wrapper, composed of three coordinated modules. The API middleware acts as the core orchestration engine, implementing an adapter design pattern to standardize communication between the frontend and heterogeneous backend resources. It receives unified requests from the presentation layer and translates them into protocol-specific service calls. To ensure execution reproducibility, the containerized environment manager dynamically provisions isolated user compute sessions (depicted as containerized Python environments), which provide the deterministic runtime required

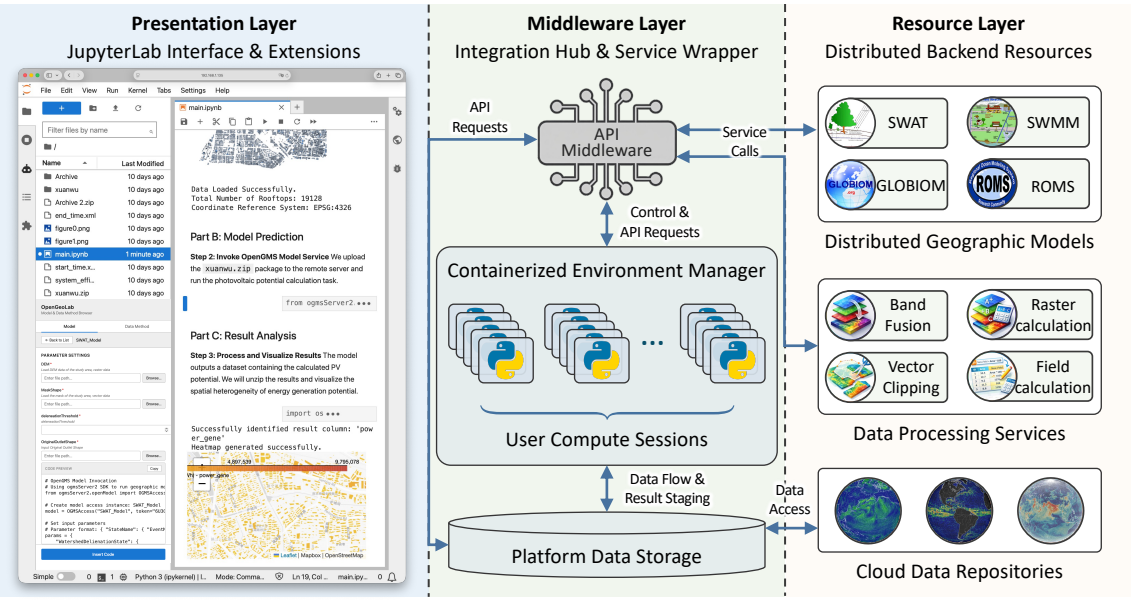


Figure 2. **Hierarchical system architecture of OpenGeoLab coordinating user interaction, logic orchestration, and resource execution.** The architecture is organized into three interdependent layers: (1) The presentation layer provides a customized JupyterLab interface equipped with domain-specific extensions for resource discovery and visual parameter configuration; (2) The middleware layer functions as an integration hub, comprising the API middleware for service wrapping, the containerized environment manager for provisioning isolated user compute sessions, and platform data storage for data staging; (3) The resource layer federates heterogeneous distributed capabilities, including geographic models (e.g., SWAT, SWMM), data processing algorithms, and cloud repositories, which are exposed through standardized service interfaces.

to execute the synthesized code. Additionally, the data storage manages data persistence and staging, facilitating distinct data flows between user sessions and external repositories.

The resource layer aggregates distributed backend resources into a unified ecosystem. In the current implementation, this layer includes distributed geographic models, data processing services, and cloud data repositories. These resources are deployed on distributed servers but are exposed to the middleware through standardized network interfaces. This organization ensures that distributed resources appear locally available to the user while maintaining the performance advantages of server-side execution.

### *3.2. Service-Oriented Resource Integration and Flow Orchestration*

OpenGeoLab aggregates distributed geographic capabilities by exposing them as locally invokable functions within the JupyterLab environment. In the current reference implementation, the platform integrates with the OpenGMS ecosystem, creating a unified access point for approximately 3,100 geographic models, alongside over 1,200 data processing algorithms extracted from libraries such as SAGA GIS, WhiteBox, and GDAL. Crucially, this extensive repository operates on a community-driven contribution model. Rather than relying solely on centralized maintenance, the ecosystem leverages a crowdsourcing paradigm where researchers and model developers worldwide voluntarily contribute, encapsulate, and share their computational

resources to foster open science.

To bridge the technical gap between these diverse, user-contributed models and modern web services, the backend infrastructure employs a distributed service encapsulation strategy. As established in prior research (Chen et al., 2020; Xu et al., 2024), heterogeneous models contributed by the community are encapsulated into standardized service component packages using the OpenGMS Wrapper System. These packages are dynamically deployed across a network of distributed computing nodes, which abstracts the specific runtime environments and binary dependencies of each model (Yue et al., 2016). Consequently, diverse modeling resources are exposed as uniform RESTful endpoints utilizing standardized parameter schemas defined in JSON format. To manage this heterogeneity at the client side, we employ a dynamic binding mechanism. Instead of hardcoding static client libraries, the Python SDK queries the remote service registry at runtime to construct callable methods matching the latest service metadata. This ensures that the frontend automatically adapts to updates in backend resources without requiring client-side software patches.

The middleware functions as the architectural bridge, coordinating the interaction lifecycle through the modules illustrated in the center of Figure 3. When a user invokes a function via the unified interface (Arrow 1), the standardized adapters within the middleware intercept the request. The auth & protocol translation module then dynamically injects session-based security credentials and translates the user's Python parameter dictionary into the specific JSON API payload required by the backend service (Arrow 3). Furthermore, the middleware handles the asynchronous nature of geographic modeling. Since simulations may run for extended periods, the system employs a non-blocking invocation pattern. The middleware submits the job to the backend computing nodes, monitors the remote execution status, and propagates any exceptions to the frontend, thereby ensuring robust error handling across the distributed environment.

A critical design feature illustrated in Figure 3 is the decoupling of control flow from data flow. To maintain the "Clean Client" philosophy, the architecture implements a data flow orchestration mechanism that relies on a pass-by-reference strategy. Users orchestrate these complex workflows by chaining unified SDK calls in the notebook, where each step triggers remote execution while preserving the logical transparency of local scripting:

1. Data Staging and Reference Generation: Data management is integrated directly into the JupyterLab file browser, which displays both local notebook files and remote platform object storage directories in a unified tree view. When users select a dataset, the system generates a persistent reference URL (Arrow 2a) rather than downloading the file content.
2. Remote Execution: When invoking a model, users pass these storage URLs as input parameters. The backend service retrieves the actual data directly from the storage system (Arrow 4). This ensures that computation moves to the data location, leveraging high-performance server I/O bandwidth.
3. Result Persistence: Upon completion, output files are written back directly to the cloud storage (Arrow 5). The SDK returns only the reference URLs and execution metadata to the user's workspace (Arrow 6), which can be immediately used as inputs for subsequent operations.

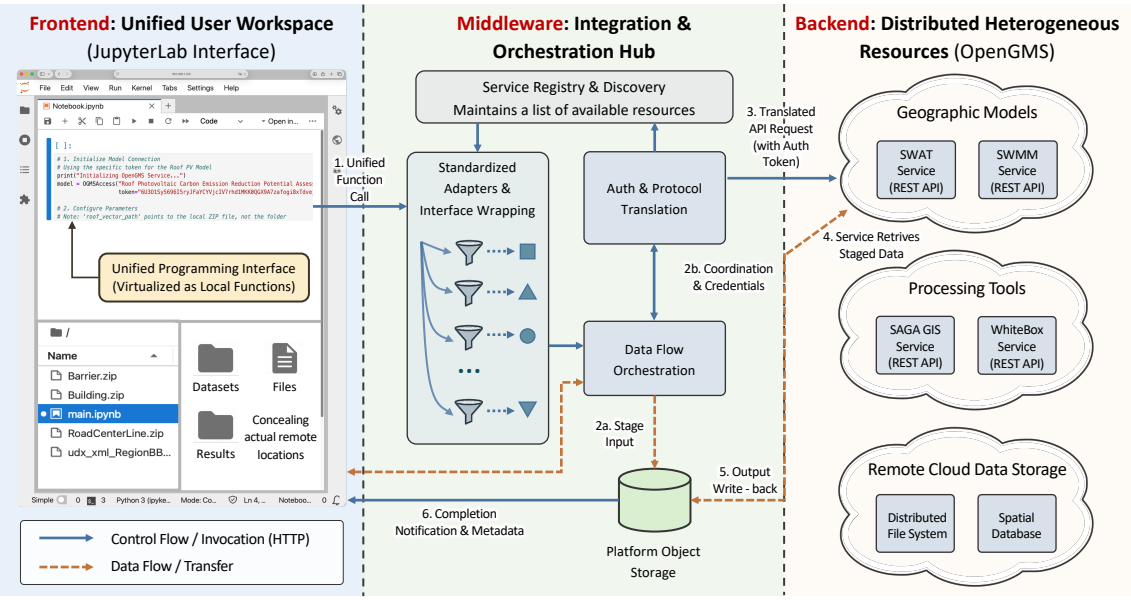


Figure 3. **Mechanism of service-oriented resource integration and flow orchestration.** The middleware intercepts unified function calls from the JupyterLab frontend (1) to coordinate credential injection (2b) and data staging (2a). It translates these into authenticated API requests (3) directed at distributed backend resources. The architecture explicitly decouples control flow (blue solid lines) from data transfer (orange dashed lines), ensuring that heavy geospatial data is accessed (4) and persisted (5) directly via the platform object storage, with only completion metadata returned to the user (6).

### *3.3. Interactive Graphical Interface for Service Invocation and Code Synthesis*

To address the cognitive barriers inherent in raw programmatic access, OpenGeoLab implements an interactive code synthesis mechanism. Technically implemented as a native JupyterLab extension, this module integrates into the researcher's workspace, functioning as a bridge that transforms visual configuration intents into transparent, executable script artifacts (Figure 4).

The interaction workflow begins within the notebook environment (Step 1), where the extension provides a collapsible sidebar for resource browsing (Step 3). This component operates on a specification-driven rendering engine. When a user selects a model service, the extension retrieves its parameter schema (defined in JSON) from the middleware and dynamically instantiates a context-aware configuration form. This mechanism ensures that the interface adapts automatically to the diverse input requirements of heterogeneous models without requiring hardcoded frontend logic.

The extension employs a semantic schema mapping strategy to translate abstract parameter definitions into concrete interface widgets. As detailed in Figure 4, the system dynamically maps distinct data types to appropriate controls to ensure configuration validity: file inputs are rendered as cloud-integrated selectors, allowing users to browse the visual tree view (Step 4) to target remote datasets, which the interface automatically resolves into the persistent reference URLs required by the backend; concurrently, numeric and categorical parameters are instantiated as text fields (Step 5) and dropdown menus, respectively, streamlining the configuration of diverse model attributes without exposing the complexity of underlying data structures. Descriptions from the service specification are presented as interactive

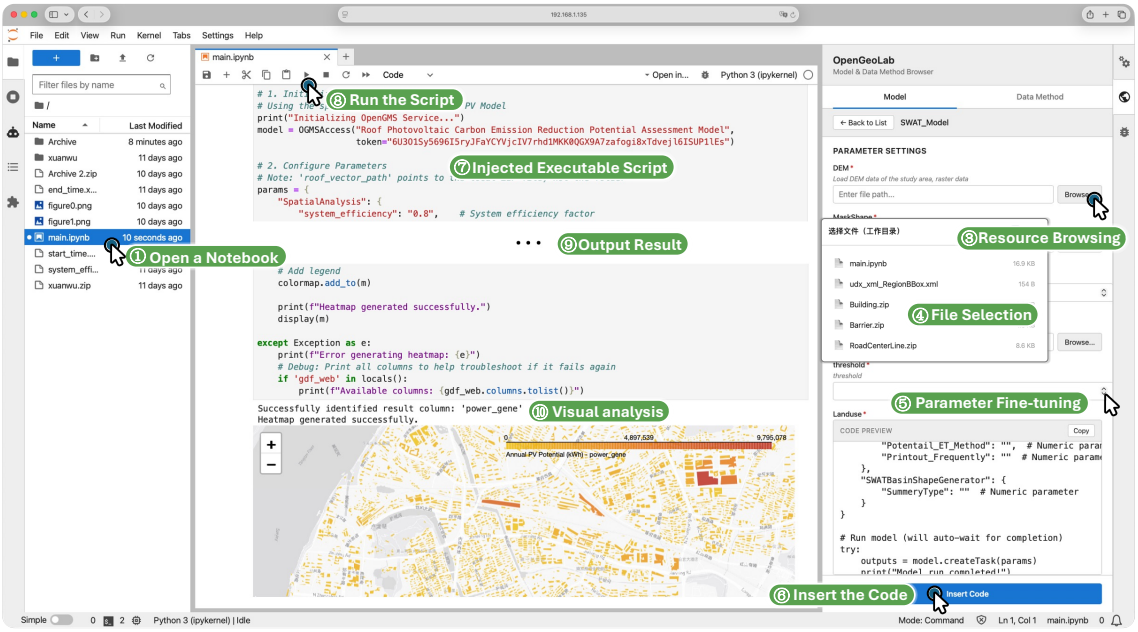


Figure 4. **Interactive code synthesis workflow bridging graphical usability and computational reproducibility.** The interface enables users to browse resources and configure parameters through a specification-driven sidebar. Crucially, instead of executing operations internally, the system transparently synthesizes visual inputs into an executable Python script injected into the notebook. This mechanism ensures that model execution and subsequent visual analysis rely on auditable script records rather than opaque interface states.

tooltips, providing immediate semantic context for each parameter.

The core innovation of this extension is the "GUI-to-Code" transformation. Upon finalizing the configuration, the user triggers the "Insert Code" action (Step 6). Unlike "black box" interfaces that execute operations internally, the extension synthesizes a syntactically complete Python script and injects it directly into the active notebook cell (Step 7). This generated code explicitly constructs the parameter dictionary, imports the necessary SDK modules, and initializes the service object.

This design shifts the role of the GUI from an opaque execution engine to easy-to-use tool, achieving three architectural objectives:

1. Transparency: The actual model invocation occurs only when the user explicitly runs the injected script (Step 8). This ensures that the entire analytical process is recorded as code within the notebook document, rather than being lost as transient interface states.
2. Feedback Loop: Upon execution, results are returned and rendered immediately in the output cell (Step 9). This enables researchers to perform instant Visual Analysis (Step 10), creating a tight feedback loop between configuration, execution, and verification.
3. Reproducibility: By persisting logic as code, the extension ensures that workflows remain reproducible and auditable by peers, strictly adhering to the principles of open science.

### *3.4. Containerized Environment Management for Reproducible Workflows*

While the code synthesis mechanism ensures logical transparency, it does not guarantee runtime reproducibility due to the persistent complexity of managing geospatial software dependencies. Unlike general data science workflows, geographic modeling depends heavily on system-level binary libraries, where issues such as the strict compilation compatibility between GDAL and PROJ often render workflows fragile. A script functioning on one researcher's local machine frequently fails on another due to minor discrepancies in underlying C++ libraries or operating system headers. To resolve this, OpenGeoLab implements a standardized containerization strategy that provides pre-configured, deterministic execution environments.

As illustrated in the upper section of Figure 5, the platform manages a curated container image registry, allowing users to select from specialized profiles tailored to specific research needs, such as "Geo-Basic" or "Geo-ML". Upon selection, the system instantiates an active Docker container instance that encapsulates a comprehensive multi-layer runtime environment. This environment integrates an interface layer providing the JupyterLab server alongside system terminals and integrated Geo-Visualizers; a harmonized geospatial software stack where conflicting dependencies between core drivers (e.g., GDAL, PROJ) and high-level tools (e.g., GeoPandas, xarray) are pre-validated; and an isolated Python runtime kernel that executes user code without interference from the host system. By bundling this entire dependency tree, the platform ensures a conflict-free state that is difficult to achieve with standard local package managers.

The architecture employs a storage-compute decoupling mechanism to manage data persistence. As shown in the right section of Figure 5, the container functions as an on-demand unit that is recycled after the session, whereas the user's workspace is maintained in persistent cloud storage. By dynamically mounting this storage volume, the system strictly segregates and preserves user notebooks, project datasets, and model results. This mechanism ensures that research artifacts persist across sessions even as the underlying computational container is updated or replaced.

This containerization strategy facilitates a robust collaboration paradigm by shifting the unit of sharing from merely code to code combined with its execution environment. As depicted in the bottom of Figure 5, when researcher A shares a workflow with researcher B, the transfer involves not only the Notebook File but also the specific Image ID. By launching the notebook against this unique ID, researcher B creates a container instance with the exact same software stack configuration. This encapsulated transfer ensures identical execution behavior across different users and hardware, addressing the requirement for reproducibility in collaborative geographic science.

### *3.5. System Implementation*

OpenGeoLab is implemented as a cohesive web-based platform that operationalizes the three-tier architecture described above. The system is engineered as a unified integrated development environment (IDE) specialized for geospatial research rather than functioning as a loosely coupled set of tools. Its implementation core centers on a metadata-driven integration engine. Unlike static interfaces that require manual updates when new models are added, the platform's implementation relies on

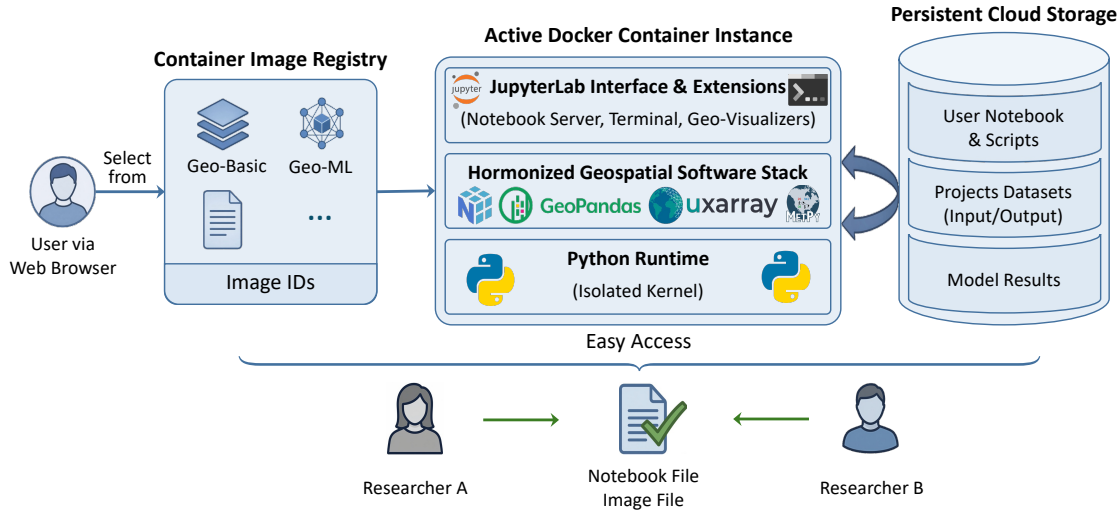


Figure 5. **Deterministic execution architecture via standardized containerization.** The system manages a registry of pre-validated image profiles to provision isolated Docker instances with harmonized geospatial software stacks. A storage-compute decoupling mechanism ensures that user notebooks and datasets remain in persistent cloud storage while the computational container is ephemeral. The bottom panel illustrates the reproducible collaboration paradigm: by sharing the notebook file alongside the specific image reference, the system guarantees that subsequent users replicate the identical runtime environment.

real-time synchronization with the backend service registry. When the middleware detects new model specifications from the unified service ecosystem, the frontend interface automatically instantiates the corresponding configuration panels without requiring system downtime or code recompilation.

The system design is realized through a set of functional interface components illustrated in Figure 6, which guide the user through the complete modeling lifecycle. The workflow initiates at the environment management interface (Figure 6a), which implements the containerization strategy. Here, users instantiate new research environments by selecting from pre-defined profiles, such as standard geospatial stacks or deep learning-optimized environments, directly triggering the backend container manager to provision the specific runtime required for the task. To support organized research, the system utilizes a dedicated project governance module (Figure 6b). This interface tracks the lifecycle of independent research tasks and displays critical metadata, while managing persistent storage associations to ensure that different research contexts remain isolated.

Within the active JupyterLab environment, the implementation features a interactive configuration workspace (Figure 6c). As illustrated in the configuration sidebar, abstract model parameters—ranging from data inputs to numerical constraints—are mapped to specific interface controls. The realization of the "GUI-to-Code" mechanism is visible in the notebook cell, where the system has synthesized the executable service invocation script based on these visual inputs. Finally, the analytical visualization (Figure 6d) demonstrates the system's integration with geospatial visualization libraries. The execution results are rendered as interactive maps directly within the notebook, verifying that the data retrieved from the remote computation has been successfully staged and visualized in the local environment for immediate analysis.

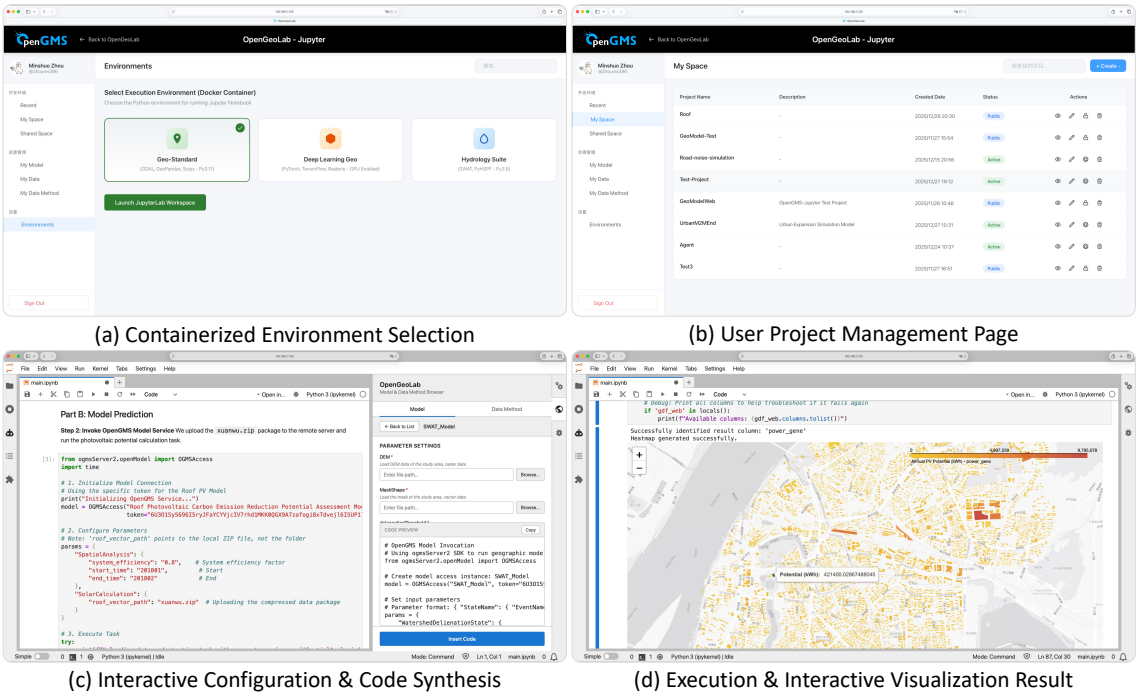


Figure 6. **Operational interface implementation facilitating the complete geographic modeling lifecycle.** The system guides researchers through four integrated stages: (a) The environment provisioning dashboard, allowing users to instantiate task-specific container profiles (e.g., Geo-Standard, Deep Learning); (b) The project governance module, used to track the status and lifecycle of independent research tasks; (c) The logic synthesis workspace, where the sidebar dynamically maps visual parameters to executable code injected into the notebook; (d) The analytical visualization interface, rendering remote computational results as interactive maps directly within the local session.

## 4. Case Study

To validate the architectural versatility of the proposed unified infrastructure pattern, this study presents two distinct application cases: urban expansion simulation and rooftop solar potential assessment. These cases were selected to represent divergent analytical paradigms within geographic modeling. The first case involves data-driven temporal raster prediction, applying a ConvLSTM deep learning model to simulate urban growth dynamics in Suzhou. The second case involves process-based spatial vector analysis, processing discrete building footprint geometries to quantify photovoltaic potential in Nanjing. Despite their differences in data structures and algorithmic logic, both analyses are orchestrated through the OpenGeoLab environment. These case studies serve to demonstrate how the platform's service integration, interactive code synthesis, and pre-configured environments enable researchers to compose complex, reproducible workflows without the overhead of manual software installation, data format conversion, or cross-platform coordination.

### *4.1. Urban Expansion Simulation in Suzhou*

#### *4.1.1. Problem Context and Modeling Objective*

Simulating spatiotemporal urban dynamics represents a critical challenge in regional planning, requiring the integration of multi-temporal land use data with

complex spatial driving factors. This case study focuses on predicting the urban growth of Suzhou, a rapidly urbanizing metropolis in the Yangtze River Delta, utilizing a comprehensive land use dataset spanning from 2000 to 2017. The analysis employs UrbanM2M (Zhou et al., 2024), a deep learning model based on the Convolutional Long Short-Term Memory (ConvLSTM) architecture. For this specific experimental setup, the model takes the historical land use sequence from 2010 to 2012 as temporal input and incorporates spatial variables, including terrain slope, water bodies, and protected areas, to capture non-linear growth patterns and project the 2013 urban extent.

Conventionally, deploying such deep learning workflows imposes steep technical barriers for geographers, necessitating high-performance GPU infrastructure and the management of complex software environments (e.g., specific CUDA drivers and Python library versions). The objective of this case is to validate how OpenGeoLab's service-oriented architecture abstracts these computational complexities. By enabling researchers to configure the model through a graphical interface and execute it on remote resources, the workflow demonstrates the platform's capacity to democratize access to advanced AI-driven geographic simulation tools while maintaining the transparency of the analytical process.

#### *4.1.2. Workflow Implementation on OpenGeoLab*

The analysis workflow commenced with data access via the platform's integrated cloud storage system. Through the JupyterLab file browser, the user accessed the project directory containing the Suzhou land use dataset. This dataset comprised a comprehensive time series of annual raster images from 2000 to 2017, alongside spatial driving factors such as terrain slope, proximity to town centers, and administrative boundaries. Crucially, these files remained resident in the cloud storage throughout the workflow. By selecting files directly through the interface, the user established persistent references that the platform's middleware accessed during subsequent processing steps. This approach effectively eliminated the overhead of downloading datasets to local machines and re-uploading them to computational environments.

Prior to simulation, the spatial data required preprocessing to ensure format compatibility and spatial alignment. OpenGeoLab facilitates this through a data processing service catalog accessible via a sidebar panel, where methods are organized by functional categories including format conversion, coordinate transformation, and spatial subsetting. In this specific case, input rasters necessitated spatial extraction using the study area boundary to isolate the urban core of Suzhou. Upon selecting the ExtractByMask method from the catalog, the user interacted with a graphical configuration interface to upload the boundary polygon file and specify output paths within the cloud directory. Following these configurations, the platform synthesized Python code to invoke the processing service through the standard SDK. This masking operation executed as a remote service, processing the cloud-stored input files and writing aligned outputs back to the user's data directory without requiring local software installation or data transfer.

With the data prepared, the workflow proceeded to the invocation of the UrbanM2M model through the integrated service catalog. UrbanM2M implements a ConvLSTM architecture that combines convolutional operations for spatial feature

extraction with recurrent mechanisms for temporal dependency modeling, enabling it to capture the complex nonlinear dynamics characteristic of urban growth processes. This model has been deployed on the OpenGMS platform (`https://geomodeling.njnu.edu.cn/modelItem/413abfee-d4a5-4272-bee9-5a1f8cd94fcf`) and is accessible as a remote service through standardized interfaces. When the user selected UrbanM2M from the catalog, the platform retrieved the model's parameter specification from the middleware and dynamically generated a configuration form. This interface presented fields for temporal coverage and spatial extent. Specifically, the user referenced the preprocessed rasters from 2010, 2011, and 2012 as the input sequence and defined 2013 as the target projection year. Subsequently, the platform generated Python code that constructed a parameter dictionary matching these selections and invoked the remote simulation service using file references that the middleware resolved to actual cloud storage locations.

The synthesized code was injected into a new notebook cell, allowing the user to inspect its structure and logic prior to execution. Upon running the code, the UrbanM2M service processed the cloud-referenced data on remote computing resources, applying the trained ConvLSTM model to generate probability maps based on the input sequence and allocating simulated urban expansion according to the specified land demand targets. Execution progress was monitored through real-time status updates displayed in the notebook interface. Upon completion, the result file for the predicted 2013 land use was written to the user's cloud directory and became immediately available for visualization and further analysis within the notebook workspace.

#### *4.1.3. Results and Validation*

The simulation yielded the predicted urban extent map for 2013, projecting spatial expansion patterns across the study area based on the antecedent three-year sequence. These results were rendered directly within the notebook workspace using standard Python visualization libraries, demonstrating the seamless integration of remote computational outputs with local analytical tools. The predicted map exhibited multiple expansion clusters distributed across satellite towns, where new urban developments emerged predominantly at the fringes of existing built-up zones. To assess model performance, the simulated 2013 map was compared against the observed 2013 land use data through pixel-wise classification. This quantitative evaluation identified correctly predicted transitions while delineating specific areas of underestimation and overestimation. Crucially, this validation process was implemented entirely through executable code within the notebook environment, confirming that the model captured primary expansion trends consistent with satellite imagery while ensuring the assessment logic remains transparent and reproducible.

### *4.2. Rooftop Solar Photovoltaic Potential Assessment in Nanjing*

#### *4.2.1. Problem Context and Assessment Objective*

Assessing renewable energy potential in high-density urban environments presents a distinct challenge, requiring the precise integration of discrete vector geometries with physical radiation models. This case study evaluates the rooftop solar photovoltaic potential of Xuanwu District in Nanjing, a densely built urban core comprising

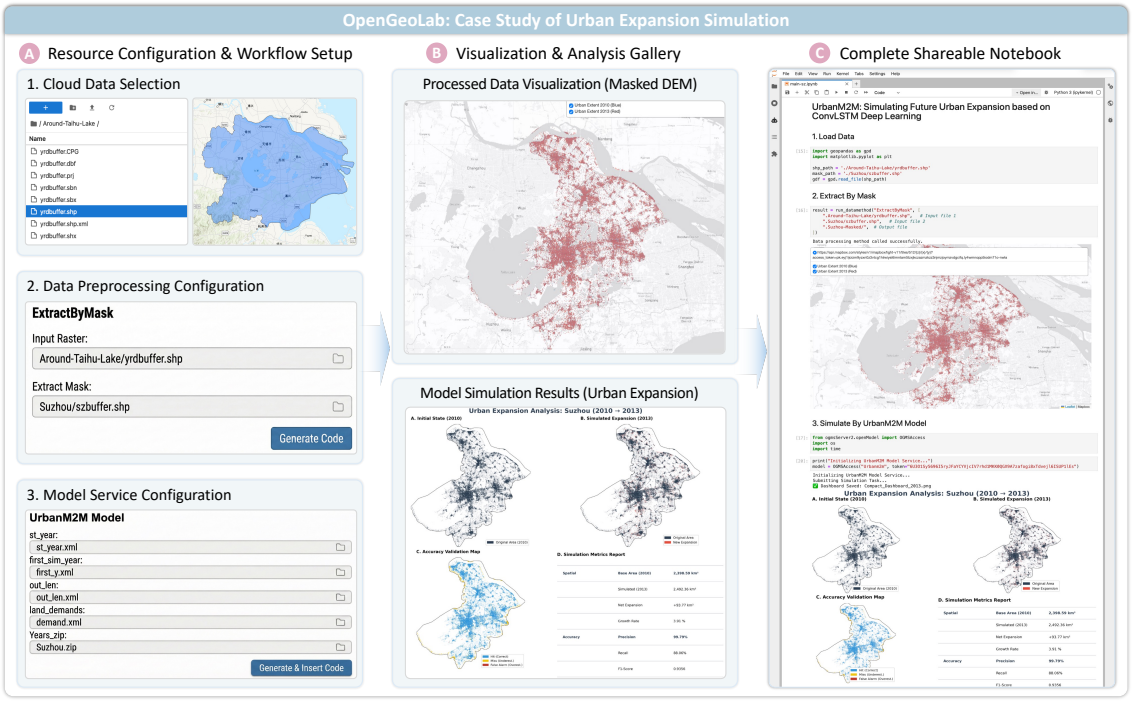


Figure 7. **Validation of the raster-based analytical pipeline through the urban expansion simulation case study.** The workflow illustrates the seamless chaining of distributed services: (A) The resource configuration phase, where users perform cloud data staging, configure the "ExtractByMask" preprocessing service, and parameterize the UrbanM2M deep learning model via specification-driven forms; (B) Visualization outputs, displaying the intermediate masked DEM and the final simulated urban extent map for 2013, accompanied by quantitative accuracy metrics; (C) The complete shareable notebook, which serves as the final reproducible artifact encapsulating the synthesized code, execution provenance, and analytical results.

approximately 19,000 individual building footprints. Unlike the data-driven approach in the previous case, this analysis employs a physics-based solar radiation model (Zhong et al., 2021). The objective is to quantify the annual electricity generation capacity for each building by synthesizing geometric attributes derived from footprint polygons, local meteorological irradiance data, and photovoltaic system efficiency parameters.

The assessment produces spatially explicit results that map potential energy output at the level of individual structures. From an architectural perspective, this workflow serves to validate the platform's capability to orchestrate vector-based spatial analysis. It demonstrates how OpenGeoLab coordinates the processing of irregular geometric features with remote simulation services, providing a necessary counterpoint to the raster-centric workflow illustrated in the urban expansion case.

### *4.2.2. Workflow Implementation on OpenGeoLab*

The analytical workflow start with data access within the platform's integrated environment. Through the JupyterLab file browser, the user located the city-scale Nanjing building footprint dataset and the administrative boundary file for Xuanwu District, both stored in the personal cloud directory. To isolate the specific study area, a spatial clipping operation was performed to extract the building polygons

falling within the district boundaries. Subsequently, to ensure data integrity prior to heavy computation, the user loaded the clipped geometry using the local GeoPandas library within the containerized environment. This step allowed for a preliminary visualization of the approximately 19,000 polygon features, confirming that the spatial extent correctly covered the intended study area. This transition between cloud storage, service-based preprocessing, and local Python libraries validates the platform's ability to support interactive data exploration before committing to computationally intensive service invocations.

Prior to the solar potential assessment, the vector data necessitated geometric preprocessing. Specifically, the workflow required deriving rooftop area values from the raw polygon geometries. Instead of performing this calculation locally, the user accessed the "Vector Attribute Calculation" service from the processing catalog. Through the graphical configuration interface, the user specified the target geometry field and defined the coordinate system projection rules to ensure accurate area measurement. Upon confirmation, the platform synthesized the Python code to invoke this remote service. The operation executed on the backend infrastructure, reading the vector data from cloud storage, computing area values for each building polygon, and generating an enriched dataset with new attribute fields. This process demonstrates how the system decouples data processing logic from the local environment, avoiding the need for manual installation of complex spatial analysis libraries.

With the data prepared, the workflow proceeded to the invocation of the Roof Photovoltaic Carbon Emission Reduction Potential Assessment Model. This specific computational resource, which had been encapsulated and deployed on the OpenGMS platform (`https://geomodeling.njnu.edu.cn/modelItem/f2e45fcd-4a60-49cb-9f17-59215d456594`), calculates annual electricity generation potential based on rooftop area, solar irradiance data for the study region, and photovoltaic system efficiency parameters. When the user selected this model from the integrated catalog, the middleware dynamically rendered a configuration form based on the service specification. The interface presented fields for critical physical parameters, including the system efficiency coefficient and temporal coverage. The user specified an efficiency value of 0.8 and defined the temporal range for the annual calculation. The preprocessed vector dataset was referenced directly via the cloud file selector, eliminating manual path specification.

The platform's code generation mechanism then translated these configuration states into an executable Python script. This synthesized code imported the service SDK, constructed the parameter dictionary, and invoked the remote assessment service using persistent references to the cloud-stored data. Upon execution, the backend service processed the building dataset, calculating the potential for each rooftop and returning the results as a further enriched vector dataset. Each building polygon in the result file included a new attribute field quantifying its annual generation potential in kilowatt-hours (kWh). Finally, the results were visualized directly within the notebook workspace using the Folium mapping library. The user generated an interactive choropleth map where building polygons were color-coded from yellow to red based on their potential values. This spatial representation effectively revealed the heterogeneity of energy potential across the district, highlighting the seamless integration of remote model execution with local, interactive geospatial visualization.

#### *4.2.3. Results and Spatial Patterns*

The assessment successfully quantified the annual electricity generation potential for approximately 19,000 individual building footprints within the Xuanwu District. The results exhibited significant heterogeneity, with individual rooftop potentials ranging from less than 100 kWh for compact residential structures to several gigawatt-hours for large-scale commercial complexes. These results were rendered directly in the notebook as an interactive choropleth map, where vector polygons were thematically color-coded based on their calculated generation capacity. This visualization confirmed the platform's ability to seamlessly stage and render high-density vector datasets returned from remote computational nodes.

The spatial analysis revealed distinct patterns correlated with urban morphology. High-potential buildings formed significant clusters in commercial districts along major transportation corridors, where large retail centers and office complexes provided extensive rooftop surface areas. Similarly, educational zones, characterized by university campuses and institutional facilities, functioned as major potential energy hubs. In contrast, residential neighborhoods were characterized by a more distributed pattern, presenting lower individual potential values but contributing substantial aggregate capacity due to high building density.

Crucially, the platform enabled researchers to perform exploratory spatial data analysis directly within the notebook environment. Users could interactively query the result dataset to examine attribute distributions across different building types and locations without exporting data to external desktop GIS software. This workflow demonstrates that the unified infrastructure not only streamlines the execution of complex physical models but also supports the immediate, interactive interpretation of vector-based geographic phenomena.

## 5. Discussion

This study demonstrates the architectural capacity of the proposed framework to synthesize diverse geographic modeling approaches into a seamless operational environment. The seamless coordination of heterogeneous resources—spanning data-intensive processing and process-based simulations—provides empirical evidence that the field's fragmentation is amenable to unified architectural solutions. The system proves that by abstracting distributed services and encapsulating execution dependencies, the burden of technical integration is effectively shifted from the individual researcher to the middleware layer. This shift ensures that the complexity of cross-domain modeling is handled architecturally rather than manually, confirming the viability of a unified pattern for the next generation of geospatial computing.

This architectural perspective suggests that advancing cloud-native geographic computing may benefit less from incrementally improving individual platform capabilities than from reconceiving how distributed resources, user interfaces, and execution contexts relate to one another as coordinated infrastructure. The consistency observed across our case studies, where fundamentally different modeling tasks operated through uniform workflows, indicates that such coordination, once established, can accommodate diverse analytical approaches without requiring task-specific toolchains.

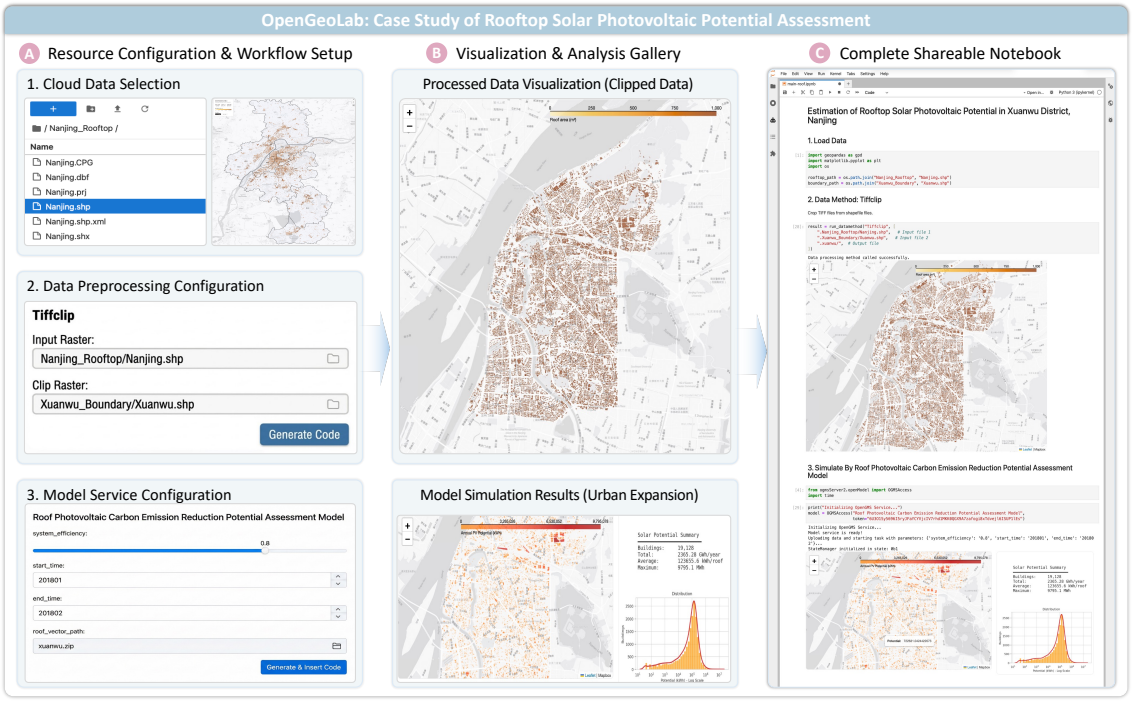


Figure 8. **Validation of the vector-based analytical pipeline through the rooftop solar photovoltaic potential assessment.** The workflow demonstrates the orchestration of discrete geometric analysis: (A) The resource configuration phase, where users select building footprint shapefiles, configure geometric preprocessing services, and parameterize the physics-based radiation model using specific efficiency coefficients; (B) Visualization outputs, displaying the spatial heterogeneity of annual generation potential across the district, complemented by statistical distribution histograms; (C) The complete shareable notebook, which captures the end-to-end vector processing logic as a reproducible document.

While the proposed architecture offers a robust framework for integration, the current reference implementation faces specific constraints regarding resource diversity and operational resilience. The system presently interfaces exclusively with the OpenGMS ecosystem, limiting the available analytical capabilities to the models hosted within that specific repository. However, the system design mitigates this dependency through the middleware adapter Pattern described in Section 3. Theoretically, this enables the federation of heterogeneous backends, such as OGC Web Processing Services or other domain repositories, by developing specific interface adapters rather than redesigning the core platform. A more immediate operational constraint involves the reliance on network connectivity. As a "clean client" architecture that delegates computation to remote nodes, the system remains vulnerable to service latency or interruptions. Future iterations must address these resilience challenges, potentially by implementing local caching strategies or hybrid execution modes that can fallback to local resources when network services are unavailable.

Beyond addressing these integration and resilience challenges, this infrastructure establishes the necessary technical foundation for AI-assisted geographic discovery (Li et al., 2025). The standardization of heterogeneous models into uniform, machine-readable service interfaces creates a structured environment where autonomous agents can operate effectively (Yang et al., 2025; Ray, 2025; Xu et al., 2024b). Since

the platform already transforms high-level intent into executable Python code, it provides a logical entry point for large language models to orchestrate complex modeling workflows (Chen et al., 2023). Future research will explore how this unified infrastructure can evolve from supporting human-directed simulation to enabling semi-autonomous agents that can reason about spatial data, select appropriate models, and execute analysis pipelines to assist researchers in solving complex environmental challenges.

## 6. Conclusion

This study addresses the persistent fragmentation in geographic modeling infrastructure by proposing a unified infrastructure pattern and implementing OpenGeoLab as its reference realization. By architecturally synthesizing service-oriented resource access, interactive code synthesis, and deterministic containerized execution, the platform resolves the long-standing disconnect between distributed computational capabilities and local analytical environments. The implementation validates that the burden of technical orchestration—ranging from dependency management to cross-platform authentication—can be effectively offloaded from the researcher to the middleware, thereby establishing a "Clean Client" environment that prioritizes scientific inquiry over systems administration. The operational consistency demonstrated across divergent analytical paradigms confirms the generalizability of this architectural approach. Whether applied to data-driven deep learning or process-based physical simulation, the unified workflow successfully bridges the gap between the usability required for broad adoption and the reproducibility demanded by rigorous science. By enforcing a mechanism where visual interactions are transparently translated into executable code, the system ensures that the lowering of technical entry barriers does not compromise the integrity of the research record.

Ultimately, this work suggests a trajectory for the evolution of geospatial computing from a collection of isolated tools toward a coordinated, community-driven ecosystem. While the current implementation focuses on the OpenGMS repository, the underlying design principles provide a scalable blueprint for federating heterogeneous resources. As the discipline advances toward automated scientific discovery, such integrated infrastructure will be essential in transforming massive, distributed computational resources into accessible, actionable geographic knowledge.

## Acknowledgements

We appreciate the detailed suggestions and comments from the editor and anonymous reviewers. We express heartfelt thanks to the other members of the OpenGMS team. This work was supported by the National Natural Science Foundation (NSF) of China (Grant No.42325107).

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## CRediT authorship contribution statement

Peilong Ma: Writing – review & editing, Writing – original draft, Software, Methodology, Investigation, Formal analysis, Conceptualization. Minshuo Zhou: Writing – review & editing, Writing – original draft, Software, Validation, Investigation. Wanhao Li: Writing – review & editing, Validation, Data curation. Wei Xie: Writing – review & editing, Resources, Data curation. Tianyu Sheng: Writing – review & editing, Resources. Yongning Wen: Writing – review & editing, Resources. Songshan Yue: Writing – review & editing, Supervision. Guonian Lv: Supervision. Min Chen: Writing – review & editing, Supervision, Methodology, Funding acquisition, Conceptualization.

## Data availability

Data will be made available on request.

## References


Ballatore, A., Bertolotto, M., Wilson, D.C., 2014. Linking geographic vocabularies through wordnet. Annals of GIS 20, 73–84.

Beg, M., Taka, J., Kluyver, T., Konovalov, A., Ragan-Kelley, M., Thiéry, N.M., Fangohr, H., 2021. Using jupyter for reproducible scientific workflows. Computing in Science & Engineering 23, 36–46.

Ben Guebila, M., Weighill, D., Lopes-Ramos, C.M., Burkholz, R., Pop, R.T., Palepu, K., Shapoval, M., Fagny, M., Schlauch, D., Glass, K., et al., 2022. An online notebook resource for reproducible inference, analysis and publication of gene regulatory networks. Nature methods 19, 511–513.

Boettiger, C., 2015. An introduction to docker for reproducible research. ACM SIGOPS Operating Systems Review 49, 71–79.

Brinckman, A., Chard, K., Gaffney, N., Hategan, M., Jones, M.B., Kowalik, K., Kulasekaran, S., Ludäscher, B., Mecum, B.D., Nabrzyski, J., et al., 2019. Computing environments for reproducibility: Capturing the "whole tale". Future Generation Computer Systems 94, 854–867.

Bröring, A., Stasch, C., Echterhoff, J., 2012. Ogc sensor observation service interface standard, version 2.0. .

Bucchiarone, A., Di Rocco, J., Di Vincenzo, D., Pierantonio, A., 2025. Modeling in jjodel: Bridging complexity and usability in model-driven engineering. arXiv preprint arXiv:2502.09146 .

Cavallaro, G., Heras, D.B., Wu, Z., Maskey, M., Lopez, S., Gawron, P., Coca, M., Datcu, M., 2022. High-performance and disruptive computing in remote sensing: Hdcrs—a new working group of the grss earth science informatics technical committee [technical committees]. IEEE geoscience and remote sensing magazine 10, 329–345.

Chen, M., Lv, G., Zhou, C., Lin, H., Ma, Z., Yue, S., Wen, Y., Zhang, F., Wang, J., Zhu, Z., et al., 2021. Geographic modeling and simulation systems for geographic research in the new era: Some thoughts on their development and construction. Science China Earth Sciences 64, 1207–1223.

Chen, M., Qian, Z., Boers, N., Jakeman, A.J., Kettner, A.J., Brandt, M., Kwan, M.P., Batty, M., Li, W., Zhu, R., et al., 2023. Iterative integration of deep learning in hybrid earth surface system modelling. Nature Reviews Earth & Environment 4, 568–581.

Chen, M., Voinov, A., Ames, D.P., Kettner, A.J., Goodall, J.L., Jakeman, A.J., Barton, M.C., Harpham, Q., Cuddy, S.M., DeLuca, C., et al., 2020. Position paper: Open web-distributed integrated geographic modelling and simulation to enable broader participation and applications. Earth-Science Reviews 207, 103223.

Choi, Y.D., Roy, B., Nguyen, J., Ahmad, R., Maghami, I., Nassar, A., Li, Z., Castronova, A.M., Malik, T., Wang, S., et al., 2023. Comparing containerization-based approaches for reproducible computational modeling of environmental systems. Environmental Modelling & Software 167, 105760.

Crego, R.D., Stabach, J.A., Connette, G., 2022. Implementation of species distribution models in google earth engine. Diversity and Distributions 28, 904–916.

Gorelick, N., Hancher, M., Dixon, M., Ilyushchenko, S., Thau, D., Moore, R., 2017. Google earth engine: Planetary-scale geospatial analysis for everyone. Remote sensing of Environment 202, 18–27.

Han, G., Chen, J., He, C., Li, S., Wu, H., Liao, A., Peng, S., 2015. A web-based system for supporting global land cover data production. ISPRS Journal of Photogrammetry and Remote Sensing 103, 66–80.

Hernandez, J.A., Colom, M., 2025. Reproducible research policies and software/data management in scientific computing journals: a survey, discussion, and perspectives. Frontiers in Computer Science 6, 1491823.

Hou, S., Liang, J., Zhao, A., Wu, H., 2025. Gee-ops: an operator knowledge base for geospatial code generation on the google earth engine platform powered by large language models. Geo-spatial Information Science , 1–22.

Kang, J.Y., Aldstadt, J., Vandewalle, R., Yin, D., Wang, S., 2020. A cybergis approach to spatiotemporally explicit uncertainty and global sensitivity analysis for agent-based modeling of vector-borne disease transmission. Annals of the American Association of Geographers 110, 1855–1873.

Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J., Grout, J., Corlay, S., et al., 2016. Jupyter notebooks–a publishing format for reproducible computational workflows, in: Positioning and power in academic publishing: Players, agents and agendas. IOS press, pp. 87–90.

Koo, J., Kim, Y.G., 2022. Resource identifier interoperability among heterogeneous iot platforms. Journal of King Saud University-Computer and Information Sciences 34, 4191–4208.

Lehmann, A., Giuliani, G., Ray, N., Rahman, K., Abbaspour, K.C., Nativi, S., Craglia, M., Cripe, D., Quevauviller, P., Beniston, M., 2014. Reviewing innovative earth observation solutions for filling science-policy gaps in hydrology. Journal of Hydrology 518, 267–277.

Li, S., Dragicevic, S., Veenendaal, B., Brovelli, M.A., 2013. Theme section" towards intelligent geoprocessing on the web". ISPRS Journal of Photogrammetry and Remote Sensing 83, 138–139.

Li, Z., Ning, H., Gao, S., Janowicz, K., Li, W., Arundel, S.T., Yang, C., Bhaduri, B., Wang, S., Zhu, A.X., et al., 2025. Giscience in the era of artificial intelligence: A research agenda towards autonomous gis. Annals of GIS 31, 501–536.

Liu, X., Chen, M., Claramunt, C., Batty, M., Kwan, M.P., Senousi, A.M., Cheng, T., Strobl, J., Cöltekin, A., Wilson, J., et al., 2022. Geographic information science in the era of geospatial big data: A cyberspace perspective. The Innovation 3.

Longley, P., Chen, M., 2025. Smart data, information geographies and intelligent data services. Information Geography 1, 100013.

Lü, G., Li, X., Chen, M., Zhou, L., Yue, S., Zhang, X., Wu, M., Wang, J., Yu, Z., Yuan, L., 2025. Towards information geography in ternary space. Information Geography , 100024.

Ma, P., Chen, M., Zhang, S., Zhu, Z., Qian, Z., Ma, Z., Zhang, F., Li, W., Yue, S., Wen, Y., 2025a. Facilitating sensitivity analysis of hydrological models through knowledge-driven configuration and distributed online model services. Journal of Hydrology , 133406.

Ma, P., Tao, F., Gao, L., Leng, S., Yang, K., Zhou, T., 2022. Retrieval of fine-grained pm2. 5 spatiotemporal resolution based on multiple machine learning models. Remote Sensing 14.

Ma, Z., Li, H., Zhang, K., Wang, J., Yue, S., Wen, Y., Lü, G., Chen, M., 2025b. Knowledge co-creation during urban simulation computation to enable broader participation. Sustainable Cities and Society 118, 105994.

Martin, P., Chen, Y., Hardisty, A., Jeffery, K., Zhao, Z., 2017. Computational challenges in global environmental research infrastructures, in: Terrestrial Ecosystem Research Infrastructures. CRC Press, pp. 305–340.

Mehmood, A., Arif, M., Mehmood, F., 2025. Towards a unified digital ecosystem: The role of platform technology convergence. Electronics 14, 4787.

Oladosu, S.A., Ige, A.B., Ike, C.C., Adepoju, P.A., Amoo, O.O., Afolabi, A.I., 2022. Reimagining multi-cloud interoperability: A conceptual framework for seamless

integration and security across cloud platforms. Open Access Res J Sci Technol 4, 26.

Overeem, I., Berlin, M.M., Syvitski, J.P., 2013. Strategies for integrated modeling: The community surface dynamics modeling system example. Environmental modelling & software 39, 314–321.

Palomino, J., Muellerklein, O.C., Kelly, M., 2017. A review of the emergent ecosystem of collaborative geospatial tools for addressing environmental challenges. Computers, Environment and Urban Systems 65, 79–92.

Perkel, J.M., 2018. Why jupyter is data scientists' computational notebook of choice. Nature 563, 145–147.

Ranatunga, S., Ødegård, R.S., Jetlund, K., Onstein, E., 2025. Use of semantic web technologies to enhance the integration and interoperability of environmental geospatial data: A framework based on ontology-based data access. ISPRS International Journal of Geo-Information 14, 52.

Ray, P.P., 2025. A survey on model context protocol: Architecture, state-of-the-art, challenges and future directions. Authorea Preprints .

Samuel, S., Mietchen, D., 2024. Computational reproducibility of jupyter notebooks from biomedical publications. GigaScience 13, giad113.

Siddik, M.S., Li, H., Bezemer, C.P., 2025. A systematic literature review of software engineering research on jupyter notebook. arXiv preprint arXiv:2504.16180 .

Tarboton, D.G., Ames, D.P., Horsburgh, J.S., Goodall, J.L., Couch, A., Hooper, R., Bales, J., Wang, S., Castronova, A., Seul, M., et al., 2024. Hydroshare retrospective: Science and technology advances of a comprehensive data and model publication environment for the water science domain. Environmental Modelling & Software 172, 105902.

Veenendaal, B., Brovelli, M.A., Wu, L., 2016. Cloud/web mapping and geoprocessing services-intelligently linking geoinformation. ISPRS Journal of Photogrammetry and Remote Sensing 114, 243–244.

Wang, J., Shi, L., Zhang, X., Xu, K., Ma, Z., Wen, Y., Chen, M., 2024. Research on service-oriented sharing and computing framework of geographic data for geographic modeling and simulation. Applied Sciences (2076-3417) 14.

Wang, S., 2010. A cybergis framework for the synthesis of cyberinfrastructure, gis, and spatial analysis. Annals of the Association of American Geographers 100, 535–557.

Wang, S., Huang, X., Zhang, M., Bao, S., Liu, L., Fu, X., Zhang, T., Song, Y., Kedron, P., Wilson, J., et al., 2025. Open science 2.0: revolutionizing spatiotemporal data sharing and collaboration. Computational Urban Science 5, 4.

Wen, Y., Chen, M., Lu, G., Lin, H., He, L., Yue, S., 2013. Prototyping an open environment for sharing geographical analysis models on cloud computing platform. International Journal of Digital Earth 6, 356–382.

Xia, J., Yang, C., Li, Q., 2018. Using spatiotemporal patterns to optimize earth observation big data access: Novel approaches of indexing, service modeling and cloud computing. Computers, Environment and Urban Systems 72, 191–203.

Xu, K., Chen, M., Yue, S., Zhang, F., Wang, J., Wen, Y., Lü, G., 2024a. The portal of opengms: Bridging the contributors and users of geographic simulation resources. Environmental Modelling & Software 180, 106142.

Xu, K., Yue, S., Chen, Q., Wang, J., Zhang, F., Wang, Y., Ma, P., Wen, Y., Chen, M., Lü, G., 2024b. Construction of an open knowledge framework for geoscientific models. Transactions in GIS 28, 154–175.

Yang, C., Raskin, R., Goodchild, M., Gahegan, M., 2010. Geospatial cyberinfrastructure: past, present and future. Computers, Environment and Urban Systems 34, 264–277.

Yang, Y., Ma, M., Huang, Y., Chai, H., Gong, C., Geng, H., Zhou, Y., Wen, Y., Fang, M., Chen, M., et al., 2025. Agentic web: Weaving the next web with ai agents. arXiv preprint arXiv:2507.21206 .

Yue, S., Chen, M., Wen, Y., Lu, G., 2016. Service-oriented model-encapsulation strategy for sharing and integrating heterogeneous geo-analysis models in an open web environment. ISPRS Journal of Photogrammetry and Remote Sensing 114, 258–273.

Zhong, T., Zhang, Z., Chen, M., Zhang, K., Zhou, Z., Zhu, R., Wang, Y., Lü, G., Yan, J., 2021. A city-scale estimation of rooftop solar photovoltaic potential based on deep learning. Applied Energy 298, 117132.

Zhou, C., 2025. Exploring future gis visions in the era of the scientific and technological revolution. Information Geography 1, 100007.

Zhou, Z., Chen, Y., Liu, X., Zhang, X., Zhang, H., 2024. A maps-to-maps approach for simulating urban land expansion based on convolutional long short-term memory neural networks. International Journal of Geographical Information Science 38, 503–526.

Zhu, L., Wang, Y., Kong, Y., Hu, Y., Huang, K., 2024. A containerized service-based integration framework for heterogeneous-geospatial-analysis models. ISPRS International Journal of Geo-Information 13, 28.

Zhu, Z., Chen, M., Qian, Z., Li, H., Wu, K., Ma, Z., Wen, Y., Yue, S., Lü, G., 2023a. Documentation strategy for facilitating the reproducibility of geo-simulation experiments. Environmental Modelling & Software 163, 105687.

Zhu, Z., Chen, M., Sun, L., Qian, Z., He, Y., Ma, Z., Zhang, F., Wen, Y., Yue, S., Lue, G., 2023b. Reproducing computational processes in service-based geo-simulation experiments. International Journal of Applied Earth Observation and Geoinformation 124, 103520.