HydroModels.jl: A Flexible and Efficient Framework for Hydrological Modeling

Abstract

1 Introduction

Hydrological modeling has undergone significant evolution over the past decades, reflecting our growing understanding of water systems and advancements in computational capabilities. This progression can be broadly categorized into three main phases: conceptual hydrological models, deep learning-based models, and the current hybrid approach combining deep learning with physical principles.

Conceptual hydrological models, developed in the mid-20th century, aimed to represent watershed processes using simplified mathematical equations. These models, while effective in many scenarios, often struggled with complex, non-linear hydrological processes and required extensive calibration. As computational power increased, distributed and semi-distributed models emerged, allowing for more detailed spatial representation of watershed characteristics and processes.

The advent of machine learning, particularly deep learning, in the early 21st century brought a paradigm shift to hydrological modeling. Deep learning models, leveraging large datasets and powerful algorithms, demonstrated remarkable capabilities in capturing complex, non-linear relationships in hydrological systems. These data-driven approaches often outperformed traditional models in prediction tasks but lacked the physical interpretability crucial for understanding underlying processes.

The current frontier in hydrological modeling lies in hybrid approaches that combine the strengths of deep learning with physical principles. These physics-informed neural networks and theory-guided data science methods aim to leverage the predictive power of machine learning while maintaining consistency with known physical laws and domain knowledge. This approach promises more robust, interpretable, and generalizable models capable of handling the complexities of hydrological systems across diverse conditions.

The construction of hydrological models has evolved alongside these conceptual advancements. Conceptual models typically involve defining interconnected components representing various hydrological processes, often requiring careful parameterization and calibration. Distributed and semi-distributed models necessitate more complex structures to account for spatial heterogeneity, often incorporating GIS data and advanced routing schemes. Deep learning models, in contrast, rely on designing appropriate network architectures and training procedures to learn from large datasets.

Despite these advancements, the development of hydrological models remains challenging. The diversity of modeling approaches has led to the creation of various model building frameworks, each with its own strengths and limitations. For instance, the Framework for Understanding Structural Errors (FUSE) provides a flexible approach to conceptual hydrological modeling by allowing the combination of different model structures. The Structure for Unifying Multiple Modeling Alternatives (SUMMA) offers a unified approach to hydrological modeling, enabling the representation of multiple modeling approaches within a single framework. Superflex provides a highly flexible framework for conceptual hydrological modeling, allowing for the creation of customized model structures. The Modular Assessment of Rainfall–Runoff Models Toolbox (MARRMoT) offers a collection of conceptual hydrological models and tools for model comparison and evaluation. However, these frameworks often focus on specific types of models or modeling approaches, and may lack the flexibility to easily incorporate machine learning techniques or hybrid approaches. Moreover, the complexity of modern hybrid approaches can make model construction and modification a daunting task, requiring expertise in both hydrology and machine learning, which is not always fully addressed by existing frameworks.

These challenges highlight the need for a unified, flexible framework for hydrological model construction. Such a framework should accommodate various modeling paradigms, from simple conceptual models to complex hybrid approaches, while providing a consistent interface for model development, calibration, and analysis. It should facilitate the integration of different modeling components, allow for easy experimentation with novel approaches, and promote reproducibility in hydrological research.

In response to these needs, we present HydroModels.jl, a comprehensive and adaptable framework designed to streamline the process of hydrological model development and application. By providing a unified platform for diverse modeling approaches, HydroModels.jl aims to accelerate innovation in hydrological science and enhance our ability to address critical water resource challenges in an era of environmental change.

2 Architecture of HydroModels.jl

HydroModels.jl is built upon a modular and flexible architecture, designed to accommodate a wide range of hydrological modeling approaches. The framework's core structure comprises four main classes: Flux, Bucket, Route, and Model. Each of these classes plays a crucial role in representing different aspects of the hydrological system, from individual processes to complete model structures.

2.1 Flux Class

The Flux class serves as the fundamental building block of HydroModels.jl, representing basic hydrological processes. This class embodies the mathematical formulations that govern the movement and transformation of water within various components of the hydrological cycle. By design, the Flux class accommodates a diverse array of water fluxes, ranging from precipitation and evapotranspiration to infiltration and runoff. Its strength lies in its ability to not only encapsulate process-specific equations but also efficiently manage inputs, outputs, and parameters. Moreover, the Flux class demonstrates remarkable versatility in supporting both straightforward algebraic relationships and more complex differential equations. This flexibility enables it to capture a broad spectrum of hydrological processes, spanning from simple linear interactions to intricate dynamic systems, thereby providing a robust framework for comprehensive hydrological modeling.

To accommodate various modeling needs, the Flux class is extended into several specialized subclasses, see the following table:

Flux Type	**Description**	**Supported Dimensions**
SimpleFlux	Implements basic algebraic relationships for straightforward process representations.	Vector, Matrix, Array
StateFlux	Designed for processes involving state variables, allowing for more complex dynamics.	Vector, Matrix, Array
NeuralFlux	Incorporates machine learning techniques, enabling the integration of data-driven	Vector, Matrix, Array

	approaches within the traditional modeling framework.	
RouteFlux	Specialized for routing processes, handling water movement through the system.	Vector, Matrix, Array
UnitHydroFlux	Implements the unit hydrograph concept for rainfall-runoff modeling.	Vector, Matrix, Array
TimeVaryingFlux	Addresses time-dependent processes, allowing for time series inputs.	Vector, Matrix, Array

These diverse Flux subclasses, while tailored to specific hydrological processes, share a common interface and structure within the HydroModels.jl framework. This uniformity is a key strength of the design, ensuring consistency in metadata management, function calls, and overall usage across different flux types. Each subclass, regardless of its specialization, adheres to a standardized approach for handling inputs, outputs, and parameters. This consistency not only simplifies the implementation of new flux types but also enhances the framework's usability, allowing modelers to seamlessly integrate and interchange various flux components within their hydrological models. The unified structure facilitates a modular approach to model construction, where different flux types can be easily combined or substituted without necessitating significant changes to the overall model architecture.

2.2 Bucket Class

The Bucket class is a fundamental component in HydroModels.jl, representing water storage modules within the hydrological system. It serves as a versatile abstraction for various water reservoirs, including soil moisture, groundwater, and surface water bodies, enabling the simulation of dynamic changes in water storage over time.

At its core, the Bucket class dynamically generates functions for solving ordinary differential equations (ODEs) and processing non-StateFlux components. These functions are constructed at runtime using anonymous functions, efficiently integrating multiple flux components. This approach allows for flexible and efficient representation of hydrological processes within the bucket system. The detailed implementation of this process will be discussed later in the paper.

The reason for using this approach is that this method is highly efficient and quick, avoiding matrix update calculations and other computational overheads. It allows for flexible and efficient representation of the relationships between different hydrological processes within the bucket, enabling seamless integration of various flux components and adaptability to diverse modeling scenarios.

The Bucket class seamlessly integrates multiple flux components, manages state variables representing water storage, and implements ODEs for dynamic simulations. This versatility allows modelers to customize the bucket to represent various types of water storage units, adapting to the specific needs of different hydrological scenarios. The primary implementation, HydroBucket, exemplifies this flexibility, offering a customizable model that can be tailored to a wide range of water storage representations.

2.3 Route Class

The Route class plays a crucial role in HydroModels.jl by simulating water movement through landscapes and river networks. It serves as the key differentiator between lumped, multi-node, semi-distributed, and fully distributed hydrological models. While these models may share similar runoff generation processes, their routing calculations vary significantly, leading to distinct model types.

The Route class is extended into three main subclasses, each catering to different spatial representations and routing approaches:

1. SumRoute: This subclass is designed for multi-node models, integrating results from multiple calculation units. It implements a simple weighted accumulation routing scheme, allowing for the aggregation of outputs from various model components.
2. GridRoute: Tailored for fully distributed models, GridRoute specializes in routing processes for grid-based calculation units. It primarily operates on flow direction matrices of watersheds, enabling detailed spatial representation of water movement across a gridded landscape.
3. VectorRoute: This subclass is particularly suited for semi-distributed models, focusing on routing processes in watershed network calculations. It employs directed graph computations to represent and simulate water flow through river networks and channel systems.

The functionality of these Route subclasses, particularly GridRoute and VectorRoute, is enhanced by their unique ability to support different RouteFlux implementations. This feature sets HydroModels.jl apart from many other modeling frameworks. Both GridRoute and VectorRoute construct a unified system of ordinary differential equations that incorporates two types of states: those specific to the RouteFlux components and those pertaining to the Route itself. This approach allows for a more comprehensive and flexible representation of routing processes, enabling the integration of various hydrological phenomena such as time delays and flow attenuation. The detailed implementation of this unified ODE system is provided in the appendix.

The Route class and its subclasses offer several advantages in hydrological modeling. They enable the construction of models with varying levels of spatial complexity, from simple lumped models to sophisticated distributed systems. This flexibility allows researchers and practitioners to choose the most appropriate spatial representation for their specific modeling needs, balancing computational efficiency with the desired level of detail in representing hydrological processes.

2.4 Model Class

The Model class represents the highest-level structure in HydroModels.jl, serving as an abstract base for various hydrological system representations. Its primary implementation, HydroModel, integrates multiple Flux, Bucket, and Route components to create comprehensive watershed models. HydroModel capitalizes on the uniform interfaces of these underlying components, enabling a streamlined and efficient simulation process.

HydroModel orchestrates the overall simulation by sequentially iterating through its constituent components. This approach leverages the standardized interfaces of Flux, Bucket, and Route classes, The consistency of these interfaces allows for flexible combination of modules within the framework while maintaining conceptual integrity. This design philosophy supports a wide range of model configurations, enabling researchers to construct and experiment with diverse hydrological representations tailored to specific research needs or watershed characteristics, all within a coherent and unified modeling environment. Further, the sequential computation simplifies model execution and facilitates easy integration of diverse hydrological processes and spatial representations.

By managing the data flow between components, handling time stepping, and generating outputs, the Model class provides a unified framework for hydrological modeling. This design allows HydroModels.jl to accommodate a wide range of modeling approaches, from simple lumped models to complex, spatially-distributed systems. The architecture's emphasis on modularity and extensibility enables researchers and practitioners to easily implement new processes or modeling techniques within the existing structure, fostering innovation and adaptability in hydrological modeling.

The Model class thus serves as a powerful tool for developing, testing, and applying diverse hydrological models. Its ability to seamlessly integrate various components while maintaining a consistent computational approach underscores the flexibility and robustness of HydroModels.jl as a comprehensive hydrological modeling framework.

3 Methodologies of HydroModels.jl

3.1 Construction Methods

The construction methods are designed to transform user-defined components and parameters into cohesive, computationally efficient structures that represent complex hydrological systems.

3.1.1 Symbolic Programming and Runtime Function Generation

The construction methods in HydroModels.jl are fundamentally based on symbolic programming, metadata extraction, and runtime function generation. This approach forms the backbone of the framework's flexibility and efficiency.

At the core of HydroModels.jl's construction methods is symbolic programming, implemented using the Symbolics.jl package, which is analogous to Python's SymPy. Symbolic programming serves as the foundation for building various types within the framework, providing crucial indicators for inputs, outputs, and parameters of different components.

In the context of hydrological modeling, symbolic programming allows for the representation of intermediate fluxes in the model's computational process. These symbolic variables, when combined with the framework's core classes, enable the representation of complex hydrological processes. Essentially, these variables act as intermediate fluxes in the model, while the classes define how these fluxes are calculated within the model structure.

Following the symbolic representation, the framework generates metadata for each component. This metadata, stored in string format, records essential information about each class, including its inputs, outputs, parameters, and states. This feature allows users to access component information readily and facilitates efficient data computation. The metadata serves as a guide for the framework, enabling quick access to component characteristics and aiding in the orchestration of data flow within the model.

The final step in the construction process involves the building of anonymous functions for specific components. This process varies depending on the type of component:

1. Flux and Bucket Components: For runoff generation modules, anonymous functions are typically constructed at runtime. At its core, the Bucket class dynamically generates two types of functions: one for solving ordinary differential equations (ODEs) and another for processing non-StateFlux components. These functions are constructed at runtime using anonymous functions, which efficiently sequence and integrate multiple flux components. The functions are built once during the model initialization phase and remain fixed thereafter, avoiding repeated construction during model execution. The function generation process utilizes the `build_ele_func` method, leveraging the `RuntimeGeneratedFunctions.jl` and `SymbolicUtils.jl` packages. This approach

employs symbolic computation techniques to seamlessly combine various flux components, including neural network-based fluxes, into a cohesive and efficient computational structure. The resulting functions process input data to output all internal variables of the fluxes, enabling a flexible and powerful representation of hydrological processes within the bucket system.

2. RouteFlux and Route Components: The construction of these components depends on specific requirements. RouteFlux components are first constructed based on the chosen routing method (e.g., Muskingum routing, Nash unit hydrograph). Then, the Route component is built and its routing calculation method is set. These components do not typically require runtime anonymous function construction.

3. UnitHydroFlux Components: For these components, the main requirement is to specify the type of unit hydrograph (e.g., triangular unit hydrograph) to be used in the calculations.

This construction approach offers several advantages, including simplicity in building, strong developmental potential, and high computational efficiency, making HydroModels.jl a powerful and flexible framework for hydrological modeling.

3.2 Computational Methods

HydroModels.jl employs advanced computational strategies to enhance model parameter calibration and computational efficiency, particularly for multi-unit simulations. The framework supports both single-node (lumped models) and multi-node (distributed models) computations through innovative array-based approaches. The key computational methods are as follows:

3.2.1 Dimensional Handling for Input Arrays

In model simulations, input arrays are categorized based on the number of calculation units:

1. Two-dimensional matrices (N × T): Represent single calculation unit inputs, where N is the variable dimension and T is the time length.
2. Three-dimensional arrays (N × M × T): Represent multi-unit inputs, where M is the number of calculation units.

3.2.2 Metadata-Driven Data Extraction and Result Storage

HydroModels.jl employs a sophisticated metadata-driven approach for efficient data extraction and result storage during model execution. This method enhances computational efficiency and facilitates seamless data flow between model components. The key aspects of this approach are:

1. Metadata Generation and Pre-computation Planning: During model construction, metadata for each component is generated and stored, including information about inputs, outputs, and intermediate fluxes. Based on this metadata, the framework pre-determines the sequence of calculations, optimizing the computational flow before simulations begin.

2. Efficient Data Extraction and Result Storage: During execution, the framework uses the pre-stored metadata to extract precisely the required input data for each module and store results in a predetermined order. This approach minimizes unnecessary data processing and allows for efficient concatenation of results.

3. Optimized Data Flow: By leveraging the metadata-driven approach, the framework ensures smooth data flow between components without redundant calculations or data reorganization, managing intermediate fluxes based on the pre-planned sequence.

This metadata-driven method significantly enhances the framework's efficiency, particularly in complex models with multiple interconnected components, allowing for streamlined data management and improved overall model performance.

3.2.3 Slicing and Broadcasting Techniques

Since the generated computation functions support only single time point calculations, the framework implements array broadcasting and slicing techniques for both 2D and 3D array computations:

1. For 2D matrices (single calculation unit):
   - Time dimension slicing creates sub-arrays (an iterator).
   - Parameters are broadcast to match the dimensions of the sliced iterator.
   - Results are computed for each time step using the computation function.
2. For 3D arrays (multiple calculation units):
   - Slicing occurs along both time and calculation unit dimensions.
   - Both input arrays and parameters are sliced for each calculation unit.
   - Computation proceeds similarly to the 2D matrix case for each unit.

The array broadcasting mechanism automatically matches input parameters with sliced arrays, enabling element-wise computation and producing output results of appropriate dimensions.

3.2.4 ODE Solving for Multiple Calculation Units

To ensure simultaneous solving across all calculation units in ODE computations, the framework represents equation state variables as 2D matrices. Each matrix represents the state variables for all calculation units at a specific time point. This approach allows for efficient parallel solving of ODEs across multiple units.

By supporting both 2D and 3D array computations, HydroModels.jl provides a flexible and efficient framework for a wide range of hydrological modeling applications, from simple lumped models to complex distributed systems. This computational approach enhances the framework's ability to handle diverse spatial representations and computational requirements in hydrological modeling.

3.3 Optimization and Solving Methods

HydroModels.jl leverages the powerful SciML ecosystem, particularly the DifferentialEquations and Optimization packages, to provide robust optimization and solving capabilities. This deep integration allows users to access a wide range of state-of-the-art methods while maintaining flexibility and extensibility.

3.3.1 Solving Methods

The framework supports both continuous and discrete ordinary differential equations (ODEs), corresponding to ODEProblem and DiscreteProblem in the DifferentialEquations library. Users can select from a variety of solvers and adjust precision settings to suit their specific modeling needs. This flexibility allows for efficient handling of diverse hydrological processes, from rapid surface runoff to slow groundwater movements.

For continuous ODEs, users can choose from methods such as Runge-Kutta, Adams, or BDF (Backward Differentiation Formula) solvers, depending on the stiffness and characteristics of their system. Discrete ODEs, often used in conceptual models, can be solved using appropriate discrete solvers.

3.3.2 Optimization Approaches and Scenarios

HydroModels.jl offers two main optimization approaches: black-box optimization and gradient-based optimization. Both methods construct objective functions that measure the discrepancy between model outputs and observed data.

1. Black-box Optimization: This approach treats the model as a black box, making it suitable for traditional hydrological models with fewer parameters. It utilizes algorithms that do not require gradient information, such as Particle Swarm Optimization or Differential Evolution.
2. Gradient-based Optimization: Leveraging the Zygote automatic differentiation library and SciMLSensitivity, this method computes gradients of the model computations and ODE solutions. It is particularly effective for distributed models or those with neural network embeddings, allowing for more efficient optimization of large-scale problems.

The framework also supports various optimization scenarios:

1. Neural Network Pre-training: To reduce the complexity of subsequent parameter optimization, HydroModels.jl allows for pre-training of neural networks using specific datasets before embedding them into hydrological models.
2. Distributed Hydrological Model Optimization: Parameters are categorized into three main types: a) Model parameters b) Initial states c) Neural network parameters

   For model parameters and initial states, the framework further groups them by calculation unit types (e.g., Type 1, Type 2). Each parameter group then contains specific model parameter values.

   For neural network parameters, instead of creating separate networks for each calculation unit type (which would lead to exponential growth in parameters), HydroModels.jl implements a shared neural network across all calculation units in a distributed model. To differentiate calculations for different unit types, unit-specific attributes are used as additional inputs to the neural network.

The choice between these optimization methods and scenarios depends on the specific model characteristics, computational resources, and the desired balance between optimization speed and robustness. By providing these diverse optimization capabilities, HydroModels.jl enables researchers to effectively calibrate their models and explore parameter spaces, enhancing overall model performance and reliability across various hydrological modeling contexts.

4 Model Implementation and Case Studies

To illustrate the capabilities and flexibility of HydroModels.jl, this section presents several practical examples of its application in hydrological modeling. These examples demonstrate how the framework can be used to construct, simulate, and optimize various types of hydrological models.

4.1 Lumped Model: Neural Network Embedding in Exphydro Model

The Exp-Hydro model, originally developed by Patil and Stieglitz (2014) and re-implemented by Jiang et al. (2020), is a parsimonious hydrologic bucket-type model designed for daily runoff prediction. It comprises two state variables (snow pack and soil water storage) and simulates five hydrological processes: rain, snow, evaporation, transpiration, and discharge. The model's structure includes five mechanistic processes with process-specific driving forces, model states, and parameters. Despite its simplicity, the Exp-Hydro model has demonstrated excellent performance on the CAMELS dataset in the United States. This performance, coupled with its straightforward structure, makes it an ideal candidate for integration with machine learning techniques.

Building upon the Exp-Hydro framework, several enhanced models have been developed to improve predictive performance by incorporating Physics-Informed Neural Networks (PINN):

- M50: Replaces the original formulas for actual evaporation and runoff with two feed-forward neural networks (NNet and NNq).
- M100: An extension of the M50 model with further neural network integrations.

- PRNN (Process-based Recurrent Neural Network): Combines recurrent neural networks with the Exp-Hydro structure.
- ENN (Embedded Neural Network): Embeds neural networks within the Exp-Hydro framework to enhance specific process representations.

These neural network embedding models demonstrate the flexibility of the Exp-Hydro structure and its potential for integration with machine learning techniques to improve hydrological predictions. To illustrate how HydroModels.jl can be used to construct the Exp-Hydro model and transform it into the M50 model, we have created a comprehensive diagram. Figure 1 provides a clear depiction of this process, showcasing the framework's capabilities in implementing and enhancing hydrological models.

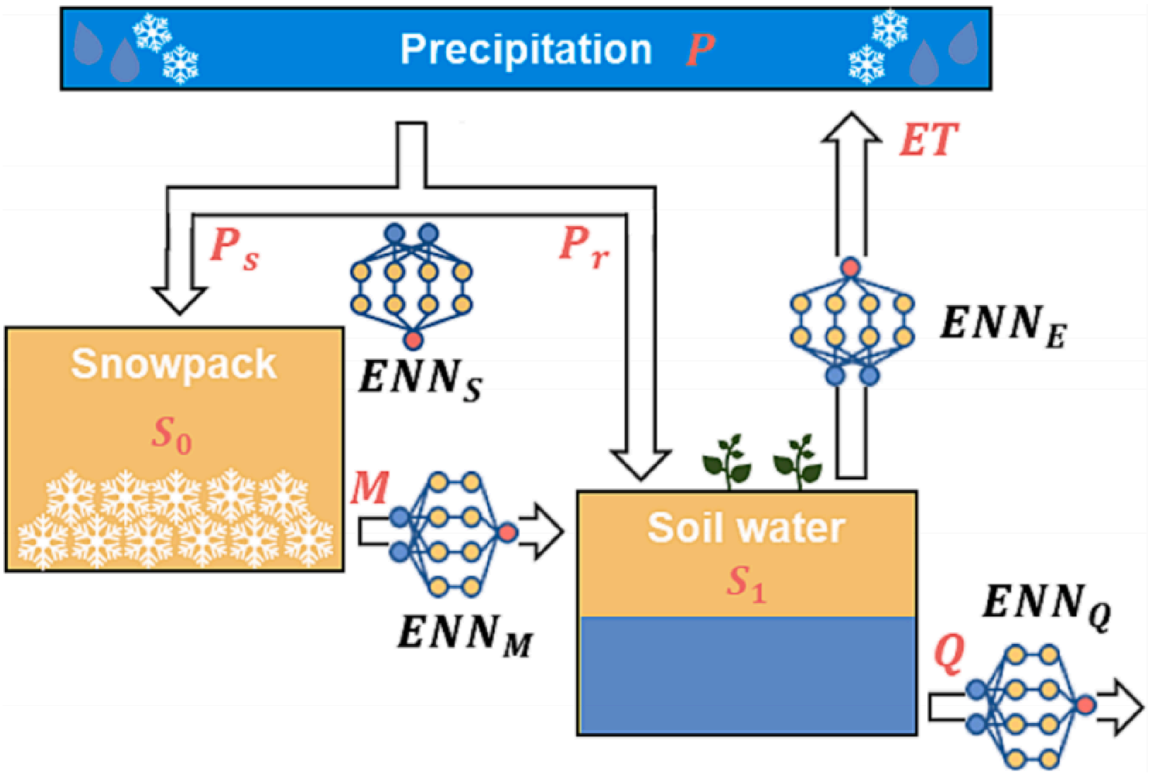

Figure 1: Diagram illustrating the implementation of the Exp-Hydro model and its neural network-enhanced version (M50) using HydroModels.jl. Subpanel (a) showcases the original Exp-Hydro model structure along with its computational formulas, detailing various hydrological processes. Subpanel (b) demonstrates how these structures and formulas are translated into code within the HydroModels.jl framework. Subpanel (c) highlights the improvements made in the M50 model compared to the original Exp-Hydro, particularly focusing on the integration of neural networks. Finally, subpanel (d) illustrates how these enhancements are implemented within the HydroModels.jl framework, showcasing the seamless integration of traditional hydrological modeling with machine learning techniques.

Figure 1 demonstrates HydroModels.jl's ability to translate hydrological formulas into code structures and seamlessly integrate neural networks into existing models. The implementation process in HydroModels.jl involves several key steps:

1. Defining Symbolic Variables: The framework begins by defining symbolic variables for fluxes, using these to construct Flux instances based on calculation formulas. Each Flux instance is provided with input, output, and parameter symbolic variables, along with a calculation formula

expressed in terms of these variables. (An example of this will be provided later to illustrate this process.)

2. Neural Network Integration: For neural network embedding, the framework similarly defines symbolic variables for the Flux inputs and outputs. Neural networks are constructed using Lux.jl, named, and then input as parameters to the Flux instances.
3. State variable equation: The framework constructs StateFlux instances for each computational module based on the incoming and outgoing fluxes. These StateFlux instances represent the state variables of the module and define the mass balance equations, thereby expressing the system dynamics within each bucket component.
4. Model Assembly: Finally, the framework assembles the balance modules into bucket elements and integrates these to form complete models such as the original Exp-Hydro and the enhanced M50 model.

To evaluate the performance and efficiency of our Exp-Hydro and M50 model implementations, we utilized the CAMELS (Catchment Attributes and Meteorology for Large-sample Studies) dataset for validation. This comprehensive dataset provides a wealth of hydrological and meteorological data for 671 watersheds spanning the continental United States. It offers daily measurements of crucial variables such as streamflow, photoperiod, temperature, and precipitation. The dataset's forcing variables are sourced from the high-resolution Daymet dataset, which has consistently demonstrated excellent performance across various modeling scenarios.

To further evaluate the performance and efficiency of our framework, we conducted a comparative analysis using different implementation approaches. We randomly selected 10 catchments from the CAMELS dataset and simulated them using ExpHydro and M50, with four different methods: our HydroModels.jl framework, a custom Julia implementation, JAX, and PyTorch. The results of this comparison are presented in Figure 2, which illustrates both the average simulation accuracy and computational efficiency for each method.

As shown in Figure 2, our HydroModels.jl framework demonstrates competitive performance in terms of both accuracy and efficiency. The average Nash-Sutcliffe Efficiency (NSE) for the HydroModels.jl implementation is comparable to that of the custom Julia implementation and slightly higher than the JAX and PyTorch implementations. This suggests that our framework maintains the high level of accuracy achievable with Julia while providing the added benefits of a structured modeling environment.

In terms of computational efficiency, HydroModels.jl demonstrates performance comparable to the custom Julia implementation. The average computation time per simulation year for HydroModels.jl is similar to that of the custom Julia code, while being substantially faster than both JAX and PyTorch implementations. This efficiency can be attributed to the structured nature of our framework and its effective utilization of Julia's inherent performance capabilities, allowing it to maintain the high computational speed characteristic of Julia.

These results highlight the strengths of HydroModels.jl in balancing accuracy and computational efficiency. The framework not only maintains the high performance characteristics of Julia but also provides a structured, flexible environment for hydrological modeling that can compete with, and in some cases outperform, other popular scientific computing frameworks.

4.2 Distributed Model: Application in neural network-based spatial hydrological modeling

The framework's capabilities extend beyond lumped models to support distributed hydrological modeling, demonstrating its versatility in handling complex spatial computations. This section explores the application of HydroModels.jl in constructing hybrid models that integrate neural networks with distributed hydrological models, using the upper Hanjiang River basin as a case study. We will demonstrate the framework's flexibility in implementing semi-distributed and fully distributed versions of the GR4J model, showcasing its ability to adapt to various spatial discretization schemes and routing methods.

4.2.1 Case Study: Upper Hanjiang River Basin

The upper Hanjiang River basin, located in central China, serves as an ideal testbed for our distributed modeling approach. This region is characterized by complex topography and diverse land cover, presenting challenges typical of many hydrological modeling scenarios. Figure X illustrates the basin's discretization into both sub-basins for the semi-distributed approach and grid cells for the fully distributed model, highlighting the spatial heterogeneity considered in our modeling efforts.

4.2.2 Model Construction Using HydroModels.jl

The construction of multi-node, semi-distributed, and fully distributed hydrological models within our framework involves several key steps, leveraging the flexibility and modularity of HydroModels.jl:

1. Runoff Generation Module: We begin by adapting the GR4J model structure, focusing on the runoff generation components while excluding the unit hydrograph calculations. This modified GR4J serves as the base for our spatial models, ensuring consistent process representation across different spatial scales. Further, we replace the original percolation calculation formula in GR4J with a neural network. This neural network takes soil moisture as input and predicts the percolation rate, allowing for a more data-driven representation of this complex process.
2. Routing Modules: The framework's versatility is demonstrated through the implementation of different routing schemes for each model type:

   a) Multi-node Model: We utilize the original GR4J unit hydrograph module for each node, combined with a WeightSumRoute to aggregate outputs at the basin outlet.

   b) Semi-distributed Model: Based on the sub-basin delineation, we construct a directed graph representing the basin's connectivity. A VectorRoute method is implemented using the Muskingum routing algorithm to describe inter-sub-basin flow relationships.

   c) Fully Distributed Model: Leveraging the grid-based discretization, we compute flow directions using DEM data to generate a D8 flow direction matrix. This informs the GridRoute method, which employs the HD (hydrodynamic) routing approach to simulate water movement between grid cells.

Figure 3 showcases the simulation results for the upper Hanjiang River basin using semi-distributed and fully distributed versions of the GR4J model implemented with HydroModels.jl.

As shown in Figure 3, both the semi-distributed and fully distributed models achieve high accuracy in simulating the streamflow of the upper Hanjiang River basin. The fully distributed model demonstrates slightly higher performance, with a Nash-Sutcliffe Efficiency (NSE) of 0.85 compared to 0.82 for the semi-distributed model. This marginal improvement in accuracy comes at the cost of increased computational complexity, as illustrated in Figure 4.

Figure 4 illustrates the computational efficiency of both the semi-distributed and fully distributed models implemented using HydroModels.jl. The framework demonstrates high performance in executing these complex hydrological models. For instance, the semi-distributed model completes a one-year simulation in approximately 0.5 seconds, while the more intricate fully distributed model requires about 1.25 seconds. Memory usage is also efficiently managed, with the semi-distributed and fully distributed models utilizing around 100 MB and 300 MB respectively.

4.3 Model Evaluation and Optimization

(1) Optimization for M50 model The evaluation and optimization process for our hydrological models, implemented using HydroModels.jl, demonstrates the framework's versatility and efficiency in handling both traditional and machine learning-enhanced approaches.

We begin with the optimization of the ExpHydro model parameters using black-box optimization techniques. This process involves systematically adjusting the model parameters to minimize the difference between simulated and observed streamflow. For the ML-enhanced version, we first pre-train a neural network to estimate percolation rates based on soil moisture content. This pre-trained network is then integrated into the ExpHydro model structure. The combined model undergoes a second round of training, fine-tuning both the hydrological parameters and the neural network weights simultaneously. This two-stage approach allows us to leverage the strengths of both physical process understanding and data-driven learning.

The optimization process for both models is visualized through loss curves, which show the progressive reduction in error as the optimization algorithms iterate. Notably, the ML-enhanced model demonstrates a steeper initial decline in loss, indicating faster convergence to an optimal solution. The computational efficiency of our framework is evident in the rapid execution of these complex optimization procedures, with full calibration runs completing in a matter of minutes on standard hardware.

(2) Optimization for distributed model Extending our approach to distributed modeling, we implement a spatial optimization strategy that accounts for the heterogeneity of the landscape. The parameter structure for this distributed model encompasses three main components: model parameters, initial states, and neural network weights. To capture spatial variability, we define distinct computational units based on soil types and land use categories. This approach allows for a more nuanced representation of hydrological processes across the basin.

The optimization of the distributed model involves a holistic approach, simultaneously adjusting parameters across all spatial units to minimize a global objective function. This process is computationally intensive but is efficiently handled by our framework, leveraging parallel computing capabilities where available. The resulting optimization trajectory reveals interesting patterns of parameter convergence across different landscape units, providing insights into the spatial variability of hydrological processes within the basin.

5 Discussion

A user-friendly and logically structured model development framework can empower professionals to leverage their expertise in creating unique and superior models. Much like how deep learning frameworks such as PyTorch and TensorFlow have supported the flourishing of deep learning research, the development of hydrological models similarly requires an efficient, flexible, and accessible development framework. This section will discuss the characteristics of the HydroModels.jl framework, evaluating its flexibility, efficiency, and other key features.

5.1 Framework Features

1. Ease of Use: HydroModels.jl is designed to provide reliable support for both hydrological model application and development. For those seeking to apply hydrological models for forecasting, users can directly utilize the mature hydrological models provided within the framework. Model inputs can be read from common file formats such as txt or csv, and constructed as NamedTuple types, eliminating the need for additional input file integration. For model development needs, users can build hydrological fluxes, modules, and models by constructing components from Flux to Element to Unit, following a rigorous and logical name-driven construction approach. For model modification requirements, users can employ update, add, and remove methods to alter Flux calculation formulas or element input-output water balances. After completing model simulations, users obtain results in NamedTuple format, including all intermediate calculation fluxes within the hydrological model, without the need for additional function calls.
2. Flexibility: The framework's flexibility is foundational to the versatile application and development of hydrological models. The design of structures such as Flux, Element, and Unit allows users to quickly and freely construct new hydrological models based on calculation formulas. This flexibility is underpinned by the use of NamedTuples for storing and calling intermediate calculation results of hydrological fluxes. As long as input and output fluxes match, models can be flexibly combined to construct various types of hydrological models. However, there is a trade-off between flexibility and computational efficiency. To address this, the framework employs anonymous construction methods to generate ODE calculation functions during model building, avoiding the storage and retrieval of intermediate calculation results and maintaining excellent computational performance while ensuring flexibility.
3. Reusability: The field of hydrological modeling has produced numerous mature and widely applied models, including calculation formulas and modules for various hydrological processes. HydroModels.jl facilitates the reuse of these classic model components, allowing users to combine calculation formulas and modules from different models, opening up infinite possibilities for hydrological forecasting. The framework's design, inspired by MARRMoT, constructs calculation formulas according to different flux types, enabling users to call these formulas based on input, output, and parameter names (leveraging Julia's multiple dispatch feature).
4. Decoupling: HydroModels.jl decouples model construction from parameter setting. Unlike some frameworks where model construction requires simultaneous specification of parameter values, HydroModels.jl separates these processes. Model parameters are provided as inputs alongside data during model simulation. This design philosophy simplifies the model construction process, supports model reusability, and avoids repetitive model building during parameter calibration, thereby improving calibration efficiency.
5. Differentiability: The framework supports differentiable programming, enabling the use of gradient-based optimization methods and the development of physics-informed neural networks (PINNs) for hydrological modeling.
6. Integration with Modern Algorithms: HydroModels.jl leverages the SciML ecosystem, particularly packages like Lux.jl, DifferentialEquations.jl, and Optimization.jl. This integration allows the framework to utilize mature ecosystems and efficient solvers for PINN model development, runoff simulation, and parameter calibration.

5.2 Framework Limitations and Future Development

1. Extension to Distributed Models: While currently focused on lumped models, future development will consider spatial computations based on HydroModels.jl. This expansion will enable the construction of semi-distributed and distributed hydrological models, implementing multi-model optimization for nodes in dPL-HBV, river-network, and dPL-distributed models. Large-scale

distributed hydrological models often involve parallelization and GPU acceleration techniques to enhance simulation efficiency. Future work will explore how to fully utilize Julia's computational performance, including the use of StructArrays.jl, GPU acceleration, and metaprogramming.

2. Enhanced Element Reusability: To facilitate the combination of calculation modules from different models, future work may focus on developing a unified model template that constrains the input and output fluxes of various module types. This standardization would support the integration of computational modules from different models, enhancing the framework's flexibility and reusability.
3. Graphical Programming Interface: To provide a more convenient user experience for hydrological forecasting practitioners and researchers, future development may include a graphical user interface. This interface would allow users to construct fluxes by inputting formulas and build elements and units through topological connections, saving and applying these constructions to simulation and calibration work with minimal coding required.

In conclusion, HydroModels.jl offers a powerful and flexible framework for hydrological modeling, addressing many of the challenges faced by researchers and practitioners in the field. Its modular design, integration of traditional and machine learning approaches, and efficient computational strategies position it as a valuable tool for advancing hydrological science. Future developments will further enhance its capabilities, particularly in distributed modeling and user accessibility, ensuring its continued relevance in addressing complex water resource challenges in an era of environmental change.

6 Appendix

6.1 build_ele_func

1. Variable Preparation:
   - Collects variables from all input functions (funcs and dfuncs).
   - Converts these variables to symbols and creates a named tuple.
2. Function Parameter Construction:
   - Extracts input, state, and parameter variables from the prepared variables.
   - Collects neural network parameters (if present).
3. Assignment and Output List Construction:
   - Iterates through each input function, handling both regular and neural fluxes.
   - For neural fluxes, it matches inputs to neural network inputs and outputs to calculation results.
   - For regular fluxes, it matches output variables to their corresponding expressions.
   - Builds a comprehensive list of assignments and outputs.
4. Function Generation:
   - Constructs argument lists for both flux and state functions.
   - Uses `@RuntimeGeneratedFunction` to create efficient, dynamically generated functions.
   - For flux functions, it combines all flux outputs into a single vector function.
   - For state functions (if present), it creates a separate function for differential equations.
5. Output:
   - Returns two functions: a merged flux function and a merged state function (if applicable).