1 Workflow

The workflow of the project is as follow:

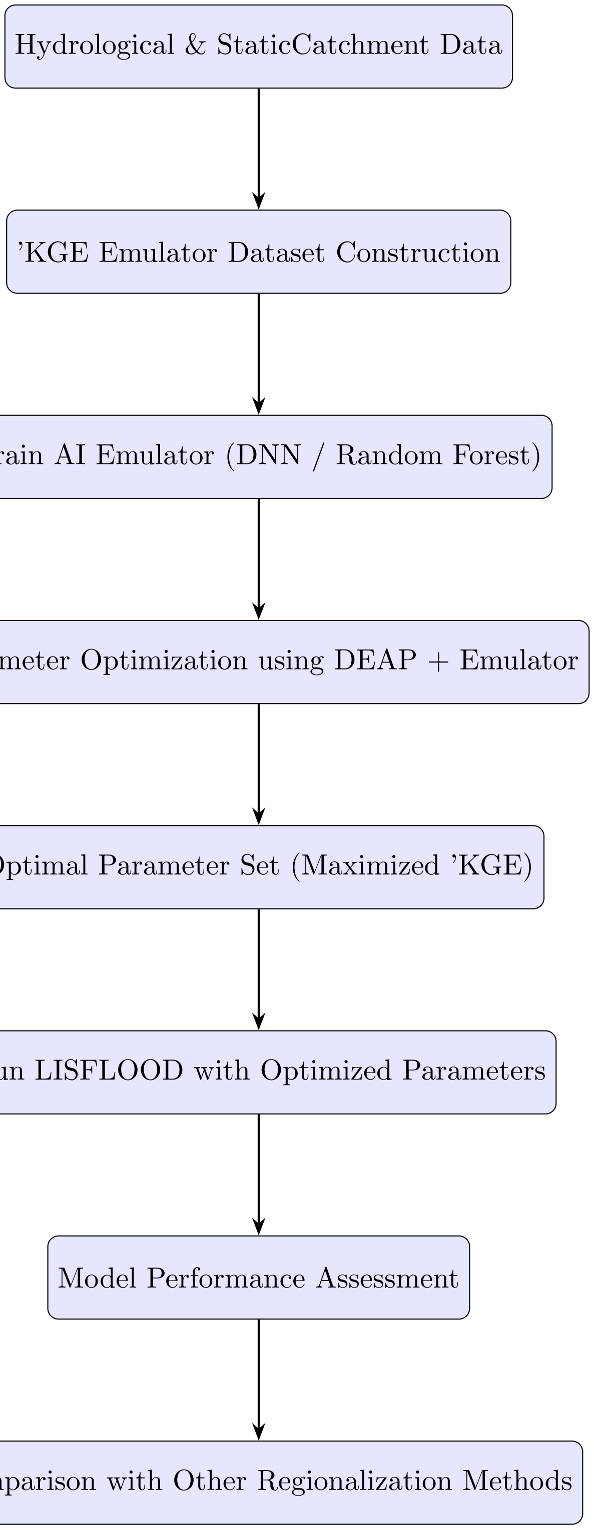
Hydrological & StaticCatchment Data
'KGE Emulator Dataset Construction
Train AI Emulator (DNN / Random Forest)
Parameter Optimization using DEAP + Emulator
Optimal Parameter Set (Maximized 'KGE)
Run LISFLOOD with Optimized Parameters
Model Performance Assessment
Comparison with Other Regionalization Methods

Figure 1: Workflow for 'KGE emulator-based parameter optimization and evaluation in the LISFLOOD model.

To run the code,

1. 'KGE Emulator:
   - Run using: the YAMl file
   - The variable needed should always be added to the list_variable.py file since that file is the one determining which variable are keep or not.
   - python experiments\run_experiment.py exp1_dnn.yaml or exp1_RF.yaml
2. Deap Algo:
   - Run using: the YAMl file
   - python optimization\run_optimization.py deap_config.yaml
3. Run Lisflood with said paramter:
   - Run as usual the complete set of catchment in the test separation.

1.1 'KGE Emulator Code Architecture and Workflow

The 'KGE emulator is designed as a modular and configurable machine-learning pipeline aimed at approximating the relationship between catchment attributes, calibrated Lisflood parameters, and the resulting Kling–Gupta Efficiency ('KGE). The codebase supports both deep neural networks (DNNs) and Random Forest (RF) models. Configuration is fully externalized using YAML files.

1.1.1 Main Execution Workflow

The main execution logic of the AI training is implemented in a single entry-point script, which orchestrates the entire experiment. The workflow can be summarized as follows:

1. **Configuration loading**: Experiment parameters are read from a YAML configuration file, including model type, hyperparameters, data paths, and logging options.
2. **Experiment setup**: A dedicated experiment directory is created, containing subfolders for model checkpoints, plots, and diagnostics.
3. **Data preparation**: Input features, parameter histories, and 'KGE targets are loaded, merged, normalized, and split into training, validation, and test datasets.
4. **Model instantiation**: Based on the configuration, either a DNN or RF emulator is created using a common abstract interface.
5. **Training and evaluation**:
   - For RF models, cross-validation is performed using scikit-learn, followed by training on the full training set and evaluation on validation and test sets.
   - For DNN models, training is performed using PyTorch with mini-batch gradient descent and explicit validation loss tracking.
6. **Diagnostics and outputs**: Model predictions are inverse-transformed to physical 'KGE units and used to generate diagnostic plots, spatial error maps, and performance metrics.

This design ensures a clear separation between experiment configuration, data handling, model definition, and evaluation logic.

1.1.2 Data Loading and Augmentation

The data loading pipeline is implemented in a dedicated module and is responsible for constructing the learning dataset from heterogeneous sources. Input features consist of static catchment attributes and climate indices, while targets are derived from Lisflood parameter calibration histories.

For each catchment:

- Static attributes are merged from multiple CSV sources using a common catchment identifier.
- Parameter history files are loaded, cleaned, and expanded such that each parameter realization becomes an independent training sample.
- The corresponding 'KGE value is extracted and associated with each realization.

Missing parameters are filled using global default values derived from reference calibration files. Samples with invalid values (NaNs) or extremely poor performance ('KGE below a fixed threshold) are removed. An optional equifinality filter allows exclusion of catchments with excessively high parameter dispersion.

The dataset is then split into training, validation, and test subsets using predefined catchment ID lists.

1.1.3 Normalization and DataLoaders

Input features are normalized using a standard score transformation (zero mean, unit variance). The 'KGE target is transformed using a quantile-based normalization, mapping its empirical distribution to a Gaussian distribution. This choice stabilizes training and improves learning in the tails of the 'KGE distribution.

After normalization:

- Data are converted to PyTorch tensors.
- Dedicated `DataLoader` objects are created for training, validation, and testing.

The normalization scalers are stored and reused during post-processing to inverse-transform model predictions back to the original 'KGE scale for evaluation and visualization.

1.1.4 Model Architecture

All emulator models inherit from a common abstract base class defining a shared interface for training, prediction, saving, and loading. This abstraction allows seamless switching between model types.

The DNN emulator consists of a fully connected feedforward architecture with configurable depth, width, activation functions, and dropout regularization. The model supports either a single multi-output network or independent subnetworks per output variable.

The RF emulator relies on a scikit-learn Random Forest regressor and supports both native multi-output prediction and independent forests via a wrapper. Hyperparameters such as tree depth, number of estimators, and feature subsampling are fully configurable through the experiment YAML file.

1.1.5 Loss Function for DNN Training

For neural network training, a custom weighted mean squared error loss is employed to address systematic biases observed in 'KGE prediction. The loss function applies asymmetric penalties depending on the position within the 'KGE distribution:

- Overestimation is penalized more strongly for low-'KGE samples.
- Underestimation is penalized more strongly for high-'KGE samples.

This asymmetric formulation mitigates overly optimistic predictions for poorly performing parameter sets and encourages accurate reconstruction of high-performing calibrations. The final loss is computed as the mean over all samples in the mini-batch.

Overall, the emulator architecture is designed to balance flexibility, reproducibility, and physical interpretability, making it suitable for large-scale parameter regionalization experiments within the GloFAS v5 and Lisflood modelling framework.

1.2 Emulator-Based Parameter Optimization Using DEAP

To efficiently identify optimal LISFLOOD parameter sets, a genetic algorithm optimization framework was implemented using the DEAP (Distributed Evolutionary Algorithms in Python) library. The optimization relies on the AI-based 'KGE emulator trained in the previous subsection, which replaces computationally expensive hydrological simulations during the search process.

1.2.1 Optimization Strategy

For each catchment in the test sub-dataset, the objective is to maximize the Kling–Gupta Efficiency ('KGE) by optimizing a subset of LISFLOOD parameters while keeping catchment-specific static and climatic variables fixed. Let

$$\mathbf{x} = \{\mathbf{x}_{\text{fixed}}, \mathbf{x}_{\text{opt}}\}$$

denote the full input vector to the emulator, where $\mathbf{x}_{\text{fixed}}$ represents non-optimized catchment descriptors and $\mathbf{x}_{\text{opt}}$ corresponds to the LISFLOOD parameters subject to optimization.

The trained AI emulator (either a Deep Neural Network or a Random Forest model) is used to predict the expected 'KGE value:

$$\widehat{\text{'KGE}} = f_{\text{AI}}(\mathbf{x})$$

This surrogate-based approach allows rapid exploration of the parameter space without repeatedly running the LISFLOOD model.

1.2.2 Fitness Function

The DEAP framework is configured to minimize a fitness function derived from the emulator-predicted 'KGE. For a given individual (parameter vector) $\mathbf{x}_{\text{opt}}$, the fitness is defined as:

$$\mathcal{F}(\mathbf{x}_{\text{opt}}) = -\widehat{\text{'KGE}}$$

where $\widehat{\text{'KGE}}$ is predicted by the emulator after input normalization and inverse transformation of the target variable. Predicted 'KGE values are constrained to the physically meaningful range $[-1, 1]$.

This formulation enables DEAP's minimization procedure to effectively maximize the predicted 'KGE.

1.2.3 Genetic Algorithm Configuration

The genetic algorithm hyperparameters are defined in a YAML configuration file to ensure reproducibility and flexibility. The main settings include population size, number of generations, crossover and mutation probabilities, and tournament selection size. Parallel evaluation of individuals is enabled to improve computational efficiency.

Parameter bounds are read from an external CSV file, ensuring that optimized values remain within physically and hydrologically plausible ranges.

1.2.4 Catchment-Specific Optimization

Optimization is performed independently for each catchment in the test dataset. For a given catchment, fixed input variables are extracted from the first available sample, while the optimization variables are initialized within their prescribed bounds. The DEAP toolbox is then constructed dynamically using the emulator-based fitness function and the catchment-specific configuration.

At the end of the evolutionary process, the parameter vector yielding the highest predicted 'KGE is retained as the optimal solution for that catchment.

1.2.5 Output and Post-Processing

The optimized parameter sets and corresponding emulator-predicted 'KGE values are stored in a consolidated output file. These parameters are subsequently used to run the LISFLOOD model explicitly, allowing direct comparison against baseline regionalization methods and assessment of the hydrological performance gains achieved through emulator-guided optimization.