Check for updates

AGU ADVANCING EARTH AND SPACE SCIENCES

Earth and Space Science

RESEARCH ARTICLE
10.1029/2026EA005015

CaMa-PyTorch (v1.0): A New Catchment-Based Macro-Scale Floodplain Model Written in PyTorch

Zenghui Liu[1], Qiyu Li[1], Qingliang Li[1], Zhongwang Wei[2], Cheng Zhang[3], Hua Yuan[2], Yunfeng Lv[4], Jingquan Ren[5], Xiaoming Shi[6], and Yongjiu Dai[2]

[1]College of Artificial Intelligence, Changchun Normal University, Changchun, China, [2]School of Atmospheric Sciences, Sun Yat-Sen University, Zhuhai, China, [3]College of Computer Science and Technology, Jilin University, Changchun, China, [4]College of Geographical Sciences, Changchun Normal University, Changchun, China, [5]Jilin Meteorological Administration, Changchun, China, [6]Division of Environment and Sustainability, The Hong Kong University of Science and Technology, Hong Kong, China

Key Points:

- CaMa-Flood v4.23 was reimplemented in PyTorch for deep learning integration
- A custom indexing scheme preserves river-network logic and numerical fidelity from the Fortran version
- A diagnostic tool validated GRDC-based 0.1° CONUS simulations, ensuring high fidelity and efficiency

Supporting Information:

Supporting Information may be found in the online version of this article.

Correspondence to:

Q. Li,
liqingliang@ccsfu.edu.cn

Citation:

Liu, Z., Li, Q., Li, Q., Wei, Z., Zhang, C., Yuan, H., et al. (2026). CaMa-Pytorch (v1.0): A new catchment-based macro-scale floodplain model written in PyTorch. *Earth and Space Science*, *13*, e2026EA005015. https://doi.org/10.1029/2026EA005015

Received 11 JAN 2026
Accepted 8 OCT 2026

Author Contributions:

Conceptualization: Qingliang Li
Formal analysis: Zenghui Liu, Qiyu Li, Qingliang Li
Funding acquisition: Qingliang Li
Investigation: Qingliang Li
Methodology: Zenghui Liu, Qingliang Li, Zhongwang Wei, Hua Yuan, Yongjiu Dai
Resources: Qingliang Li
Software: Zenghui Liu, Qiyu Li, Qingliang Li
Supervision: Qingliang Li
Validation: Zenghui Liu, Qiyu Li, Cheng Zhang, Yunfeng Lv, Jingquan Ren, Xiaoming Shi
Visualization: Zenghui Liu, Qiyu Li

© 2026 The Author(s). Earth and Space Science published by Wiley Periodicals LLC on behalf of American Geophysical Union.
This is an open access article under the terms of the Creative Commons Attribution-NonCommercial License, which permits use, distribution and reproduction in any medium, provided the original work is properly cited and is not used for commercial purposes.

Abstract Accurate simulation of river discharge underpins hydrological research, climate-impact assessment, and flood forecasting. CaMa-Flood, a widely used global river-routing model, applies physically based principles to represent water transport through river networks. However, its legacy Fortran implementation restricts compatibility with modern AI frameworks and prevents efficient GPU utilization. Here, we present CaMa-PyTorch (v1.0), a Python reimplementation of CaMa-Flood v4.23 within PyTorch. A custom indexing scheme reproduces the flexible array operations of the Fortran model while preserving its river network logic and core algorithms. We evaluated numerical fidelity through modular tests and a continuous simulation over the continental United States at 0.0833° resolution from 2005 through 2010. Daily streamflow observations from the Global Runoff Data Centre (GRDC) provided an external benchmark, and model behavior was assessed using hydrograph comparisons, residual analyses, spatial diagnostics, and standard performance metrics. Across these evaluations, CaMa-PyTorch closely reproduced the temporal and spatial discharge patterns generated by the Fortran implementation. Its tensor architecture also supports parallel computation and GPU execution. As a proof-of-concept for machine learning interoperability, a long short-term memory (LSTM) network was used to postprocess simulated discharge and improved predictive performance. This experiment demonstrates output level coupling with a PyTorch data-driven module but does not constitute a fully differentiable modeling system. CaMa-PyTorch therefore provides a numerically faithful and computationally flexible basis for large scale river modeling and future research on more closely integrated physical and data-driven approaches.

Plain Language Summary Rivers play a critical role in water resources, climate studies, and flood forecasting. To simulate how water moves through rivers, scientists often use a model called CaMa-Flood. The original version of this model was written in Fortran, which makes it hard to connect with modern artificial intelligence tools. In this study, we rewrote CaMa-Flood in Python using the PyTorch framework. This new version can work smoothly with AI methods, but still keeps the same scientific accuracy as the original. We also built a user-friendly evaluation tool that automatically prepares data, compares model results with river observations, and produces clear graphs and statistics. Using river flow data from the Global Runoff Data Centre, we showed that the new PyTorch version produces nearly identical results to the original Fortran version. This upgrade provides a modern, AI-ready platform that can support more accurate flood forecasts, faster model testing, and new ways of combining physics and machine learning in hydrology.

1. Introduction

Global warming and intensified human activities have significantly disrupted the hydrological cycle, leading to growing water resource challenges and heightened flood risks (Wang et al., 2019). Reliable river discharge estimates are critical for tracking changes in the global water cycle, supporting flood management, and informing a wide range of hydrological and environmental applications (Andreadis Coss et al., 2025; Yamazaki et al., 2011). It plays a key role in flood forecasting by enabling timely warnings and effective emergency responses. In water resource management, discharge estimates help allocate water across domestic, agricultural, and industrial uses in a balanced and sustainable way (Coxon et al., 2015; Gleason & Smith, 2014). Beyond these operational applications, reliable discharge data are essential for tracking long-term hydrological trends and assessing climate

Writing – original draft: Zenghui Liu, Qiyu Li
Writing – review & editing: Qingliang Li

change impacts. They support historical analyses, scenario development, and the design of adaptive management strategies (Cong et al., 2009; Nearing et al., 2023; Quilbé et al., 2008; Ruhi et al., 2018).

Runoff typically begins at the land surface or in shallow soils after rainfall (Bronstert & Plate, 1997; Pearce et al., 1986), and this process is well captured by land surface and rainfall–runoff models (Campling et al., 2002; Dai et al., 2003). Once runoff enters river networks, however, its downstream flow and interaction with floodplains shape key hydrograph features such as peak reduction, delayed recession, and temporary water storage (Dey et al., 2022; Tull et al., 2022). To represent these dynamics, hydrodynamic models have been developed to couple river channels with adjacent floodplains, often based on simplified forms of the shallow water equations (Guo et al., 2021; Sampson et al., 2012; Yamazaki et al., 2013). These integrated approaches support more accurate flood forecasting and inform basin-scale water management. To enable large-scale simulations of river–floodplain dynamics, Yamazaki et al. (2011) developed the CaMa-Flood (Catchment-based Macro-scale Floodplain) model, which divides basins into linked catchments to simulate downstream routing and lateral floodplain exchange. By combining simplified hydrodynamics with detailed topography, the CaMa-Flood model captures processes such as flood wave propagation and overbank inundation (Yamazaki et al., 2011, 2014, 2017). To meet increasing demands for accurate flood modeling and large-scale risk assessment, the CaMa-Flood model has undergone ongoing refinements in both structural design and hydrodynamic representation (Zhou & Liu, 2024). Despite these advances, key challenges remain, including parameter uncertainty and incomplete understanding of physical processes (Shen et al., 2023). These issues suggest opportunities for deep learning to enhance model structure and quantify uncertainties more effectively.

Recent years have seen growing interest in applying deep learning models to streamflow simulation. Unlike traditional models based on physical equations, deep learning models learn complex, nonlinear relationships directly from data by optimizing network parameters (Hornik, 1991; Hornik et al., 1989). This allows for improved simulation of rainfall–runoff dynamics, especially in regions with limited process understanding or sparse observations (Adera et al., 2024; Cai et al., 2024; Feng et al., 2020, 2022). However, deep learning models often operate as black boxes, relying only on input–output data mappings and disregarding physical laws (Li et al., 2024). This limits their ability to capture internal processes or ensure consistency with known hydrologic behavior. Current hybrid approaches typically involve simple coupling strategies—such as using AI to correct physical model outputs or residuals (Ng et al., 2023)—which improve accuracy but offer limited insight into system dynamics or structural uncertainty. To address these issues, recent developments have focused on differentiable modeling, which embeds physical equations directly into neural networks (Gelbrecht et al., 2023). Instead of merely constraining outputs, these models incorporate governing equations as part of the learning process, allowing for the simultaneous estimation of parameters, learning of system dynamics, and quantification of uncertainty (Shen et al., 2023). For example, Feng et al. (2022) applied a differentiable LSTM–HBV model that outperformed the traditional HBV and matched the performance of state-of-the-art deep learning methods. Similar strategies have been applied to stream temperature modeling and atmospheric dynamics with promising results (Kochkov et al., 2024; Rahmani et al., 2023). Building on this direction, our previous work (Li et al., 2024) embedded symbolic regression into a differentiable Penman–Monteith framework, enabling the discovery of explicit physical relationships while improving model robustness. These advances suggest a promising path forward—one where AI models do not merely mimic observed behavior but learn to represent hydrologic processes through the integration of physical laws within their core structure.

However, more complex systems—such as the CaMa-Flood model (latest version v4.23, https://github.com/global-hydrodynamics/CaMa-Flood_v4, last access: 2024-11-25)—remain largely unexplored in this context. The CaMa-Flood model represents a physically rigorous hydrodynamic framework for simulating river–floodplain interactions at large scales. Its high computational cost and intricate process formulations pose significant challenges for integration with differentiable learning frameworks. Transforming such models into differentiable structures represents a critical and ambitious step.

To address these technical challenges, we developed CaMa-PyTorch (v1.0), a Python reimplementation of the original Fortran CaMa-Flood model (v4.23) within the PyTorch framework. Its tensor architecture supports parallel execution and GPU acceleration while allowing hydrodynamic calculations and machine learning components to operate within a common computational environment. CaMa-PyTorch therefore provides both an efficient flood modeling system and a flexible foundation for future hybrid modeling. To demonstrate this interoperability, we conducted a proof-of-concept experiment in which a long short term memory network

23335084, 2026, 11, Downloaded from https://agupubs.onlinelibrary.wiley.com/doi/10.1029/2026EA005015 by Hong Kong University Of, Wiley Online Library on [01/11/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

23335084, 2026, 11, Downloaded from https://agupubs.onlinelibrary.wiley.com/doi/10.1029/2026EA005015 by Hong Kong University Of, Wiley Online Library on [01/11/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

postprocessed discharge simulated by CaMa-PyTorch. The correction improved discharge simulation performance and showed that model outputs could be effectively integrated with data-driven modules in PyTorch. This experiment demonstrates the feasibility of coupling CaMa-PyTorch outputs with a data driven correction model. Future work may extend this output level application toward closer integration of hydrodynamic modeling and artificial intelligence, including fully differentiable formulations.

2. CaMa-PyTorch (v1.0) Model

To enhance scalability and maintainability in hydrodynamic simulations, we reimplemented the CaMa-Flood model (version 4.23), originally developed in Fortran, into a modular framework built with PyTorch, named CaMa-PyTorch (v1.0). While preserving the core logic of the original model, our restructured version replaces procedural Fortran routines with efficient, vectorized tensor operations and a custom indexing system designed to represent complex river–floodplain networks. Tensorization replaces the spatial loops over river network cells in the original Fortran implementation with tensor indexing, elementwise operations, and reductions. However, each cell's outflow is still calculated from the water level, storage, and discharge values available at the start of the corresponding sub-step. As a result, tensorization preserves the original numerical update order and the physical dependencies among state variables. This reimplementation improves computational efficiency through GPU acceleration, simplifies the codebase for better readability, and facilitates integration with hydrologic and machine learning tools. The rest of this section is organized as follows: Section 2.1 introduces the structure and routing mechanisms of CaMa-Flood (v4.23), while Section 2.2 presents the design and workflow of the CaMa-PyTorch (v1.0) framework. Section 2.3 describes the evaluation software developed for assessing CaMa-PyTorch outputs.

2.1. CaMa-Flood (v4.23) Model

2.1.1. Model Structure

This section outlines the core components and simulation workflow of CaMa-Flood, from map preparation to hydrodynamic routing.

2.1.1.1. Hydrological Map Preparation

The model relies on a pre-processed hydrological map that specifies river networks, flow direction, upstream area, river width, slope, and related attributes for each grid cell. These maps are derived from digital elevation models and processed using flow accumulation algorithms to define drainage structure and connectivity across complex river systems (Yamazaki et al., 2017).

2.1.1.2. Grid Structure and River Network

Based on the hydrological map, the model runs on a regular latitude–longitude grid, where each cell represents a river segment. The main river network is represented as a directed graph in which each nonterminal river cell is linked to one primary downstream cell through the one-dimensional integer index tensor I1NEXT. This tensor is derived from the downstream coordinates in the original two-dimensional river network and ordered from headwaters to outlets during model initialization. All subsequent routing calculations are based on the river-network topology defined by I1NEXT. When a river cell has multiple downstream branches, CaMa-PyTorch preserves the separate bifurcation channel representation used in the Fortran model. These bifurcation links must be specified in the input river-network data. This structure enables efficient flow routing while preserving the hierarchical organization of main channels and tributaries.

2.1.1.3. Hydrologic Inputs and Boundary Conditions

With the spatial structure defined, CaMa-Flood dynamically calculates the minimum stable time step based on hydraulic properties at each grid cell—specifically, local flow velocity, channel length, and slope. This ensures compliance with the Courant–Friedrichs–Lewy (CFL) condition for numerical stability (Bates et al., 2010; Yamazaki et al., 2013). The calculation time step is then adjusted accordingly to maintain stable and accurate flow routing. Under the default configuration, the PyTorch implementation uses the same CFL candidate time step calculation, valid cell filtering, river mouth treatment, minimum water depth threshold, global minimum

23335084, 2026, 11, Downloaded from https://agupubs.onlinelibrary.wiley.com/doi/10.1029/2026EA005015 by Hong Kong University Of, Wiley Online Library on [01/11/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

selection, and internal sub-step count formula as the Fortran version. Consequently, both implementations enforce the CFL constraint in a mathematically equivalent manner. Tensorization changes only the computational procedure by replacing explicit Fortran loops and the OpenMP minimum reduction with tensor indexing and torch.min, while preserving the original CFL formulation.

2.1.1.4. Hydrodynamic Flow Simulation

The model uses a local inertia formulation of the Saint-Venant equations to simulate water level, discharge, and flow velocity at each time step, ensuring mass conservation and representing hydraulic gradients (Wu et al., 2014). This approach enables the capture of both in-channel flow and floodplain inundation across large basins.

2.1.1.5. Floodplain Interaction

When river water levels exceed the bankfull threshold, excess flow is routed laterally into surrounding floodplains. These areas serve as temporary storage zones, gradually releasing water back into the main channel based on local topography and hydraulic gradients. This subgrid-scale representation enhances the model's ability to capture flood retention and delayed drainage in low-lying regions.

2.1.1.6. Model Outputs and Applications

At each time step, CaMa-Flood outputs gridded estimates of water level, discharge, inundation depth, and flooded area. These outputs can be validated against observations and serve as inputs for downstream applications such as flood risk assessment or water resource management.

2.1.2. Key Contributions and Challenges in the PyTorch Reimplementation

Building on the structure introduced in Section 2.1.1, we highlight two major contributions in the development of CaMa-PyTorch. These advances stem from a critical evaluation of the original Fortran-based CaMa-Flood model and its underlying computational design. In particular, we focus on the shift from a procedural, loop-driven approach to a modular, tensor-based framework optimized for GPU parallelization.

In CaMa-Flood, hydrodynamic processes are implemented using explicit loops over grid cells and river segments, with state variables stored in arrays accessed through manual indexing. While efficient on CPUs, this method poses challenges for scalability and for integration with modern machine learning workflows. Moreover, the model's tightly coupled subroutines and fixed configurations reduce its flexibility and adaptability.

In contrast, CaMa-PyTorch adopts a modular structure designed for parallel execution on GPUs, as described in Section 2.2. This approach improves scalability and facilitates integration with data-driven methods, while preserving the core hydrodynamic principles of the original model.

However, the transition from Fortran to PyTorch is not without challenges. Certain low-level optimizations in Fortran, such as direct memory access and loop unrolling, are difficult to replicate in high-level tensor frameworks. Additionally, fully vectorizing operations over irregular river networks can reduce code flexibility and make debugging more complex.

2.2. CaMa-PyTorch (v1.0) Model Architecture and Workflow

Figure 1 illustrates the structural diagram of the CaMa-PyTorch framework. The framework adopts a modular package structure that separates key components such as hydrodynamic logic, routing algorithms, data input/output, and configuration management. These modules interact through clearly defined interfaces, and detailed class attributes and method signatures are provided in Text S1 of the Supporting Information S1.

At the core of this architecture are two components: a custom indexing system and a matrix-based computation engine. The indexing system maps hydrological states across river segments and neighboring grid cells, while the computation engine leverages PyTorch tensor operations to update water storage and fluxes efficiently. Together, these components enable scalable simulation of river–floodplain dynamics on GPUs. The simulation workflow proceeds through four main stages: initializing the network topology, loading boundary conditions, performing iterative routing updates, and generating outputs.

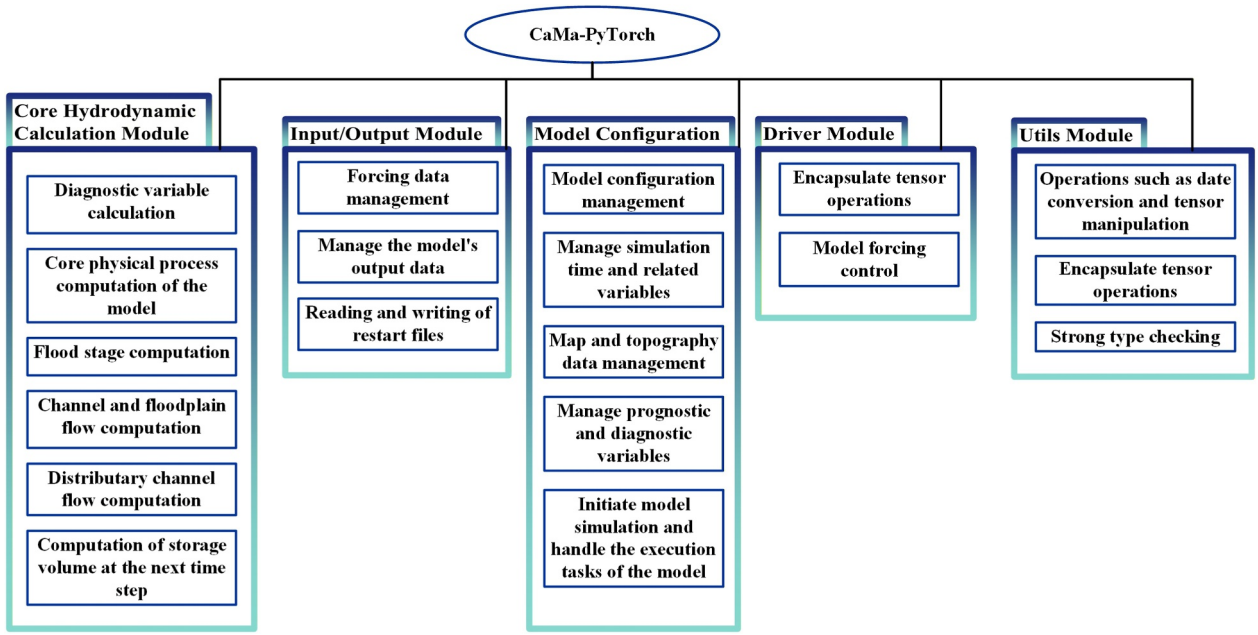


Figure 1. The main modules of CaMa-PyTorch.

2.2.1. Custom Indexing System

To preserve the original Fortran-style array behavior—where indexing starts from 1 and arrays are accessed using integers, slices, or boolean masks—we introduce three wrapper classes: **Ftensor_1D, Ftensor_2D and Ftensor_3D**. These wrappers adapt PyTorch tensors to support 1-based indexing, enabling direct translation of Fortran logic into Python without requiring constant manual index adjustment.

Each wrapper tracks the starting index for every dimension (defaulting to 1) and automatically shifts user-provided indices to align with Python's zero-based indexing. This shift is applied consistently across all common indexing types, including integers, slices, and boolean conditions.

The **Ftensor_1D** class wraps a one-dimensional tensor and adjusts all index operations accordingly. It also provides a **where(condition)** method that returns the positions where a given condition holds, using 1-based indexing.

The **Ftensor_2D** and **Ftensor_3D** classes generalize the same principle to two and three dimensions, respectively. Each dimension has its own starting index, and all indexing behavior mirrors that of **Ftensor_1D**.

This approach allows us to express code such as **D2RIVOUT(ISEQ,1)** from Fortran directly as **D2RIVOUT [ISEQ, 1]** in Python, with the index shift handled transparently. As a result, the original algorithmic structure is preserved, and the code remains readable, accurate, and compatible with GPU acceleration via PyTorch. The model's core state variables are organized and accessed through these wrapper classes to maintain consistency with the one-based indexing scheme of the original Fortran implementation. In the formal CaMa-PyTorch implementation, both the CPU and GPU backends use tensor execution pathways optimized for their respective computing devices. To reduce the overhead associated with index conversion and repeated river-network topology queries, some performance-critical calculations operate directly on zero-based raw tensor views and preconstructed river-network topology indices. Although the CPU and GPU backends employ different computational optimization strategies, they preserve the same physical equations, source-to-target relationships within the river network, and numerical update sequence.

2.2.2. Matrix-Based Computation Approach

The CaMa-PyTorch (v1.0) framework simulates large-scale hydrological processes by organizing computations as tensor operations, using broadcasting and masking to enable efficient parallel processing across all grid cells.

23335084, 2026, 11, Downloaded from https://agupubs.onlinelibrary.wiley.com/doi/10.1029/2026EA005015 by Hong Kong University Of, Wiley Online Library on [01/11/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

23335084, 2026, 11, Downloaded from https://agupubs.onlinelibrary.wiley.com/doi/10.1029/2026EA005015 by Hong Kong University Of, Wiley Online Library on [01/11/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

This design supports accelerated execution on both CPUs and GPUs while maintaining compatibility with PyTorch's dynamic computation graph.

To highlight the structural differences between the Fortran and PyTorch implementations, we present the pseudocode of the P2STOOUT calculation for normal cells and river mouth grids in the CMF_CALC_INFLOW function as a representative example in Algorithm 1 (Fortran) and Algorithm 2 (PyTorch). Although only one function is shown, many components in the model follow a similar transition from nested-loop logic to tensor-based operations. In the Fortran version, flow is computed using explicit loops, with a limiter applied to prevent total outflow from exceeding the grid cell's available water. This limiter modifies the discharge, originally calculated by the local inertial equation, to maintain physical consistency. By contrast, the PyTorch implementation uses batched tensor operations to aggregate outflow demands across spatial cells, thereby avoiding explicit iteration over individual river cells. This structure improves computational efficiency and supports integration with machine learning workflows.

Tensorization changes only the execution of spatial calculations within each time step. It does not modify the temporal integration sequence or the data dependencies among physical modules. At each external time step, the model first derives river channel and floodplain water levels from the current water storage and then determines the adaptive number of internal substeps. During each substep, it computes river and floodplain outflows, evaluates optional bifurcation flows, aggregates upstream inflows, stores the previous state, updates water storage, and recalculates water levels.

Nevertheless, some processes in hydrological modeling remain inherently sequential and cannot be fully vectorized. A representative example is the **CMF_CALC_FLDSTG_DEF** subroutine, which calculates water storage and depth in river channels and floodplains. It first checks whether the combined storage exceeds the river channel's capacity. If so, the model updates floodplain layers one at a time, recalculating water depth and hydraulic gradients using physically based parameters such as river width, length, and slope. The number of layers varies with local water volume, introducing conditional logic that requires iterative computation.

Cells within the same floodplain layer can be processed in parallel, but the floodplain layers of each cell must be evaluated sequentially. This is necessary because the water depth and effective floodplain width of a given layer depend on cumulative values from the preceding layers. Retaining this sequence preserves the layered relationship between water level and storage in CaMa-Flood. This subroutine highlights a key exception to the otherwise parallel structure of CaMa-PyTorch. When multiple upstream river reaches discharge into the same target cell, CaMa-PyTorch uses index_add_ to accumulate river inflow, floodplain inflow, and bifurcation flow. Each upstream flux is added to its corresponding downstream cell, and water storage is then updated from the balance between total inflow and outflow. This tensor-based accumulation preserves the original correspondence between fluxes in the discrete governing equations. By combining explicit sequential logic where needed with predominantly tensor-based operations, the framework achieves both computational efficiency and hydrodynamic fidelity. This structure also lays the groundwork for future integration of physical process models with AI-based learning components.

Algorithm 1 CMF_CALC_INFLOW--- calculate **P2STOOUT** part (Fortran version).

```
Input: 1D downstream I1NEXT, end of river-link sequence NSEQRIV, end of river-mouth sequence NSEQALL, River outflow D2RIVOUT, floodplain outflow D2FLDOUT, time step length DT
Output: total outflow from a grid P2STOOUT = (P2STOOUT_1, … , P2STOOUT_NSEQALL)
# initialize P2STOOUT for all cells
1: for (ISEQ = 1 to NSEQALL) do
2:     P2STOOUT(ISEQ, 1) = 0
3: end for
# calculate P2STOOUT for normal cells
4: for (ISEQ = 1 to NSEQRIV) do
5:     JSEQ              = I1NEXT(ISEQ)              //next cell's pixel
6:     OUT_R1            = max(  D2RIVOUT(ISEQ, 1), 0)
7:     OUT_R2            = max(–D2RIVOUT(ISEQ, 1), 0)
```

23335084, 2026, 11, Downloaded from https://agupubs.onlinelibrary.wiley.com/doi/10.1029/2026EA005015 by Hong Kong University Of, Wiley Online Library on [01/11/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

```
8:    OUT_F1                = max(  D2FLDOUT(ISEQ, 1), 0)
9:    OUT_F2                = max(−D2FLDOUT(ISEQ, 1), 0)
10:   DIUP                  = (OUT_R1 + OUT_F1) × DT
11:   DIDW                  = (OUT_R2 + OUT_F2) × DT
12:   P2STOOUT(ISEQ, 1)     = P2STOOUT(ISEQ, 1) + DIUP
13:   P2STOOUT(JSEQ, 1)     = P2STOOUT(JSEQ, 1) + DIDW
14: end for
# calculate P2STOOUT for river mouth grids
15: for (ISEQ = NSEQRIV + 1 to NSEQALL) do
16:   OUT_R1                = max(  D2RIVOUT(ISEQ, 1), 0)
17:   OUT_F1                = max(  D2FLDOUT(ISEQ, 1), 0)
18:   P2STOOUT(ISEQ, 1)     = P2STOOUT(ISEQ, 1) + OUT_R1 * DT + OUT_F1 * DT
19: end for
20: return P2STOOUT
```

Algorithm 2 CMF_CALC_INFLOW--- calculate **P2STOOUT** part (PyTorch version).

```
Input: 1D downstream I1NEXT, end of river-link sequence NSEQRIV, end of river-mouth sequence NSEQALL, River outflow D2RIVOUT, floodplain outflow D2FLDOUT, device cuda, 8 byte double-precision float JPRD, time step length DT
Output: total outflow from a grid P2STOOUT = (P2STOOUT_1, … , P2STOOUT_NSEQALL)
# initialize P2STOOUT for all cells
1:   P2STOOUT = torch.zeros((NSEQALL, 1), dtype = torch.float64, device = device)
2: ZeroT      = torch.tensor(0.0, dtype = torch.float64, device = device)
// Ftensor_2D is introduced in Section 2.2.1 Custom indexing system
3: P2STOOU T  = Ftensor_2D(P2STOOUT, start_row = 1, start_col = 1)
# calculate P2STOOUT for normal cells
4: RC_Index     = torch.arange(1, NSEQRIV + 1, device = device)
5: JSEQ         = I1NEXT[RC_Index]    //next cell's pixel
6: OUT_R1       = torch.maximum(  D2RIVOUT[RC_Index, 1], ZeroT)
7: OUT_R2       = torch.maximum(−D2RIVOUT[RC_Index, 1], ZeroT)
8: OUT_F1       = torch.maximum(  D2FLDOUT [RC_Index, 1], ZeroT)
9: OUT_F2       = torch.maximum(−D2FLDOUT [RC_Index, 1], ZeroT)
10: DIUP        = (OUT_R1 + OUT_F1) × DT
11: DIDW        = (OUT_R2 + OUT_F2) × DT
12: P2STOOUT.raw().index_add_(0, RC_Index − 1, DIUP.unsqueeze(1))
13: P2STOOUT.raw().index_add_(0, JSEQ − 1, DIDW.unsqueeze(1))
# calculate P2STOOUT for river mouth grids
14: RMF_Index   = torch.arange(NSEQRIV + 1, NSEQALL + 1, device = device)
15: OUT_R1      = torch.maximum(  D2RIVOUT[RMF_Index, 1], ZeroT)
16: OUT_F1      = torch.maximum(  D2FLDOUT[RMF_Index, 1], ZeroT)
17: P2STOOUT[RMF_Index, 1] = P2STOOUT[RMF_Index, 1] + OUT_R1 * DT + OUT_F1 * DT
18: return P2STOOUT
```

2.3. Evaluation Software

We further developed an evaluation tool for diagnostic assessment and visualization of runoff and inundation outputs generated by CaMa-PyTorch. The tool is implemented in Python 3.10 with an object-oriented architecture and is organized into four principal modules: (a) data input and preprocessing, (b) computation of evaluation metrics, (c) visualization, and (d) export and report generation. Key dependencies include PyQt5 (graphical interface), NumPy, pandas, xarray, netCDF4, and Matplotlib, and the tool natively supports standard raster/grid and time-series formats (e.g., NetCDF, CSV, and GeoTIFF). The design emphasizes cross-platform portability,

23335084, 2026, 11, Downloaded from https://agupubs.onlinelibrary.wiley.com/doi/10.1029/2026EA005015 by Hong Kong University Of, Wiley Online Library on [01/11/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

maintainability, and extensibility: clearly defined module interfaces facilitate the addition or replacement of evaluation metrics and visualization components, and export functionality (PNG, CSV, GeoTIFF) enables seamless integration with downstream analyses and figure production (Detailed operating instructions are provided in Text S2 of the Supporting Information S1). Collectively, these features provide a reproducible, extensible, and user-friendly workflow for quantitative validation and graphical diagnostics of hydrodynamic model outputs.

3. Application Examples and Diagnostic Evaluation

We developed a diagnostic tool to enable systematic visualization and analysis of CaMa-PyTorch simulation outputs. Implemented in Python 3.10 with a PyQt5 interface, the tool supports interactive selection of evaluation variables (e.g., discharge, maximum flood depth, inundated area), automated generation of time-series plots and spatial maps, residual and difference analyses, and focused evaluation at user-selected stations. Standard workflows are executed through the interface, and outputs can be exported in widely used formats to ensure reproducibility and integration with downstream analyses.

3.1. Data Sources

In this study, the input data for CaMa-Flood consist of static river network data and dynamic land surface forcing data. The static river network data are derived from the MERIT Hydro data set (Yamazaki et al., 2019), which provides high-resolution global information on river networks, watershed boundaries, channel slopes, and widths, and corrects for river deviations caused by artificial barriers or topographic errors in conventional DEMs, thereby ensuring the geometric accuracy of the river network and the spatial consistency of simulation results. In this study, land-surface runoff from the ERA5-Land reanalysis data set was used as the hydrological forcing for CaMa-PyTorch. ERA5-Land was developed by the European Centre for Medium-Range Weather Forecasts (ECMWF) within the framework of the Copernicus Climate Change Service and is a high-resolution offline replay of the land component of the ERA5 reanalysis. The combination of MERIT Hydro and ERA5-Land not only ensures spatial accuracy in flood propagation simulations but also maintains temporal continuity and reliability, thereby enhancing the overall performance and credibility of the model results.

Daily streamflow observations from the Global Runoff Data Centre (GRDC) served as the benchmark for validation. Model outputs were obtained from both the original Fortran implementation of CaMa-Flood and the CaMa-PyTorch reimplementation (v1.0). Station metadata (coordinates and identifiers) were taken from GRDC records, while model static data sets (e.g., grid mask, elevation fields) were used for station–grid mapping and spatial diagnostics. Details of the evaluation period, station selection, and detailed pseudocode for these procedures are provided in Figure S3 and Tables S2–S4 of the Supporting Information S1.

3.2. Station-Based Validation

Time series validation was performed at selected GRDC stations across the continental United States. These sites included stations in the 10 largest river basins, representative locations along the five largest rivers, four outlet stations shown in Figure 2, and 10 additional stations from other basins. In the spatial maps, river reaches were classified according to long term mean discharge following (Lehner & Grill, 2013). Reaches with a mean discharge of at least 1,000 m^3/s were classified as main rivers, whereas those with values from 100 to less than 1,000 m^3/s were classified as secondary rivers. Daily discharge simulated by the Fortran and PyTorch implementations was compared at each station (Figure 2). CaMa-PyTorch closely reproduced the Fortran results in both discharge magnitude and temporal variation, demonstrating strong consistency between the two implementations.

3.3. Spatial-Scale Validation

To evaluate migration fidelity at the station scale, we compared the relative differences in RMSE, KGESS, BIAS, and Pearson's correlation coefficient between CaMa-PyTorch and the original Fortran version of CaMa-Flood (Figure 3). Relative differences were calculated using the Fortran results as the reference. Positive values indicate higher metric values in CaMa-PyTorch, whereas negative values indicate lower values.

All four metrics showed strong spatial agreement between the two implementations, with relative differences close to zero at most stations. RMSE differences were generally within ±0.0004%, indicating that the overall error magnitude remained essentially unchanged. KGESS differences were within ±0.0009%, confirming that the

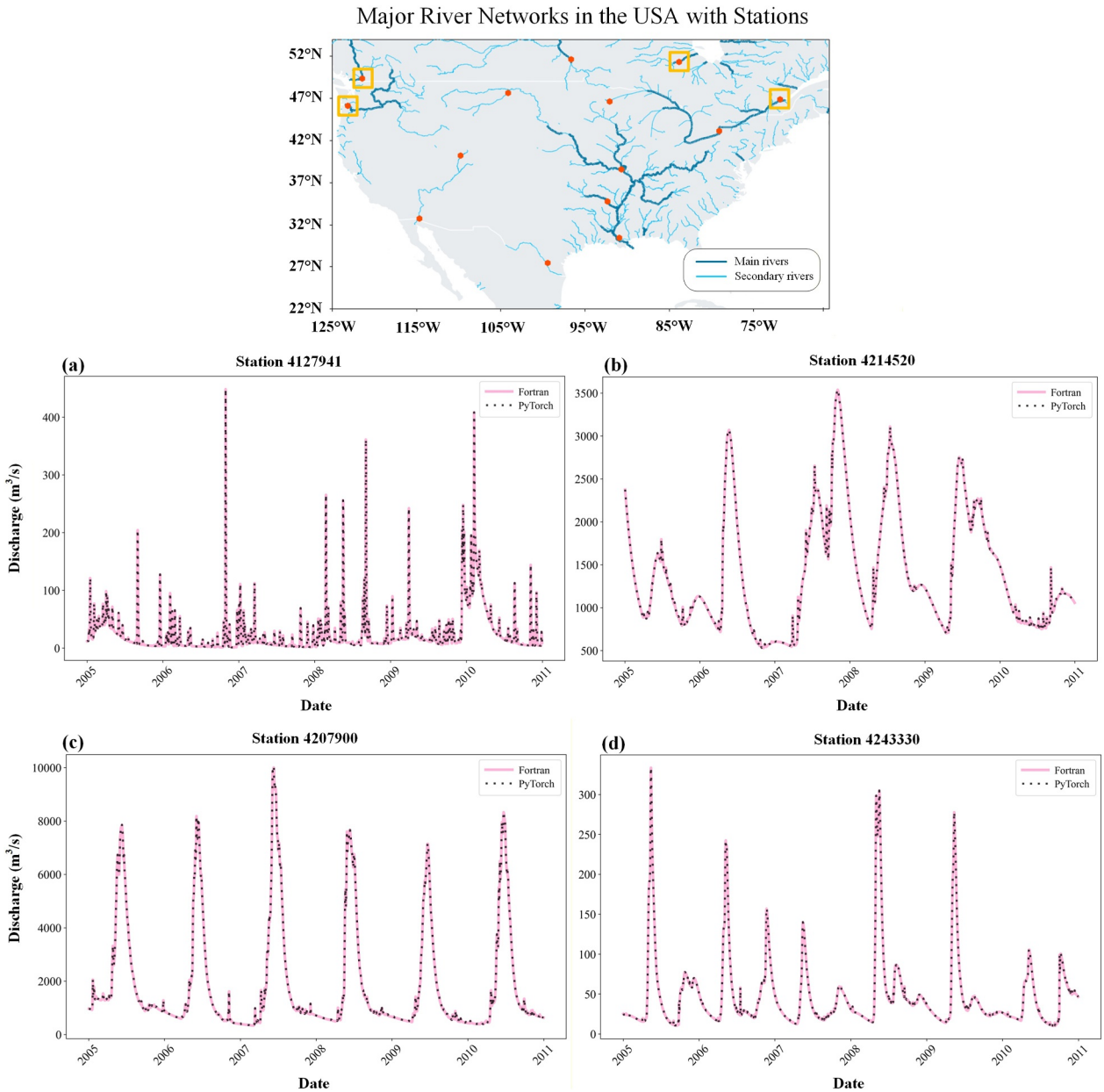


Figure 2. Comparison of simulated discharge from 2005 to 2010 between the Fortran and PyTorch implementations at multiple gauging stations: (a) Station 4127941; (b) Station 4214520; (c) Station 4207900; and (d) Station 4243330.

overall hydrological performance was preserved. BIAS differences were generally within ±0.0004% and showed no coherent regional pattern, suggesting that the migration introduced no systematic spatial shift in discharge overestimation or underestimation. Differences in Pearson's correlation coefficient were within ±0.0006% at nearly all stations, indicating that the temporal variability and agreement of simulated discharge were also preserved. The few more noticeable differences were limited to isolated stations, primarily in the central and western United States. Overall, the differences were extremely small, spatially scattered, and lacked any consistent regional pattern. These results demonstrate that CaMa-PyTorch closely reproduces the station-scale error characteristics, hydrological performance, and discharge dynamics of the original Fortran implementation.

We also evaluated the spatial consistency of floodplain simulations by comparing grid-level differences in average flooded area and average maximum flood depth over 2005 to 2010 (Figure S4 in Supporting Information S1). Differences in both variables were close to zero across most grid cells, with larger discrepancies

23335084, 2026, 11, Downloaded from https://agupubs.onlinelibrary.wiley.com/doi/10.1029/2026EA005015 by Hong Kong University Of, Wiley Online Library on [01/11/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

10.1029/2026EA005015

23335084, 2026, 11, Downloaded from https://agupubs.onlinelibrary.wiley.com/doi/10.1029/2026EA005015 by Hong Kong University Of, Wiley Online Library on [01/11/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

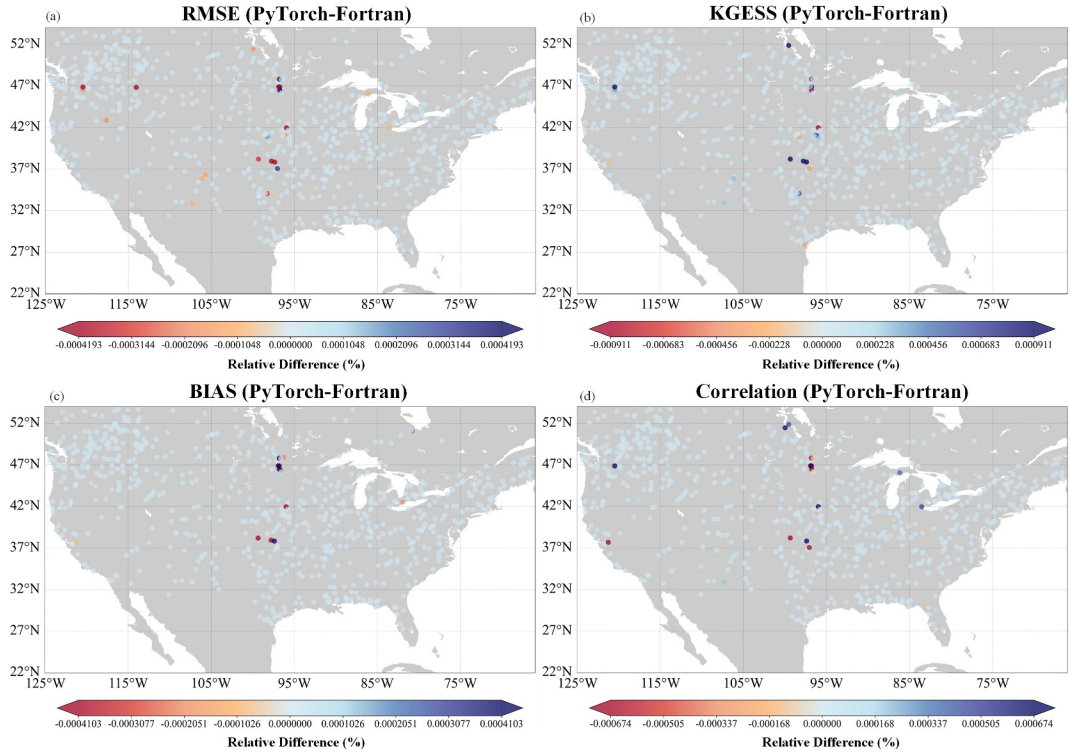


Figure 3. Spatial distributions of station-level relative differences in (a) RMSE, (b) KGESS, (c) BIAS, and (d) Pearson's correlation coefficient between CaMa-PyTorch and the Fortran implementation of CaMa-Flood. Relative differences are expressed as percentages using the Fortran results as the reference.

restricted to a few isolated locations. No widespread or consistently signed spatial bias was observed. CaMa-PyTorch therefore preserves the inundation extent, flood depth, and spatial heterogeneity simulated by the original Fortran model, further confirming the strong numerical agreement between the two implementations.

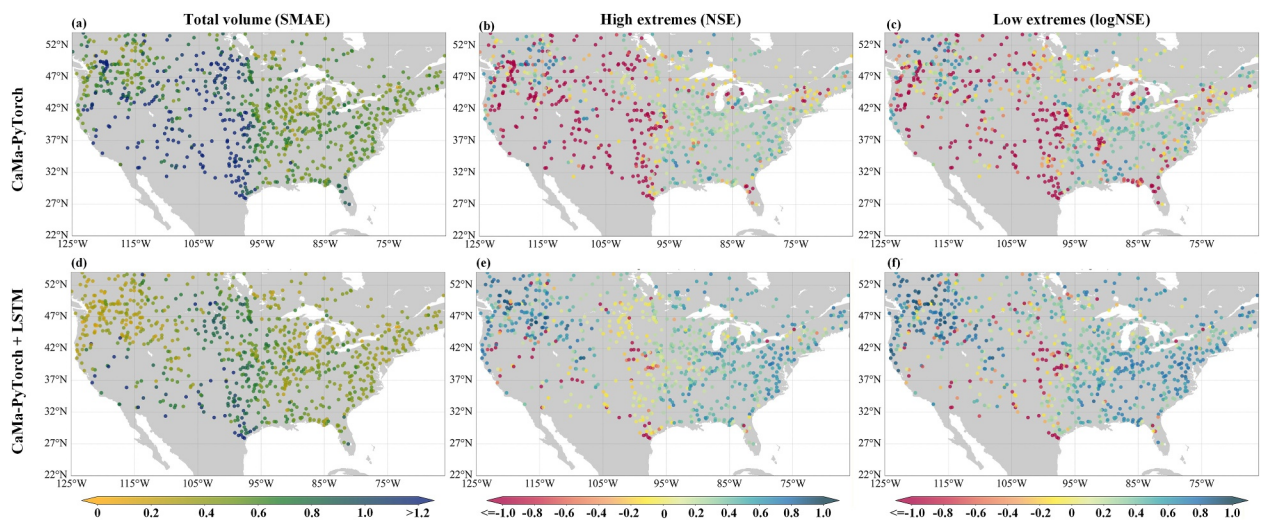


Figure 4. Station-scale spatial performance of CaMa-PyTorch and the LSTM-coupled framework. Panels (a–c) present results from the original CaMa-PyTorch simulations, whereas panels (d–f) present results after LSTM-based postprocessing.

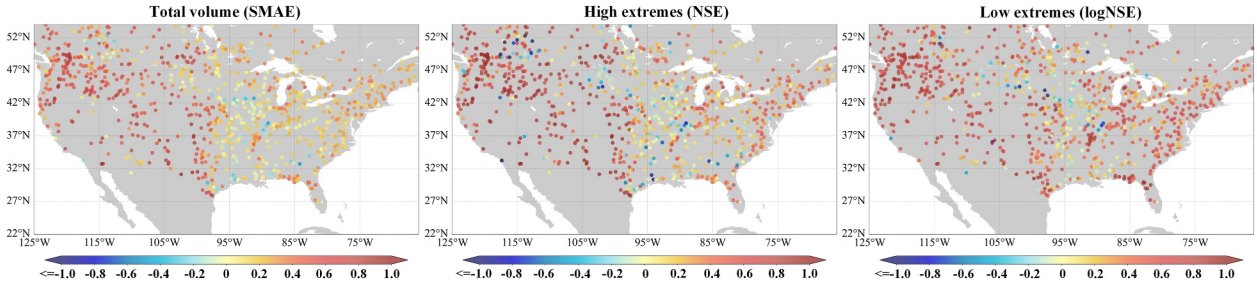


Figure 5. Spatial distributions of station-level skill scores for LSTM postprocessing relative to the original CaMa-PyTorch simulations. Positive values indicate improved performance after LSTM postprocessing, whereas negative values indicate reduced performance.

3.4. Proof-of-Concept AI-Coupling Experiment

To evaluate the compatibility of CaMa-PyTorch with PyTorch-based machine-learning modules, we conducted a proof-of-concept coupling experiment. Figures 4 and 5 present the performance metrics for this experiment. CaMa-PyTorch first generated baseline discharge simulations using the prescribed river network, hydrodynamic equations, and land-surface runoff forcing. A long short-term memory network (LSTM) was then trained to estimate the discrepancies between the simulated discharge and observations from the Global Runoff Data Centre (GRDC). The trained network was subsequently applied to correct the baseline discharge estimates.

The analysis used daily GRDC observations that passed station-to-grid matching and quality-control procedures. Performance was evaluated for three aspects of streamflow simulation. Scaled mean absolute error (SMAE) represented overall error, Nash–Sutcliffe efficiency (NSE) represented general discharge dynamics, and logarithmic NSE (logNSE) emphasized low-flow performance. Metric-specific skill scores were also calculated relative to the original CaMa-PyTorch simulations following (Du & Pechlivanidis, 2025).

LSTM postprocessing improved discharge simulations at most stations. Median SMAE decreased from 0.621 to 0.447, indicating lower overall error. Median NSE increased from 0.105 to 0.361, showing better agreement between simulated and observed discharge variability. Median logNSE increased from 0.052 to 0.477, representing the largest improvement among the three metrics and indicating particularly strong gains under low-flow conditions. SMAE improved at 812 stations, corresponding to 88.5% of the evaluated sites. NSE improved at 769 stations, or 83.9%, and logNSE improved at 822 stations, or 89.6%. The median skill scores were 0.270 for SMAE, 0.375 for NSE, and 0.530 for logNSE. The higher skill score for logNSE confirms that the greatest relative improvement occurred in the representation of low flows.

The metric distributions and spatial patterns indicate that LSTM postprocessing improved performance across most basins, although the magnitude of improvement varied among regions. Gains were modest at some stations, and performance declined at a small number of sites. These results show that the LSTM can learn and reduce a substantial portion of the systematic discrepancies in CaMa-PyTorch discharge outputs, but its effectiveness varies across locations and hydrological conditions.

3.5. Evaluation Metrics and Computational Performance

Model fidelity was assessed using four complementary diagnostics: RMSE for overall deviations, KGESS integrating correlation, bias and variability, BIAS for systematic over- or underestimation, and Pearson correlation for temporal agreement. Together, these provide a compact yet robust evaluation framework.

To evaluate computational performance, we conducted module-level profiling and a full-year wall-clock benchmark of the original Fortran CaMa-Flood implementation and the CPU and GPU backends of CaMa-PyTorch. All three implementations used the same model configuration and annual simulation setup. Reported runtimes are expressed as the mean ± standard deviation from repeated runs.

The mean runtimes for the Fortran, CPU-based CaMa-PyTorch, and GPU-based CaMa-PyTorch implementations were 4634.67 ± 11.24 s, 10394.67 ± 110.86 s, and 4080.33 ± 22.48 s, respectively. CPU-based CaMa-PyTorch required 2.24 times as long as the Fortran implementation, primarily because of the additional overhead associated with tensor operations and memory management in PyTorch. GPU parallelization substantially reduced

23335084, 2026, 11, Downloaded from https://agupubs.onlinelibrary.wiley.com/doi/10.1029/2026EA005015 by Hong Kong University Of, Wiley Online Library on [01/11/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

23335084, 2026, 11, Downloaded from https://agupubs.onlinelibrary.wiley.com/doi/10.1029/2026EA005015 by Hong Kong University Of, Wiley Online Library on [01/11/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

this overhead. The GPU-based implementation was 2.55 times faster than CPU-based CaMa-PyTorch and 1.14 times faster than the Fortran implementation, giving it the shortest runtime in the annual benchmark.

Peak host-memory usage was 0.249 GiB for the Fortran implementation, 1.218 GiB for CPU-based CaMa-PyTorch, and 1.521 GiB for GPU-based CaMa-PyTorch. Peak GPU device-memory usage was 0.882 GiB. The larger memory footprint of CaMa-PyTorch reflects the additional storage required for tensor data structures and intermediate computational states. However, the measured memory demand remained well within the capacity of the GPU used in this study.

Module-level profiling showed that most of the computational cost was concentrated in a small number of hydraulic and diagnostic operations, particularly the calculation of discharge and flow velocity. Forcing, output, and auxiliary operations accounted for only a small fraction of the total runtime. Detailed descriptions of the analyzed modules and their relative computational contributions are provided in Tables S5–S6 of the Supporting Information S1.

The current implementation prioritizes code clarity, reproducibility, modularity, and compatibility with artificial intelligence workflows rather than computational speed alone. Nevertheless, the profiling results identify several opportunities for further optimization. These include replacing Python loops with vectorized operations, applying just-in-time compilation, reusing computational buffers, evaluating mixed-precision GPU computation, and improving input and output performance through chunked formats such as Zarr or parallel NetCDF. These strategies may further reduce runtime and memory use while preserving numerical accuracy and model modularity.

Migration fidelity was further evaluated by comparing RMSE, KGESS, BIAS, and Pearson's correlation coefficient from the Fortran and PyTorch simulations against the same observational data set. Station-level relative differences were calculated using the Fortran results as the reference and mapped to identify spatial changes introduced by the migration. Differences were close to zero at most stations, and the few larger discrepancies were spatially isolated and showed no coherent regional pattern. These results indicate that CaMa-PyTorch closely preserves the hydrological performance and spatial behavior of the original Fortran implementation (Figure 3). Together with the runtime analysis, the findings demonstrate that CaMa-PyTorch maintains numerical and hydrological fidelity while benefiting from GPU acceleration.

3.6. Numerical Consistency

To evaluate the numerical consistency of CaMa-PyTorch across different computational backends, we compared the CPU implementation with the GPU implementation. Both simulations used identical forcing data, initial conditions, model parameters, time-step settings, and simulation periods, and covered a continuous 10-day period comprising 240 external model time steps. The results showed that the two computational backends produced generally consistent spatial patterns and temporal evolution. The maximum absolute differences in flooded fraction (fldfrc) and floodplain depth (flddph) were 4.51×10^{-5} and 2.60×10^{-4}, respectively, with differences at every grid cell remaining below 10^{-3}. For outflow, the maximum absolute difference was 6.81×10^{-2} m^3/s, and all grid cell differences remained below 1 m^3/s. These discrepancies remained bounded throughout the simulation and did not produce systematic divergence between the computational backends.

Overall, the CPU and GPU implementations were numerically consistent within the adopted floating-point tolerances, with no appreciable growth in discrepancies during continuous integration. Under the tested configuration and simulation period, CaMa-PyTorch therefore demonstrated consistent results across computational backends and stable numerical behavior. Reproducibility across computing environments requires the same model version, configuration files, forcing data, river network map, input mapping matrix, numerical precision, and software dependencies. The framework can also support global simulations when compatible global forcing data, a CaMa-Flood map at the required spatial resolution, and the corresponding input mapping matrix are available.

4. Discussion

The close agreement between CaMa-PyTorch and the original Fortran implementation across discharge time series, spatial patterns, and flood related state variables indicates that the reformulation using tensor operations

23335084, 2026, 11, Downloaded from https://agupubs.onlinelibrary.wiley.com/doi/10.1029/2026EA005015 by Hong Kong University Of, Wiley Online Library on [01/11/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

preserved the numerical behavior of the river routing and floodplain processes. This agreement establishes the fidelity of the software implementation but does not independently validate the physical assumptions inherited from CaMa-Flood. The GPU benchmarks further indicate that tensor execution can improve computational performance, although irregular floodplain calculations remain an important constraint on hardware utilization.

4.1. Integration of Hydrology and Artificial Intelligence

To evaluate the practical integration of CaMa-PyTorch with machine learning, we conducted a proof-of-concept experiment in which an LSTM was used to postprocess simulated discharge. CaMa-PyTorch first generated baseline simulations based on the prescribed river network and governing hydrodynamic equations. The LSTM was then trained to learn the systematic discrepancies between the simulated and observed time series. The loss function was calculated from the corrected discharge, and gradients were propagated only through the LSTM. Consequently, the routing equations, physical parameters, and numerical integration scheme remained unchanged.

The proof-of-concept results demonstrate that the data-driven model can identify and reduce systematic errors in CaMa-PyTorch discharge simulations. Following LSTM correction, overall streamflow errors and the representation of both high- and low-flow processes improved at most stations, with the most pronounced improvement occurring under low-flow conditions. These results indicate that the current framework can support offline machine-learning applications that operate directly on model outputs.

However, the LSTM corrects simulated discharge only after CaMa-PyTorch has completed the river-routing calculation. The corrected discharge is therefore not fed back into channel storage, floodplain states, or upstream–downstream flux calculations.

4.2. Limitations

Although the proof-of-concept experiment shows that CaMa-PyTorch can support machine-learning postprocessing, the current coupling remains limited to discharge correction after model simulation. The LSTM learns statistical relationships between simulated and observed discharge, but it does not alter channel storage, floodplain states, river and floodplain exchanges, or physical parameters within CaMa-PyTorch. Improved discharge estimates therefore do not demonstrate that the physical sources of model error have been identified or corrected. Such errors may originate from runoff forcing, channel geometry, river-network representation, floodplain processes, reservoir operations, or observational uncertainty, and the present approach cannot distinguish among these sources.

The current LSTM also operates primarily at individual gauging stations using historical simulated and observed discharge. It does not explicitly represent catchment characteristics, river geometry, climate, hydrodynamic states, or river-network connectivity. Because corrections are applied independently after river routing is complete, they are not propagated through the network. As a result, the corrected discharge may not preserve upstream and downstream consistency, network-scale water balance, or spatial coherence. Performance improvements at training stations also may not transfer to unseen stations or ungauged basins. The spatial generalizability of the method therefore requires further evaluation. Its reliability during extreme floods and prolonged low-flow periods also remains uncertain because such events are often poorly represented in the training data.

Extending the current output-level postprocessing approach toward parameter learning within the physical model would require repeated execution of the full river-routing and floodplain simulations under different parameter combinations. However, floodplain computation remains the most irregular and resource-intensive component of CaMa-PyTorch. Grid cells can differ in storage state and in the number of active floodplain layers, requiring dynamic masking, conditional operations, index extraction, and iterative accumulation across layers. Cells within the same layer can be processed concurrently, but calculations across layers within each cell remain sequential because each layer depends on the preceding state. This irregular workload reduces GPU utilization and creates temporary tensors that increase memory consumption. These limitations may become more pronounced in high-resolution or large-domain simulations, particularly when repeated model runs are required for machine-learning applications.

CaMa-PyTorch is not limited to applications over the continental United States. This domain was selected as a representative benchmark for evaluating numerical consistency and computational performance. Applying the

framework globally requires compatible model configuration files, global forcing data, and a CaMa-Flood river-network map at the desired spatial resolution. A runoff mapping matrix is also needed when runoff must be transferred from the forcing grid to the CaMa-Flood computational grid.

4.3. Future Development

Future work should move beyond output correction toward more tightly integrated coupling between physical modeling and machine learning. One promising direction is to use machine-learning models to estimate spatially varying physical parameters from catchment attributes, river geometry, climate, runoff forcing, and other relevant predictors. Parameters that are difficult to observe directly or describe with empirical relationships could be inferred in this way. Before entering CaMa-PyTorch, however, all predicted parameters should be restricted to physically plausible ranges through predefined bounds or suitable transformations.

In this framework, machine learning would complement rather than replace the river-routing equations. The learned parameter fields would be evaluated through forward simulations in CaMa-PyTorch, while discharge, channel storage, floodplain states, and inundation would remain governed by the prescribed river network and hydrodynamic equations. This approach would allow machine-learning outputs to influence internal model processes rather than only adjust the final discharge series. It would therefore provide a more physically interpretable form of integration between hydrodynamic modeling and artificial intelligence.

Large-scale parameter estimation and repeated hydrodynamic simulations will require further gains in computational efficiency. Floodplain calculations remain irregular because grid cells can activate different numbers of floodplain layers and require dynamic masking, conditional operations, and index extraction. Future optimization should reduce temporary tensor creation, reuse computational buffers, improve irregular indexing and flooded-cell selection, and retain only the intermediate variables required for each application.

Global and higher-resolution simulations may also require multi-GPU computing and more efficient memory management. Domain decomposition must preserve upstream and downstream connectivity, exchange water fluxes correctly across computational boundaries, represent bifurcating flow paths, and apply consistent numerical-stability criteria. These advances would allow CaMa-PyTorch to evolve from a continental-scale validation framework into a scalable platform for physically constrained parameter learning, large-scale flood simulation, and hybrid hydrological modeling.

5. Conclusion

Accurate simulation of river discharge is central to hydrological research, climate-impact assessment, and flood forecasting. CaMa-Flood, a widely used global river-routing model based on physical principles, has long relied on a Fortran implementation that constrains interoperability with modern AI frameworks and limits GPU acceleration.

We reimplemented CaMa-Flood v4.23 in PyTorch, retaining its physical algorithms and core hydrological modules while standardizing input preprocessing to ensure consistency. A custom indexing scheme was introduced to replicate Fortran's flexible array handling, preserving the original river-network logic and numerical behavior without altering algorithms. To facilitate validation and reproducibility, we also developed a diagnostic evaluation tool (Python 3.10, PyQt5) that automates preprocessing, station–grid mapping, time-series and spatial comparisons, metric computation (RMSE, KGESS, BIAS, Pearson correlation), visualization, and multi-format export.

We evaluated CaMa-PyTorch at 0.0833° resolution across the continental United States using modular unit tests, direct comparisons with the original Fortran implementation, and daily GRDC streamflow observations from 2005 through 2010. The six-year experiment was designed primarily to assess numerical fidelity and stability during continuous continental simulations. It encompassed seasonal and interannual hydrologic variability and allowed us to examine whether numerical differences accumulated over time. Hydrograph comparisons, residual analyses, and spatial diagnostics showed that CaMa-PyTorch closely reproduced the Fortran outputs across stations, flow conditions, and geographic regions.

The evaluation was not designed to characterize rare floods, prolonged droughts, or the full range of hydroclimatic variability. Longer simulations and metrics tailored to individual events will therefore be needed to

23335084, 2026, 11, Downloaded from https://agupubs.onlinelibrary.wiley.com/doi/10.1029/2026EA005015 by Hong Kong University Of, Wiley Online Library on [01/11/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

assess performance during hydrologic extremes. The PyTorch implementation permits GPU execution and places the model within the same software environment as machine learning tools. In the proof-of-concept experiment, LSTM postprocessing improved simulated discharge, demonstrating the feasibility of linking CaMa-PyTorch outputs with a data-driven correction module.

These findings establish CaMa-PyTorch as a numerically faithful reimplementation of CaMa-Flood with GPU capability and practical compatibility with output level machine learning. The present study does not demonstrate seamless integration, end-to-end differentiability, or automated model optimization. Instead, it provides a technical basis for future research on more tightly integrated physical and data-driven modeling approaches.

Conflict of Interest

The authors declare no conflicts of interest relevant to this study.

Availability Statement

Version 1.0 of CaMa-PyTorch used in this study is preserved in Zenodo (Liu et al., 2026) at https://doi.org/10.5281/zenodo.21911321, is openly available under the Apache License 2.0, and is developed openly at https://github.com/2023ATAI/CaMa-PyTorch.git.

Acknowledgments
The study was partially supported by the Fundamental and Interdisciplinary Disciplines Breakthrough Plan of the Ministry of Education of China (Grant JYB2025XDXM902) and the National Natural Science Foundation of China (Grants 42575159, 42275155, and 42375144).

References

Adera, S., Bellugi, D., Dhakal, A., & Larsen, L. (2024). Streamflow prediction at the intersection of physics and machine learning: A case study of two mediterranean-climate watersheds. *Water Resources Research*, *60*(7), e2023WR035790. https://doi.org/10.1029/2023wr035790

Andreadis, K. M., Coss, M. D. S. P., Gleason, C. J., Simmons, T. T., Tebaldi, N., Bjerklie, D. M., et al. (2025). A first look at river discharge estimation from SWOT satellite observations. *Geophysical Research Letters*, *52*(9), e2024GL114185. https://doi.org/10.1029/2024gl114185

Bates, P. D., Horritt, M. S., & Fewtrell, T. J. (2010). A simple inertial formulation of the shallow water equations for efficient two-dimensional flood inundation modelling. *Journal of Hydrology*, *387*(1–2), 33–45. https://doi.org/10.1016/j.jhydrol.2010.03.027

Bronstert, A., & Plate, E. J. (1997). Modelling of runoff generation and soil moisture dynamics for hillslopes and micro-catchments. *Journal of Hydrology*, *198*(1–4), 177–195. https://doi.org/10.1016/s0022-1694(96)03306-9

Cai, K., He, J., Li, Q., Shangguan, W., Li, L., & Hu, H. (2024). Meta-LSTM in hydrology: Advancing runoff predictions through model-agnostic meta-learning. *Journal of Hydrology*, *639*, 131521. https://doi.org/10.1016/j.jhydrol.2024.131521

Campling, P., Gobin, A., Beven, K., & Feyen, J. (2002). Rainfall-runoff modelling of a humid tropical catchment: The TOPMODEL approach. *Hydrological Processes*, *16*(2), 231–253. https://doi.org/10.1002/hyp.341

Cong, Z., Yang, D., Gao, B., Yang, H., & Hu, H. (2009). Hydrological trend analysis in the Yellow River basin using a distributed hydrological model. *Water Resources Research*, *45*(7), W00A13. https://doi.org/10.1029/2008wr006852

Coxon, G., Freer, J., Westerberg, I. K., Wagener, T., Woods, R., & Smith, P. J. (2015). A novel framework for discharge uncertainty quantification applied to 500 UK gauging stations. *Water Resources Research*, *51*(7), 5531–5546. https://doi.org/10.1002/2014wr016532

Dai, Y., Zeng, X., Dickinson, R. E., Baker, I., Bonan, G. B., Bosilovich, M. G., et al. (2003). The common land model. *Bulletin of the American Meteorological Society*, *84*(8), 1013–1024. https://doi.org/10.1175/bams-84-8-1013

Dey, S., Saksena, S., Winter, D., Merwade, V., & McMillan, S. (2022). Incorporating network scale river bathymetry to improve characterization of fluvial processes in flood modeling. *Water Resources Research*, *58*(11), e2020WR029521. https://doi.org/10.1029/2020wr029521

Du, Y., & Pechlivanidis, I. G. (2025). Hybrid approaches enhance hydrological model usability for local streamflow prediction. *Communications Earth & Environment*, *6*(1), 334. https://doi.org/10.1038/s43247-025-02324-y

Feng, D., Fang, K., & Shen, C. (2020). Enhancing streamflow forecast and extracting insights using long-short term memory networks with data integration at continental scales. *Water Resources Research*, *56*(9), e2019WR026793. https://doi.org/10.1029/2019wr026793

Feng, D., Liu, J., Lawson, K., & Shen, C. (2022). Differentiable, learnable, regionalized process-based models with multiphysical outputs can approach state-of-the-art hydrologic prediction accuracy. *Water Resources Research*, *58*(10), e2022WR032404. https://doi.org/10.1029/2022wr032404

Gelbrecht, M., White, A., Bathiany, S., & Boers, N. (2023). Differentiable programming for Earth system modeling. *Geoscientific Model Development*, *16*(11), 3123–3135. https://doi.org/10.5194/gmd-16-3123-2023

Gleason, C. J., & Smith, L. C. (2014). Toward global mapping of river discharge using satellite images and at-many-stations hydraulic geometry. *PNAS*, *111*(13), 4788–4791. https://doi.org/10.1073/pnas.1317606111

Guo, K., Guan, M., & Yu, D. (2021). Urban surface water flood modelling – A comprehensive review of current models and future challenges. *Hydrology and Earth System Sciences*, *25*(5), 2843–2860. https://doi.org/10.5194/hess-25-2843-2021

Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural Networks*, *4*(2), 251–257. https://doi.org/10.1016/0893-6080(91)90009-t

Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, *2*(5), 359–366. https://doi.org/10.1016/0893-6080(89)90020-8

Kochkov, D., Yuval, J., Langmore, I., Norgaard, P., Smith, J., Mooers, G., et al. (2024). Neural general circulation models for weather and climate. *Nature*, *632*(8027), 1060–1066. https://doi.org/10.1038/s41586-024-07744-y

Lehner, B., & Grill, G. (2013). Global river network delineation derived from HydroSHEDS data at 15 arc-second resolution. *Hydrological Processes*, *27*(15).

Li, Q., Zhang, C., Wei, Z., Jin, X., Shangguan, W., Yuan, H., et al. (2024). Advancing symbolic regression for Earth science with a focus on evapotranspiration modeling. *npj Climate and Atmospheric Science*, *7*(321), 321. https://doi.org/10.1038/s41612-024-00861-5

Liu, Z., Li, Q., Li, Q., Wei, Z., Zhang, C., Yuan, H., et al. (2026). CaMa-PyTorch v1.0 (version v1.0) [Computer software]. *Zenodo*. https://doi.org/10.5281/zenodo.21911321

23335084, 2026, 11, Downloaded from https://agupubs.onlinelibrary.wiley.com/doi/10.1029/2026EA005015 by Hong Kong University Of, Wiley Online Library on [01/11/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

Nearing, G., Cohen, D., Dube, V., Gauch, M., Gilon, O., Harrigan, S., et al. (2023). Global prediction of extreme floods in ungauged watersheds. *Nature*, *627*(8004), 559–563. https://doi.org/10.1038/s41586-024-07145-1

Ng, K. W., Huang, Y. F., Koo, C. H., Chong, K. L., El-Shafie, A., & Ahmed, A. N. (2023). A review of hybrid deep learning applications for streamflow forecasting. *Journal of Hydrology*, *625*, 130141. https://doi.org/10.1016/j.jhydrol.2023.130141

Pearce, A. J., Stewart, M. K., & Sklash, M. G. (1986). Storm runoff generation in humid headwater catchments: 1. Where does the water come from? *Water Resources Research*, *22*(8), 1263–1272. https://doi.org/10.1029/wr022i008p01263

Quilbé, R., Rousseau, A. N., Moquet, J.-S., Savary, S., Ricard, S., & Garbouj, M. S. (2008). Hydrological responses of a watershed to historical land use evolution and future land use scenarios under climate change conditions. *Hydrology and Earth System Sciences*, *12*(1), 101–110. https://doi.org/10.5194/hess-12-101-2008

Rahmani, F., Appling, A., Feng, D., Lawson, K., & Shen, C. (2023). Identifying structural priors in a hybrid differentiable model for stream water temperature modeling. *Water Resources Research*, *59*(12), e2023WR034420. https://doi.org/10.1029/2023wr034420

Ruhi, A., Messager, M. L., & Olden, J. D. (2018). Tracking the pulse of the Earth's fresh waters. *Nature Sustainability*, *1*(4), 198–203. https://doi.org/10.1038/s41893-018-0047-7

Sampson, C. C., Bates, P. D., Neal, J. C., & Horritt, M. S. (2012). An automated routing methodology to enable direct rainfall in high resolution shallow water models. *Hydrological Processes*, *27*(3), 467–476. https://doi.org/10.1002/hyp.9515

Shen, C., Appling, A. P., Gentine, P., Bandai, T., Gupta, H., Tartakovsky, A., et al. (2023). Differentiable modelling to unify machine learning and physical models for geosciences. *Nature Reviews Earth & Environment*, *4*(8), 552–567. https://doi.org/10.1038/s43017-023-00450-9

Tull, N., Passalacqua, P., Hassenruck-Gudipati, H. J., Rahman, S., Wright, K., Hariharan, J., & Mohrig, D. (2022). Bidirectional river-floodplain connectivity during combined pluvial-fluvial events. *Water Resources Research*, *58*(3), e2021WR030492. https://doi.org/10.1029/2021wr030492

Wang, W., Yang, S., Stanley, H. E., & Gao, J. (2019). Local floods induce large-scale abrupt failures of road networks. *Nature Communications*, *10*(1), 2114. https://doi.org/10.1038/s41467-019-10063-w

Wu, H., Adler, R. F., Tian, Y., Huffman, G. J., Li, H., & Wang, J. (2014). Real-time global flood estimation using satellite-based precipitation and a coupled land surface and routing model. *Water Resources Research*, *50*(3), 2693–2717. https://doi.org/10.1002/2013wr014710

Yamazaki, D., Almeida, G. A. M. d., & Bates, P. D. (2013). Improving computational efficiency in global river models by implementing the local inertial flow equation and a vector-based river network map. *Water Resources Research*, *49*(11), 7221–7235. https://doi.org/10.1002/wrcr.20552

Yamazaki, D., Ikeshima, D., Sosa, J., Bates, P. D., Allen, G. H., & Pavelsky, T. M. (2019). MERIT hydro: A high-resolution global hydrography map based on latest topography dataset. *Water Resources Research*, *55*(6), 5053–5073. https://doi.org/10.1029/2019wr024873

Yamazaki, D., Ikeshima, D., Tawatari, R., Yamaguchi, T., O'Loughlin, F., Neal, J. C., et al. (2017). A high-accuracy map of global terrain elevations. *Geophysical Research Letters*, *44*(11), 5844–5853. https://doi.org/10.1002/2017gl072874

Yamazaki, D., Kanae, S., Kim, H., & Oki, T. (2011). A physically based description of floodplain inundation dynamics in a global river routing model. *Water Resources Research*, *47*(4). https://doi.org/10.1029/2010wr009726

Yamazaki, D., Sato, T., Kanae, S., Hirabayashi, Y., & Bates, P. D. (2014). Regional flood dynamics in a bifurcating mega delta simulated in a global river model. *Geophysical Research Letters*, *41*(9), 3127–3135. https://doi.org/10.1002/2014gl059744

Zhou, L., & Liu, L. (2024). Enhancing dynamic flood risk assessment and zoning using a coupled hydrological-hydrodynamic model and spatiotemporal information weighting method. *Journal of Environmental Management*, *366*, 121831. https://doi.org/10.1016/j.jenvman.2024.121831

23335084, 2026, 11, Downloaded from https://agupubs.onlinelibrary.wiley.com/doi/10.1029/2026EA005015 by Hong Kong University Of, Wiley Online Library on [01/11/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License