3 Implement hydrological models using LumpedHydro.jl

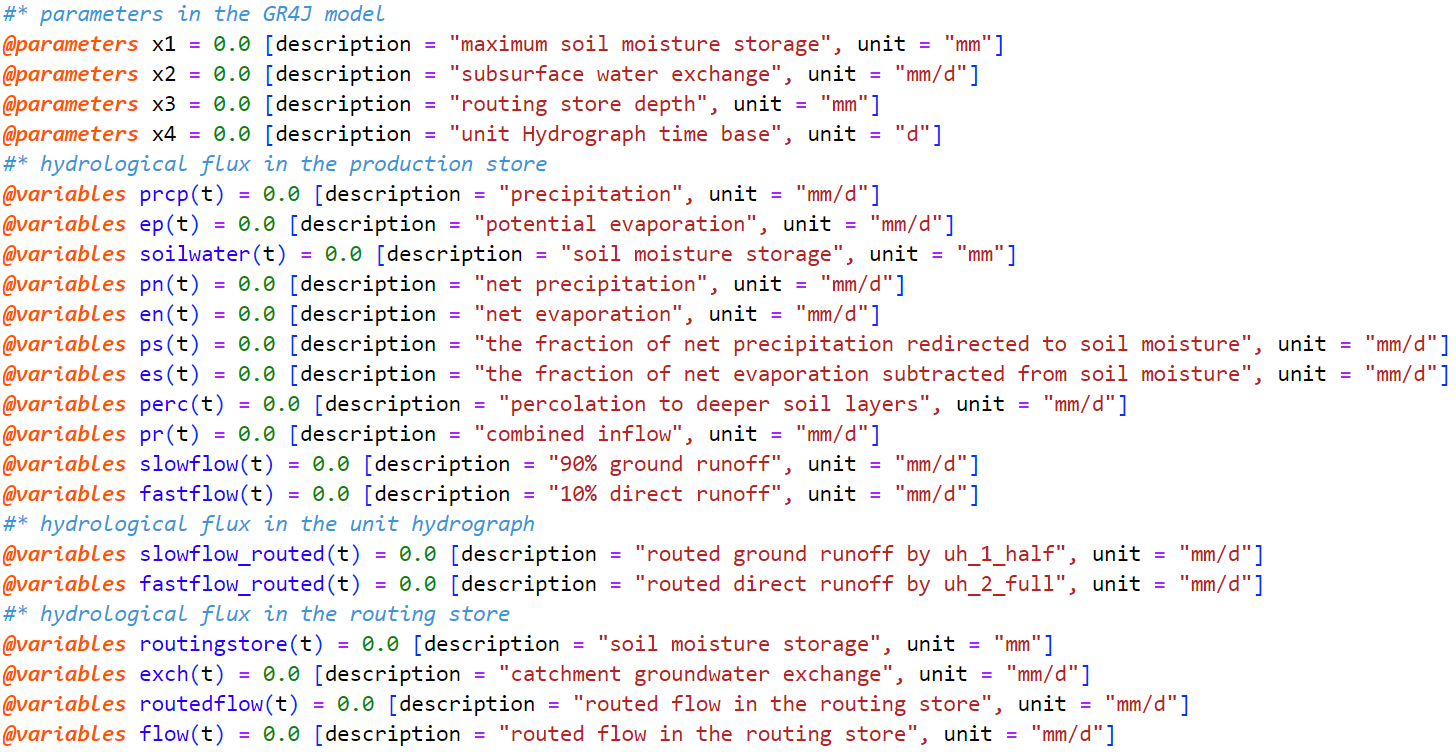
With the introduction of physics-informed neural networks (PINNs), a new paradigm has emerged in hydrological modeling. In recent years, several innovative and reliable PINN-based lumped hydrological models, such as M50, M100, dPL-HBV, and PRNN, have been developed.

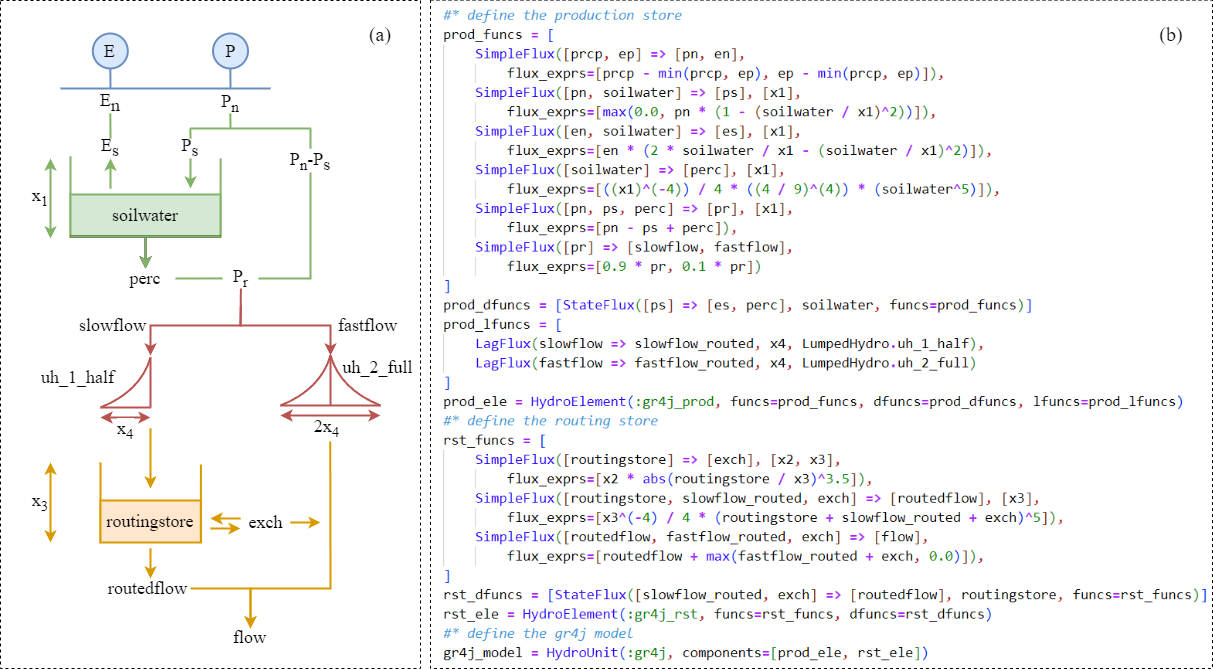
After discussing the basic types and methods of LumpedHydro.jl in the previous section, this section will provide a detailed overview of the construction and training of some representative models, including GR4J, M50, and dPL-HBV.

3.1 Implementing the concept hydrological model: GR4J

GR4 (Perrin et al., 2003) is a lumped bucket-type daily rainfall–runoff model with four free parameters. It is widely used for various hydrological applications in France (Grouillet et al., 2016; van Esse et al., 2013) and other countries (Dakhlaoui et al., 2017; Seiller et al., 2017). The model has demonstrated strong performance across a wide range of catchments (Coron et al., 2012).

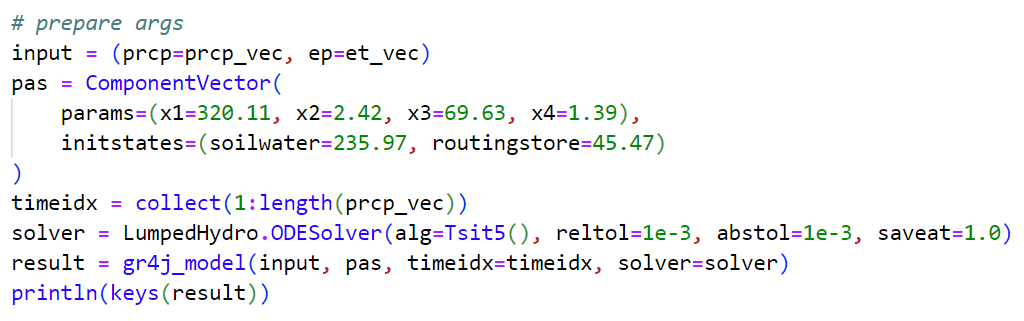
In this section, the construction of the GR4J model is based on the formulas provided in Santos's paper. The unit hydrograph, however, uses the two types described in Perrin's paper, consistent with the GR4J model provided by MARRMoT. The model structure and construction code are illustrated in the following figures.



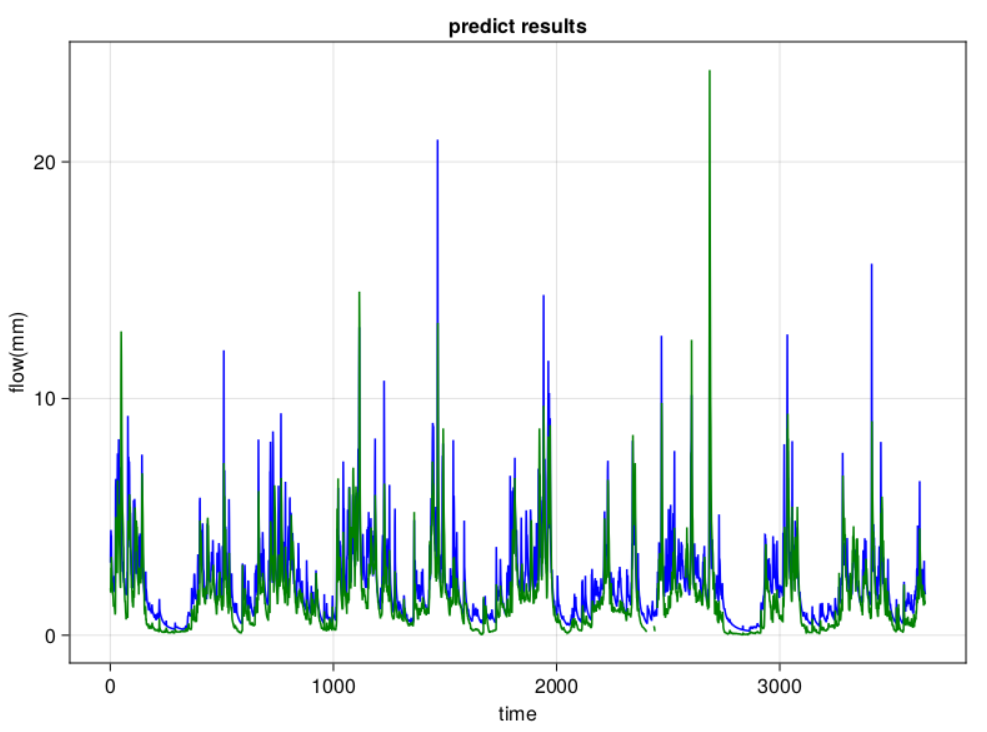


As illustrated, using the GR4J model's formulas, we can easily and directly customize the model with LumpHydro.jl. This is achieved through a variable-based construction style. The code begins by defining symbolic variables with ModelingToolkit.jl, as shown in Figure A.1, which is similar to Matlab. Following the GR4J structure, the code sequentially builds two computational modules: production and routing. We can express the computational formulas in Julia, specifying the input fluxes and parameters for the formulas to construct Flux structure. These are then assembled into two computational modules, ultimately forming a hydrological unit.

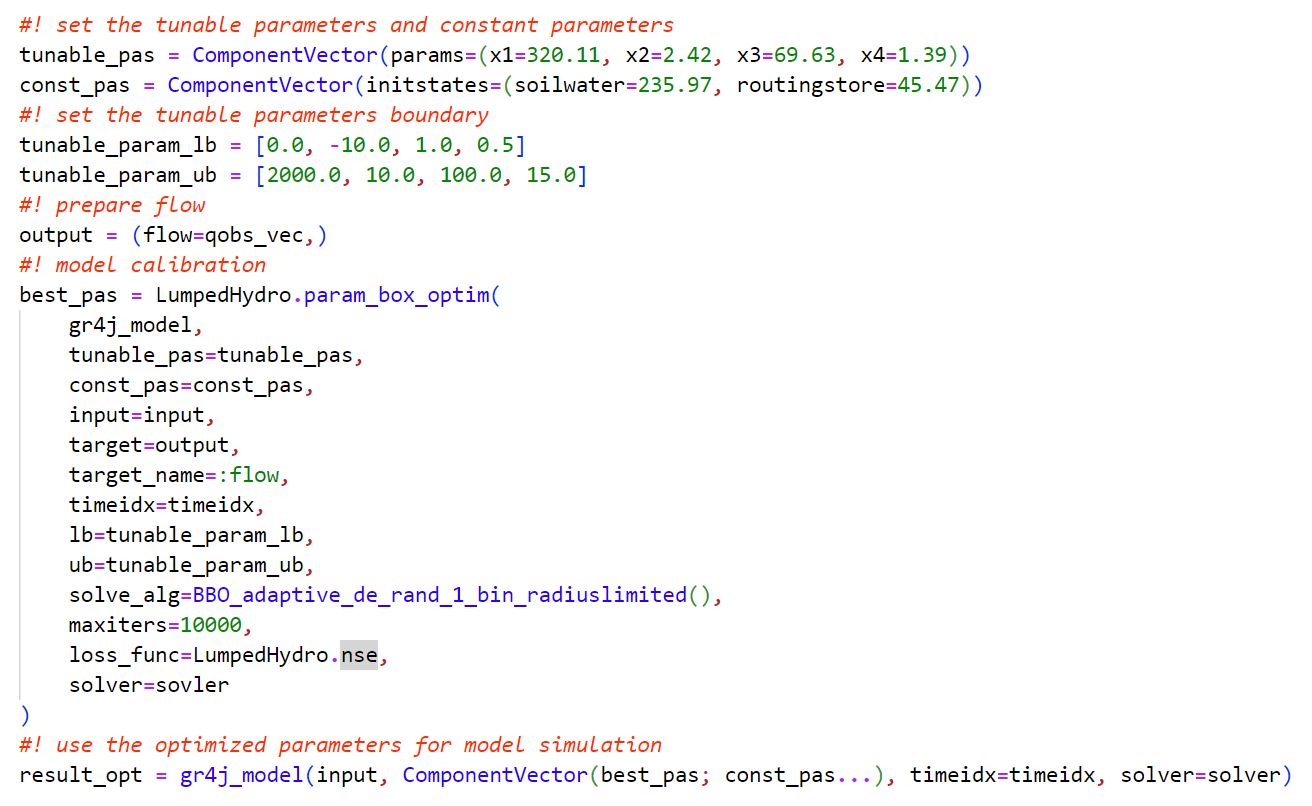
For model testing, we selected the L0123001 test data from airGR. The data was read from a CSV file and integrated into a NamedTuple type as the model input. The NamedTuple stores input data as key-value pairs and keeps track of the computation state throughout the process, eventually returning all intermediate computation states, including the runoff prediction results. The input parameters use the Component Vector type[[1]](#footnote-1), the timeidx variable serves as an index for data interpolation, while the solver is a wrapper for the solve function, the setting of the wrapper are 1e-3, 1e-3, TSit5() for solve algorithm, relative tolerance and absolute tolerance, respectively.



The simulation results of the GR4J model are shown in the figure. The GR4J model successfully replicates the runoff variation process. Comparing these results with those calculated by airGR shows that the two packages produce consistent results, confirming the accuracy of the code execution. The calculation efficiency of the model was evaluated, and the computation time for a single simulation was found to be 50 milliseconds.



To improve the model's simulation accuracy for this dataset, the code needs to provide parameters to be optimized, such as model parameters x1, x2, x3, and x4, as well as fixed initial state parameters (soilwater, routingstore). Additionally, the optimizer (Differential Evolution), the number of iterations (10,000), the objective function and its calculation flux (NSE and target), and the parameter search bounds must be set. Finally, other arguments required for model simulation, such as input, timeidx, and solver, should be provided. This setup is illustrated in the code.

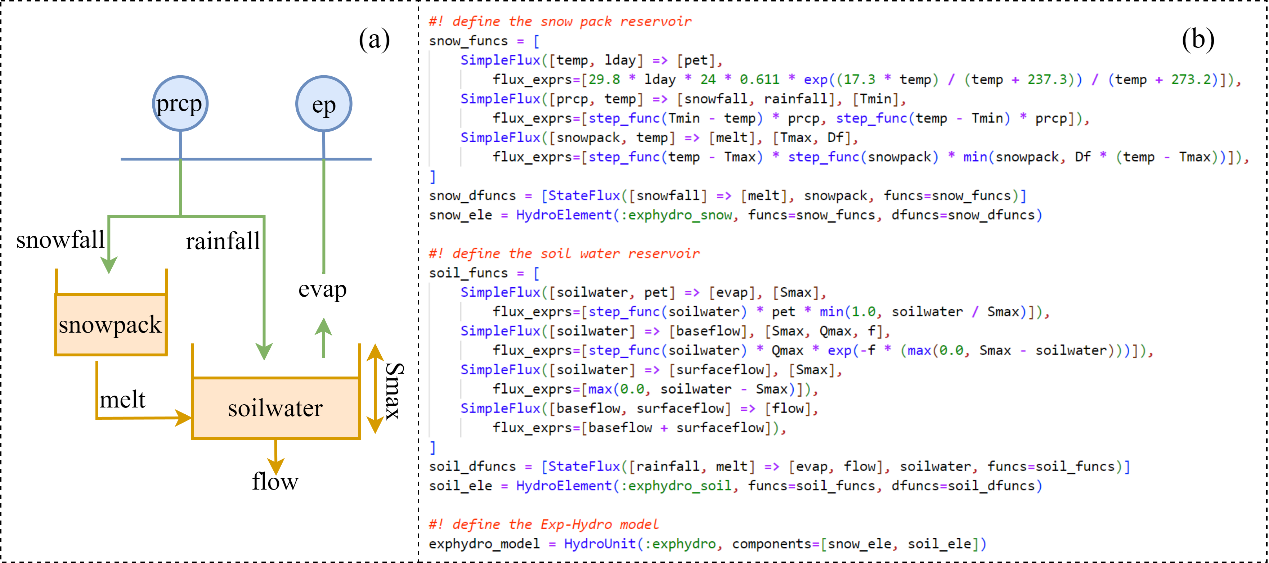


After optimization, the parameter calibration results showed an improvement in the NSE value from 0.53 to 0.78, consistent with the airGR case.

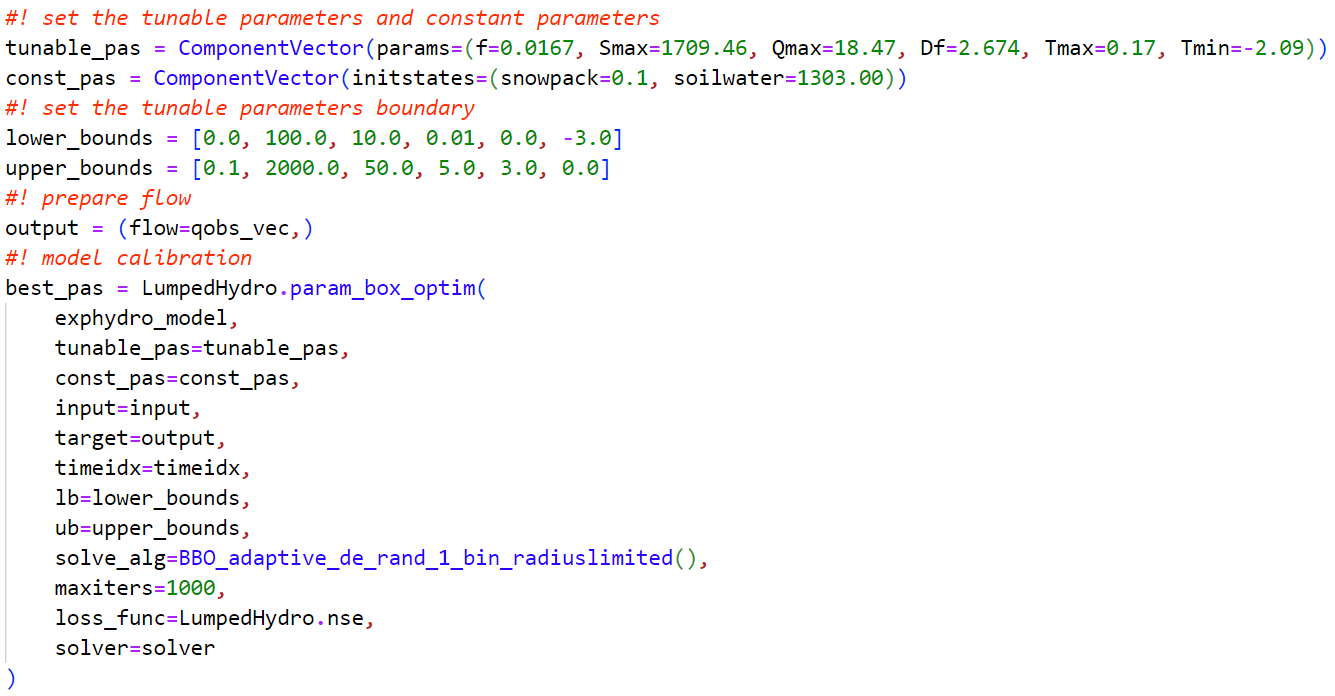
3.2 M50和Exp-Hydro

The Exp-Hydro model is a typical hydrologic bucket-type model that comprises only two state variables: snow pack and soil water storage. It includes five hydrological fluxes: rain, snow, evaporation, transpiration, and discharge. This simple model structure is commonly used for daily runoff prediction and has shown excellent performance on the CAMELS dataset in the United States.

Several models, such as M50, M100, PRNN, and ENN, have enhanced their predictive performance by building PINN models based on Exp-Hydro. For instance, the M50 model replaces the formulas for actual evaporation and runoff with two feed-forward neural networks (NNet and NNq). The model structure and code write in LumpedHydro.jl are illustrated in the figure.

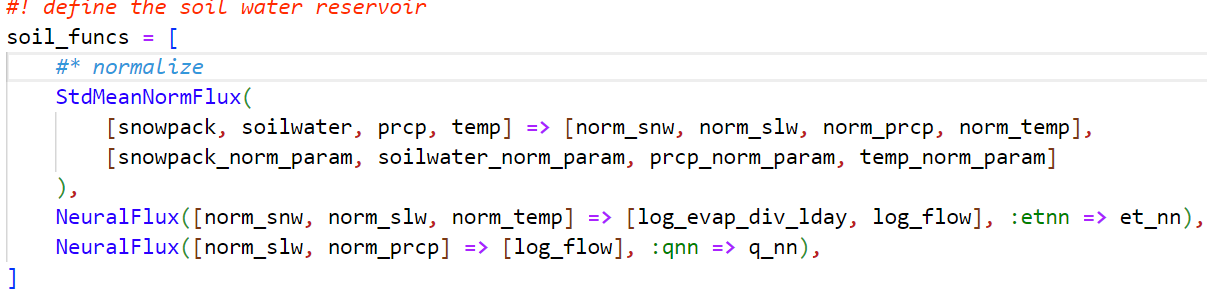


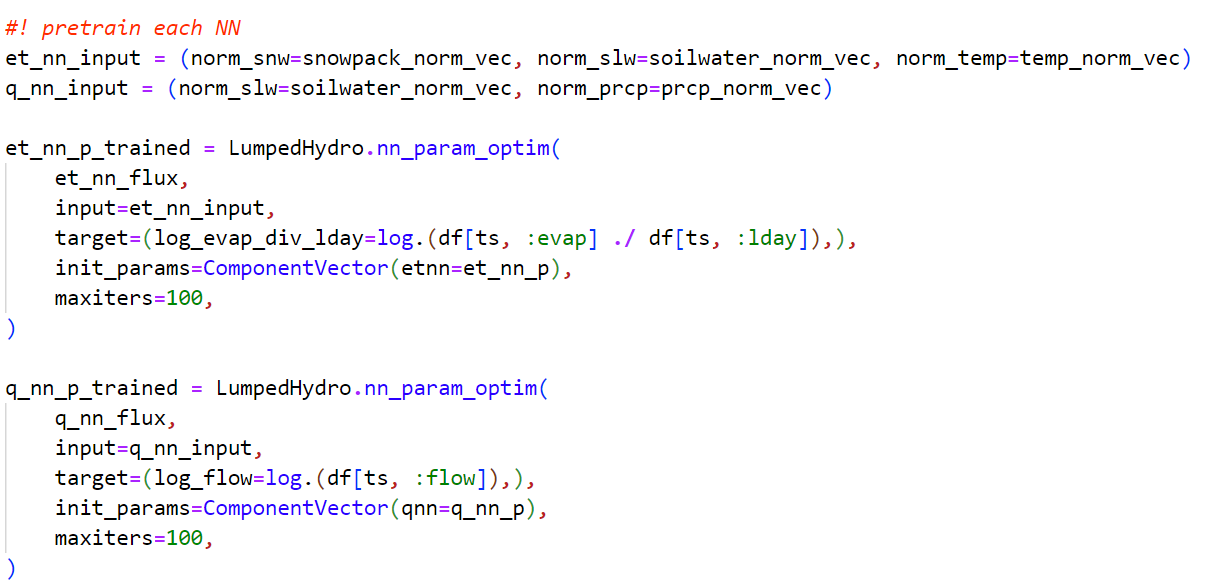
According to the training steps of the M50 model, we first constructed the Exp-Hydro model using the package. From the code, it's evident that this model consists only of two types: SimpleFlux and StateFlux, making its structure straightforward and easy to comprehend. Next, we utilized observed data from the CAMELS dataset, specifically station number 01013500, to simulate the runoff process using the Exp-Hydro model. Simultaneously, we optimized model parameters based on observed runoff data, as illustrated in the figure below.



In the figure, we set the parameters f, Smax, Qmax, Df, Tmax, and Tmin of the ExpHydro model as the parameters to be optimized. The initial values of snowpack and soilwater are fixed parameters. The types of input arguments and the optimization algorithm are consistent with section 3.1.

After obtaining the optimized parameter results, they can be used for runoff simulation to generate simulated results and hydrological fluxes from intermediate calculations in the model. Following this, in the M50 model, two neural network models named etnn and qnn are employed to predict the logarithms of evaporation (log(evap/lday)) and flow (log(flow)). The inputs to these models consist of normalized values of snowpack, soil water, rainfall, and temperature.The neural network models etnn and qnn will generate a new soil element object based on their predictions, as depicted in the figure. These models will be pre-trained using hydrological fluxes obtained from intermediate calculations within the model, as shown in the figure.





In the figure, the nn\_param\_optim method is called to optimize the parameters of the neural networks embedded within NeuralFlux using input data and simulated hydrological fluxes. It's worth noting that Lux.jl framework is selected for building the neural networks, which supports most packages within SciML. Additionally, the framework ensures a high degree of decoupling between model parameters, states, and model objects, directly incorporating them as inputs (et\_nn\_p\_trained and q\_nn\_p\_trained) for simulation and optimization.

After completing the pre-training phase, the param\_grad\_optim method is used for secondary training of the model, as depicted in the figure. In the M50 model, the author focuses solely on secondary training of the neural network parameters (nn\_params). Therefore, other parameters and model states are fixed parameters during this process. This approach avoids the need to distinguish between parameters to be optimized and fixed parameters during the optimization process.

Additionally, unlike other optimization methods, this approach also requires specifying the auto differential type used to construct the optimization problem. Typically, gradient-based optimization algorithms depend on derivatives of the optimization objective, and the auto differential type indicates the method used to compute these derivatives. Different methods can impact the efficiency of subsequent optimizations. For this purpose, AutoZygote is recommended for parameter optimization involving neural network model parameters, while AutoFiniteDiff and AutoForwardDiff are suitable for cases with fewer parameters.

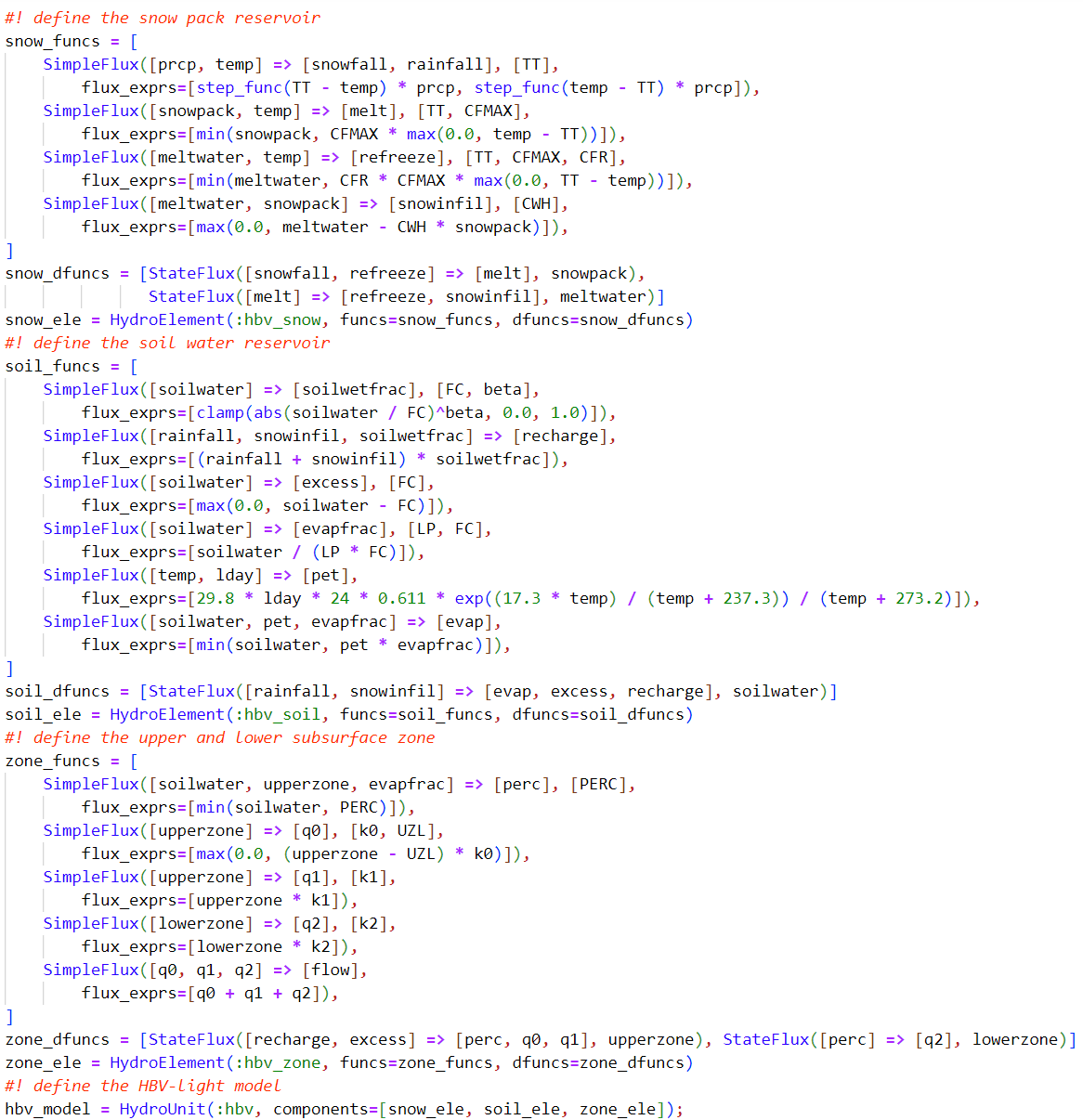
Comparing the simulation results between the original M50 code and the results from this experiment, the simulated outputs are similar. However, due to the construction of the ODE problem, M50 integrates two hydrological buckets into a single ODE problem for solution, whereas this experiment separates the two buckets into two distinct ODE problems. This difference in approach may lead to slight variations in computational results.

【M50训练结果图，包括预训练结果和二次训练结果】

3.3 HBV-light和dPL-HBV

The DL for Parameter Calibration approach presents another form of the PINN-hydrology model, with the dPL-HBV model being a prominent example. This model is based on the HBV-light structure (including snowpack, soilwater, and surface water zone). It uses an LSTM model to predict parameters of the HBV-light model (both static and dynamic parameters) based on observed data and watershed conditions. Additionally, a neural network model is constructed to predict recharge flux.

By predicting some parameters of the HBV model (both static and dynamic) using LSTM and incorporating them into the formulas for calculation, the model parameters exhibit regionalization characteristics and temporal variability. The codes for HBV-light and dPL-HBV models are illustrated in the figure.



In the HBV-light model code, the implemented continuous ODE equations compute hydrological fluxes such as melt and refreeze without promptly adjusting state variables. Instead, the changes in state variables are calculated afterward for solving the equations. This differs from the code referenced in the paper. However, it's assumed that all flux computations occur simultaneously within a time period. Therefore, frequent adjustments to state variables during one period, which could affect the calculation of other fluxes, may be considered unreasonable.

In the dPL-HBV model code, dynamic parameters beta and gamma are defined as dimensionless fluxes rather than fixed parameters. In this experiment, rainfall, temperature, potential evaporation, and soil moisture serve as inputs to a fully connected neural network to obtain dynamic parameter values for each time period. Additionally, a similar approach to the M50 model is adopted for constructing a neural network for recharge. It's worth noting that the model constructed in this experiment differs structurally from the dPL-HBV paper in two aspects: firstly, the paper utilizes an LSTM model and historical 365-day observation data as inputs, possibly to predict dynamic parameters based on the cumulative effect of observed data, especially precipitation and potential evaporation. In contrast, this experiment uses a fully connected neural network and incorporates observed data along with soil moisture to replace this model. Secondly, this experiment focuses on a single watershed and thus does not consider regional characteristics as model inputs for parameter prediction. Consequently, the remaining fixed parameters of HBV are calibrated through gradient descent rather than being outputs of the neural network model.

The model training process follows three steps: initial calibration using static parameters, pre-training of the neural network, and subsequent re-training of model parameters. The predicted results of the model are shown below.

【预测结果图，参数变化图，径流预测结果图】

1. ComponentVector, based on ComponentArrays.jl, extends NamedTuple functionality, making it suitable for solving ODE problems and optimization tasks [↑](#footnote-ref-1)