SoftwareX 35 (2026) 102899

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

BOAR: Bayesian optimization for automated roughness calibration in two-dimensional hydrodynamic models

Luiz E.D. de Oliveira [a,*], Mário J. Franca [b], Nils P. Huber [c], Davide Vanzo [b]

[a] Institute for Water and Environment (IWU) – Karlsruhe Institute of Technology (KIT), Federal Waterways Engineering and Research Institute (BAW), and Department of Geosciences – Eberhard Karls University of Tübingen (UT), Karlsruhe, Germany
[b] Institute for Water and Environment (IWU) – Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany
[c] Federal Waterways Engineering and Research Institute (BAW), Karlsruhe, Germany

ARTICLE INFO

Keywords:
Bayesian optimization
Hydraulic roughness calibration
Shallow water solvers
BASEMENT
Surrogate model
Gaussian process
Model calibration

ABSTRACT

BOAR is an open-source Python framework for automated hydraulic roughness calibration in two-dimensional shallow-water numerical solvers. *BOAR* combines Bayesian optimization with conditional sampling to integrate expert knowledge, including feasible parameter ranges and inter-parameter constraints, thereby reducing the number of model evaluations. It standardizes calibration workflows, improves reproducibility, and supports flexible, user-defined loss functions. Applied to 25 laboratory vegetated compound-channel flow cases, it converged within a median of 9 model runs for a depth-error of 0.4%–1.5% tolerance. The current implementation interfaces with BASEMENT and can be extended to other river-modeling tools, providing a flexible and extensible solution for hydraulic modeling.

Metadata

Code metadata.

Nr.	Code metadata description	Metadata
C1	Current code version	v1.0.0
C2	Permanent link to code/repository used for this code version	https://github.com/baw-de/BOAR
C3	Permanent link to Reproducible Capsule	not applicable
C4	Legal Code License	MIT License
C5	Code versioning system used	git
C6	Software code languages, tools, and services used	Python 3
C7	Compilation requirements, operating environments & dependencies	h5py, numpy, openpyxl, optuna, pandas, PyYAML, scikit-learn, scipy, torch
C8	If available Link to developer documentation/manual	https://baw-de.github.io/BOAR/
C9	Support email for questions	luiz.oliveira@kit.edu

1. Motivation and significance

Two-dimensional depth-averaged shallow water (2D-SW) solvers are well-established tools in hydraulic engineering and environmental research modeling. Their applications range from flood simulation, to morphodynamic processes, to habitat and water quality assessment [1–5].

In 2D-SW models, hydraulic roughness is the primary closure relation used to represent energy losses associated with bed surface roughness, turbulence, and major roughness elements such as vegetation [6,7]. In practice, it is commonly specified through empirical coefficients related to bed composition or land use, such as the Strickler coefficient k_{st} (units of $L^{1/3}T^{-1}$) or the Manning coefficient $n = 1/k_{st}$. Because hydraulic roughness condenses several distinct physical processes into a

* Corresponding author.
Email addresses: luiz.oliveira@kit.edu (L.E.D. de Oliveira), mario.franca@kit.edu (M.J. Franca), nils.huber@baw.de (N.P. Huber), davide.vanzo@kit.edu (D. Vanzo).

https://doi.org/10.1016/j.softx.2026.102899
Received 26 May 2026; Received in revised form 15 July 2026; Accepted 20 July 2026
Available online 29 July 2026

single parameter, it is a major source of uncertainty [7]. Moreover, as the sole closure relation in basic 2D-SW models, it also absorbs unresolved physical effects, including secondary currents, three-dimensional processes, and vertical roughness variations [6–8], as well as numerical effects related to spatial discretization or turbulence-model coupling [1]. Consequently, miscalibration can lead to errors in computed flow fields [4] and flood hazard estimations [1,6,9], thereby undermining the reliability of real-time flood modeling [5]. Since hydraulic roughness often requires recalibration whenever model discretization or boundary conditions are modified, the process can be labor-intensive. Automation would therefore not only accelerate model development but also facilitate the repeated recalibrations necessary for robust real-time modeling and early-warning systems.

Several optimization strategies are commonly applied to 2D-SW simulations, ranging from time costly, labor-intensive manual calibration to Bayesian optimization (BO) [10]. Although both approaches can achieve comparable error levels, manual calibration typically requires on the order of hundreds of trial runs [11], whereas automated BO converges after a few dozen [5,11]. This represents a limitation, as depending on the spatial scale and resolution, a single 2D-SW simulation may require days to weeks of computational time [5,11,12]. By constructing a surrogate probabilistic model that learns the shape of the loss function (usually quantified by the root-mean-square error, RMSE), BO drastically reduces the number of expensive model evaluations required for convergence [5,10,13]. In practice, BO can attain the same calibration precision as manual tuning up to eight times faster [11].

Bayesian approaches to model calibration can be understood in two related but distinct ways: (i) Bayesian-optimization-based deterministic calibration, as employed in this study, and (ii) full Bayesian calibration as statistical inference. In the first setting, a Gaussian-process surrogate approximates the expensive calibration objective, and an acquisition function selects the next model evaluation; the output is the roughness parameter set that minimizes the chosen error metric, rather than a posterior distribution over roughness parameters [14,15]. In the second setting, the model parameters are treated as uncertain quantities, and prior distributions, likelihood functions, and observation-error assumptions are combined through Bayes' theorem to obtain posterior distributions and uncertainty estimates for the parameters [16,17]. The deterministic Bayesian optimization approach avoids the additional assumptions and computational effort required for full posterior inference over the physical parameters, making it well suited for computationally expensive shallow-water simulations and real-time calibration workflows [5,14,15].

Other implementations of hydraulic roughness calibration frameworks for 2D-SW models exist, but they either (i) do not employ Bayesian optimization [12], (ii) lack a mechanism for enforcing conditional constraints (e.g., "roughness of region A must be greater than that of region B") within the optimization process [5,18,19], or (iii) are primarily designed for 2D-SW models that are coupled with sediment-transport modules and therefore do not directly address calibration under fixed-bed conditions [11,20,21].

To address these limitations, we developed *BOAR*, a lightweight, open-source Python framework designed to streamline and standardize hydraulic roughness calibration in 2D-SW models. The software integrates a hydro- and morphodynamic model with two Bayesian optimization engines. *BOAR* enables the incorporation of hydraulic expertise—such as expected hydraulic roughness ranges and relationships between spatial hydraulic roughness classes—into the calibration process. This capability accelerates convergence and enhances usability, facilitating applications in both scientific research and engineering practice.

2. Software description

BOAR is designed to automate hydraulic roughness calibration procedures in hydrodynamic simulations using BASEMENT, a GPU-accelerated two-dimensional freeware software for river processes [2,22]. The package is written in Python ≥ 3.10 and relies on widely adopted scientific libraries, namely *h5py*, *numpy*, *openpyxl*, *optuna*, *pandas*, *PyYAML*, *scikit-learn*, *scipy* and *torch*. Although the Python code itself is platform-independent (noarch), it requires a working installation of BASEMENT (v4.2) which is presently available for x86/x86-64/arm64 architectures on Microsoft Windows and Linux. *BOAR* is organized into seven modular components and a helper batch script, as illustrated in Fig. 1 and described in Table 1. The code is available on GitHub, where it is developed and maintained: https://github.com/baw-de/BOAR.

2.1. Software functionalities

2.1.1. Governing equations

In a Cartesian frame of reference (x_i, z), with x_i $(i = 1, 2)$ being the horizontal plane and z being the vertical axis, the shallow-water equations can be cast in compact form as:

$$\partial_t h + \partial_{x_i}(hu_i) = 0,$$
$$\partial_t(hu_i) + \partial_{x_j}\left(hu_i u_j + \frac{1}{2}gh^2\delta_{ij}\right) = -gh\left(\partial_{x_i} z_b + S_{f_i}\right), \tag{1}$$

where h [L] is the flow depth, u [LT^{-1}] is the depth-averaged horizontal velocity vector, δ_{ij} [-] is the Kronecker delta, g [LT^{-2}] is the gravitational acceleration, z_b [L] is the bed elevation, and S_f [-] is the depth-averaged dimensionless bottom friction vector.

Under the hypothesis of turbulent flow, and hence under the assumption that the energy line slope is proportional to the square of the flow velocity, the bottom friction terms S_{f_i} can be written as:

$$S_{f_i} = \frac{u_i \|\mathbf{u}\|}{ghc_f^2} \tag{2}$$

where $\|\mathbf{u}\|$ is the norm of the velocity vector and c_f is the dimensionless friction (flow conductance) coefficient. Several formulae are available for c_f in literature (see e.g., Table 1 in [2]). In this application we adopt and calibrate the roughness term using the well-known Strickler closure relationship:

$$c_f = k_{st} h^{1/6}/\sqrt{g}, \tag{3}$$

where the Strickler coefficient (k_{st}) [$L^{1/3}T^{-1}$] is ultimately our parameter to be calibrated.

2.1.2. Bayesian optimization procedure

The `bo_optimizer.py` module provides Bayesian optimization (BO) procedure. The module goal is to minimize a loss function given by the user at `user_defined_configs/function_loss.py`, mathematically the goal is to iteratively minimize the loss given by:

$$\mathbf{x} = \underset{\mathbf{x}}{\operatorname{argmin}}\ f(\mathbf{x}) \tag{4}$$

where $\mathbf{x}$ is the decision vector that is used to calculate the user defined loss function $f(\mathbf{x})$ [23] (see Section 2.1.4).

The Bayesian optimization procedure is built using a Gaussian-process (GP) surrogate model with a Matérn covariance kernel, due to its flexibility in representing covariance structures [14]. The GP surrogate model provides a flexible probabilistic approximation of the loss function, allowing new model evaluations to be incorporated while preserving the Gaussian process formulation [16]. Given the observation data, the kernel matrix ($\mathbf{k}(\cdot,\cdot)$) is used to derive the posterior predictive distribution at a new point $\mathbf{x}$, characterized by the predictive mean $\mu(\mathbf{x})$ (Eq. 5) and predictive variance $\sigma^2(\mathbf{x})$ (Eq. 6). The predictive mean provides an estimate of the loss function, while the predictive variance quantifies the uncertainty associated with this estimate. In *BOAR*, the GP surrogate model was implemented with a small customizable constant noise term, α (default: 1×10^{-6}), added to the diagonal of the kernel matrix to improve numerical stability.

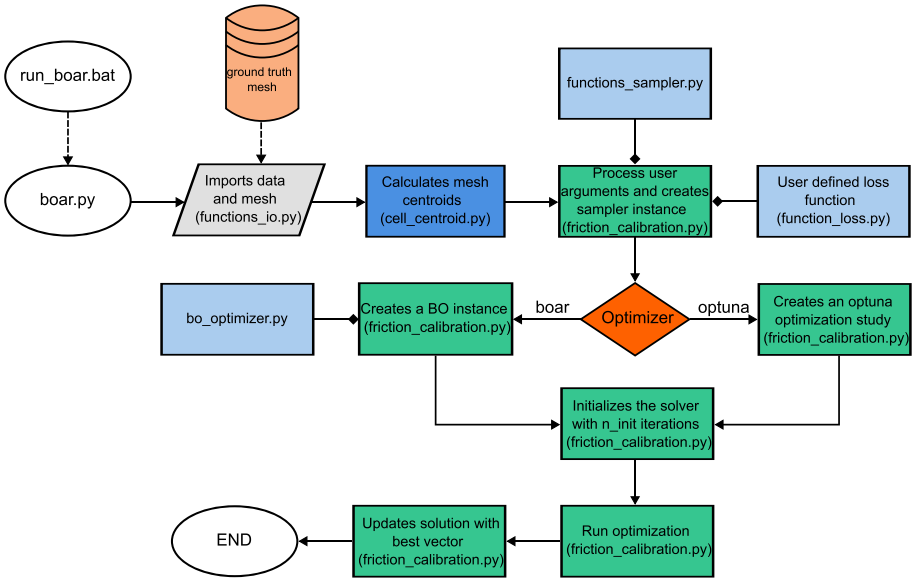

Fig. 1. Modular architecture of the *BOAR* software. The diagram illustrates the functional organization, showing the core tasks (symbols) and their respective files (colors). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Table 1
Description of *BOAR* modules. Modules highlighted in bold indicate those that require user modification (e.g., for setting optimization parameters or implementing custom loss functions).

Module name	Description
`boar.py`	Main entry point for BOAR (Bayesian Optimization for Automated Roughness calibration in two-dimensional hydrodynamic models).
`run_boar.bat`	Auxiliary batch script (Windows) that facilitates the code replication and parallel execution of multiple calibration instances.
`src/basement_tools.py`	Provides the interface between the calibration framework and the *BASEMENT* model.
`src/bo_optimizer.py`	Implements the Bayesian optimization procedure, exposed through the `BayesianOptimizer` class.
`src/cell_centroid.py`	Computes or imports the centroids of mesh cells used in the numerical domain.
`src/friction_calibration.py`	Core calibration module. It processes user inputs, assigns default values, defines the calibration workflow, and integrates the user-defined loss function before executing the Bayesian optimization routine.
`src/functions_io.py`	Handles input and output operations for all required files.
`src/functions_sampler.py`	Contains sampling strategies used to generate initial parameter sets and to support the Bayesian optimization process.
`user_defined_configs/constants.py`	Stores constant file paths, including those for the *BASEMENT* executable, simulation inputs, ground truth data, and mesh directories.
`user_defined_configs/function_loss.py`	Contains a user-defined function to evaluate the discrepancy between simulated results and ground truth data.
`user_defined_configs/user_options.yaml`	Defines user-specified configuration options controlling simulation execution, sampling strategies, optimization settings, and surrogate model parameters.

Given the training set of hydraulic roughness points ($\mathbf{X}_t$), and the loss function evaluations ($\mathbf{y_t} = f(\mathbf{X}_t)$), the GP predictive statistics for the candidate point $\mathbf{x}$ are:

$$\mu(\mathbf{x}) = \mathbf{k}(\mathbf{x})^\top K^{-1}\,\mathbf{y}_t \tag{5}$$

$$\sigma^2(\mathbf{x}) = k_{ss} - \mathbf{k}(\mathbf{x})^\top K^{-1}\,\mathbf{k}(\mathbf{x}) \tag{6}$$

where the superscript $(\cdot)^\top$ denotes transpose, $K = \left[k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})\right]_{i,j=1}^{n}$ is the kernel matrix on the training vector $\mathbf{X}_t$, $\mathbf{k}(\mathbf{x}) = \left[k(\mathbf{x}^{(i)}, \mathbf{x})\right]_{i=1}^{n}$ is the cross-kernel vector between training points and a test point $\mathbf{x}$, and $k_{ss} = k(\mathbf{x}, \mathbf{x})$ is the kernel evaluation at the test point.

Before fitting the GP, both the input parameters and objective values are standardized via z-score normalization using *scikit-learn*. After each new evaluation, the scaler is refitted to the expanded training set before the GP is retrained. The kernel matrix ($\mathbf{k}(\cdot,\cdot)$) is then computed on the normalized data.

Once the kernel regression is performed with the initial points, the new evaluation point is generated using the standard Expected Improvement (EI) acquisition function (Eq. 7). EI does not require parameter tuning and varies from zero at known points to positive values in between known points [23,24]. The next evaluation point is then calculated where $\mathbf{x_{n+1}} = \arg\max EI_n(\mathbf{x})$.

$$EI_n(\mathbf{x}) = \mathbb{E}_n\left[\left[f_n^* - f(\mathbf{x})\right]_+\right] \tag{7}$$

where $EI_n(\mathbf{x})$ is the Expected Improvement at iteration n for candidate $\mathbf{x}$, $\mathbb{E}_n[\cdot]$ is the expectation taken under the posterior distribution, given the previous loss function evaluations, $f_n^* = \min f(\mathbf{x}_i)$ is the lowest loss observed up to the iteration n. To account only for improvements, we consider the positive term of $f(\mathbf{x}) - f_n^*$ expressed by $[a]_+ := \max\{a, 0\}$.

The code iterates until one of the user-defined convergence criteria is met:

1. The maximum number of iterations (`max_tested_vectors`) is reached;
2. the maximum number of consecutive iterations without improvement (`max_no_improvement`) is reached, or
3. the error falls below a prescribed tolerance.

As an alternative to the custom-developed *BOAR* algorithm described above, the *Optuna* (≥ v4.8) Gaussian-process (GP) optimization engine can also be used, although it currently supports only stopping criterion

(1). When the *Optuna* GP engine is selected, the additional dependency *torch* must be installed. Benchmarks for both engines, utilizing the Ackley and Rosenbrock functions under constrained and unconstrained conditions, are available in the documentation.

2.1.3. Conditional sampling

The `functions_sampler.py` module provides the sampling algorithms implemented in the code. Three main sampling options were implemented using the Latin Hypercube [25], Sobol [26] and uniform random sampling strategies. These methods are employed both for generating the initial vectors during the initial GP surrogate model training, and for creating points at which the surrogate model will be evaluated.

All three sampling techniques were developed to generate samples based on user defined conditions at a given precision for a given range. This was built such that the user is capable of specifying hydraulic knowledge such as (i) the relationship between two hydraulic roughness regions ($k_{st,1} < k_{st,2}$), (ii) reducing the search range to feasible precision levels ($k_{st} = \text{round}(k_{st}, 2)$, for 2 decimal places) or (iii) specifying an expected range for each material associated with a hydraulic roughness region ($k_{st} \in [a, b]$). Collectively, these conditional and customizable constraints enable hydraulic modelers to reduce the optimization search space and, consequently, accelerate convergence. Furthermore, the sampling routines support user-defined random seeds to ensure reproducibility of results, which is typically desirable in scientific environments.

2.1.4. Loss function

The optimization loss function is customizable and user-defined. The user must provide a loss function that accepts two positional input dictionaries: (1) simulation data, containing the mesh cell centroids and all hydrodynamic results, and (2) ground-truth data containing the observed values at fixed XY coordinates. This design allows the use of any tabular format for the ground-truth data. Any resolved hydrodynamic variable (e.g., water depth, flow velocities, Reynolds stresses, etc.) can be used individually or combined to compose the loss value. Utility tools such as `utils/find_closest_cell` assist users in matching observed points to simulated cells.

The user-defined loss function must return a single floating-point value representing the global loss. Although not directly implemented, users may define variable-specific error thresholds within the loss function that, once satisfied, return a null loss value and terminate code execution. Users may also prioritize the convergence of specific variables by applying weighting factors during the composition of the global loss. Optionally, the function may return a second output containing a list of point-wise errors. These additional values do not affect the calibration process and are stored in CSV files solely for visualization purposes. Detailed instructions for implementing the loss function are provided in the "Loss function" section of the technical documentation and API reference.

2.1.5. Hydrodynamic model interface

BOAR interfaces with the freeware tool *BASEMENT* v4.2 [22], in which the two-dimensional shallow-water equations are solved. Hydrodynamic models in *BASEMENT* are configured via JSON files that specify boundary and runtime conditions, as well as ancillary files such as the mesh and discharge data. The coupling between *BOAR* and *BASEMENT* is performed by the helper module `basement_tools.py`. This module defines a class that is imported by `friction_calibration.py` and should not be executed directly. The main functionalities of `basement_tools.py` include importing *BASEMENT* configuration files, executing simulations, and extracting results. All external executables (e.g., `BMv4_simulation.exe` or `BMv4_simulation`) are invoked via Python's `subprocess` library, which guarantees cross-platform compatibility between Unix-like and Windows operating systems—the two platforms supported by *BASEMENT* v4.2. The `basement_tools.py` module can be generalized, or work as a template for interfacing *BOAR* with other hydro- and morphodynamic models that use text-based configurations.

The module `basement_tools.py` can be extended to interface with other numerical solvers. The high-level routines, *run_setup*, *run_simulation*, and *run_result_processing*, are deliberately generic and can be adapted to invoke the executables of any alternative solver. By contrast, the remaining internal functions are tightly coupled to the internal data structures of BASEMENT; therefore, a user who wishes to adopt a different solver must modify the *basement_tools.SimulationManager* class so that it provides an API compatible with the target software.

2.2. Setup and run BOAR

The code relies on three main user-defined directories: (i) `meshes/`, (ii) `ground_truth/`, and (iii) `support_files/`. In these folders, the user must provide the mesh files (i), the ground-truth CSV files for comparison (ii), and the discharge TXT files along with a lookup `discharge_dictionary.csv` containing the discharge name and its associated value in m^3/s (e.g., "Q_001", 1.00) (iii). Note that the discharge name must match those used within the `ground_truth/` directory and the `user_defined_configs/` files.

BOAR enables the calibration of multiple roughness coefficient configurations by defining specific target regions within the 2D-SW model (e.g., isolating a floodplain forest for calibration while maintaining the river and surrounding floodplain coefficients as constants). Additionally, the tool supports calibration across either a single discharge value or a range of discharge scenarios. These settings are managed through the `user_defined_configs/user_options.yaml` file and the corresponding file naming convention within the `ground_truth/` directory. Other critical parameters—including file paths, optimization settings (e.g., tolerance, engine, precision, and constraints), and custom loss functions—are defined in the files specified in Table 1. Detailed instructions for software execution are provided in the "Quickstart" section of the technical documentation.

Once *BOAR* is configured, it can be executed with:

```
python boar.py
```

or with the helper batch module (Windows exclusive):

```
.\run_boar.bat
```

Calibration progress and detailed execution logs are recorded in `boar.log` by default. Alternatively, a custom log file can be specified using the following command:

```
python boar.py --log-file [filename].
```

3. Illustrative examples

To illustrate the application of *BOAR*, we describe the calibration process of twenty-five 2D hydrodynamic simulations using the shallow water unsteady Reynolds-averaged Navier-Stokes (SW-URANS) equations. The simulations were calibrated using laboratory experiments as ground-truth data.

3.1. Laboratory experiments

Ground truth data were measured in laboratory experiments conducted in a compound-channel flume at the Federal Waterways Engineering and Research Institute (BAW). In a previous study [8], the hydrodynamics of four floodplain forest successional stages were examined under two flood-peak levels at a model scale of 1:30. In this paper we extend those experiments by adding additional flood-peak levels while retaining the same forest configurations, thereby increasing the range of submergence conditions and, consequently, the diversity of hydraulic roughness curves.

The vegetation was classified into adult and young trees to represent the different forest elements present in floodplain forests. Six forest scenarios were tested, four of which represent different successional forest ages (*Bare*, *Early*, *Mid*, and *Late*). In addition, two forest-maintenance cases were created by selectively removing vegetation from the *Mid* age,

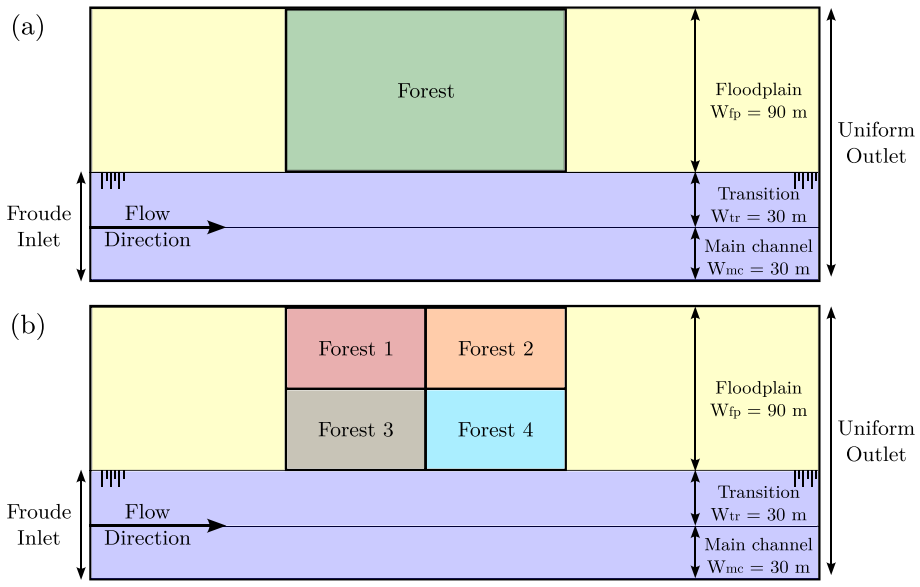

Fig. 2. Numerical simulation domain and boundary conditions. (a) For All *Bare* and *Forest* scenarios, and (b) Artificial dim = 6 scenario applied to the *Mid* vegetation. Each colour represents a hydraulic roughness region. The domain was scaled to the prototype scale (1:1) using Froude similarity. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Table 2
Vegetation and flow characteristics of the experimental cases. Colors indicate the vertical vegetation configuration: mixed adult and young trees (green), only young trees (blue), and only adult trees (Orange).

Case	Vegetation	Q [m^3/s]	U [m/s]	Fr [−]	H_{fp} [cm]	H_{mc} [cm]	H_r [−]
1	Bare	0.0358	0.24	0.28	0.0	10.0	–
2	Bare	0.0712	0.18	0.20	3.0	13.0	0.23
3	Bare	0.1196	0.22	0.21	6.0	16.0	0.38
4	Bare	0.1732	0.26	0.23	8.4	18.4	0.46
5	Bare	0.2493	0.31	0.25	11.0	21.0	0.52
6	Early	0.0712	0.18	0.20	3.0	13.0	0.23
7	Early	0.1196	0.22	0.21	6.0	16.0	0.38
8	Early	0.1732	0.26	0.23	8.4	18.4	0.46
9	Early	0.2493	0.31	0.25	11.0	21.0	0.52
10	Mid	0.0712	0.18	0.20	3.0	13.0	0.23
11	Mid	0.1196	0.22	0.21	6.0	16.0	0.38
12	Mid	0.1732	0.26	0.23	8.4	18.4	0.46
13	Mid	0.2493	0.31	0.25	11.0	21.0	0.52
14	Late	0.0712	0.18	0.20	3.0	13.0	0.23
15	Late	0.1196	0.22	0.21	6.0	16.0	0.38
16	Late	0.1732	0.26	0.23	8.4	18.4	0.46
17	Late	0.2493	0.31	0.25	11.0	21.0	0.52
18	Mid-Only-Young	0.0712	0.18	0.20	3.0	13.0	0.23
19	Mid-Only-Young	0.1196	0.22	0.21	6.0	16.0	0.38
20	Mid-Only-Young	0.1732	0.26	0.23	8.4	18.4	0.46
21	Mid-Only-Young	0.2493	0.31	0.25	11.0	21.0	0.52
22	Mid-Only-Adult	0.0712	0.18	0.20	3.0	13.0	0.23
23	Mid-Only-Adult	0.1196	0.22	0.21	6.0	16.0	0.38
24	Mid-Only-Adult	0.1732	0.26	0.23	8.4	18.4	0.46
25	Mid-Only-Adult	0.2493	0.31	0.25	11.0	21.0	0.52

resulting in configurations that retained only young trees (*Mid-Only-Young*) or only adult trees (*Mid-Only-Adult*). The forest was positioned on the floodplain as a 3-m-wide patch spanning the entire floodplain width and extending 7 m in length (Fig. 2).

Five steady-flow conditions were tested. These discharges generated a range of submergence levels for the flexible vegetation, yielding either single-layer or double-layer vegetated flow structures and, consequently, different hydraulic roughness conditions. Table 2 summarizes the main flow and vegetation characteristics of all 25 tested experiments: discharge Q [l/s], bulk velocity $U = Q/A$ (where A [m^2] is the flow area), Froude number $Fr = U/\sqrt{g\,A/B}$ (with B [m] the free-surface width), floodplain depth H_{fp} [m], main-channel depth H_{mc} [m], the relative depth $H_r = H_{fp}/H_{mc}$ [-] [27], and vegetation coverage.

3.2. Numerical model

The simulations were performed with the two-dimensional hydrodynamic model *BASEMENT* v4.2 [2], which solves the shallow-water, unsteady Reynolds-averaged Navier–Stokes equations (SW-URANS). Both the numerical domain (Fig. 2) and the experimental discharge and water-level data were scaled to prototype (1:1) size using Froude similarity, thereby placing the model in the same flow characteristic range in which the Gauckler–Manning–Strickler equations were designed. The

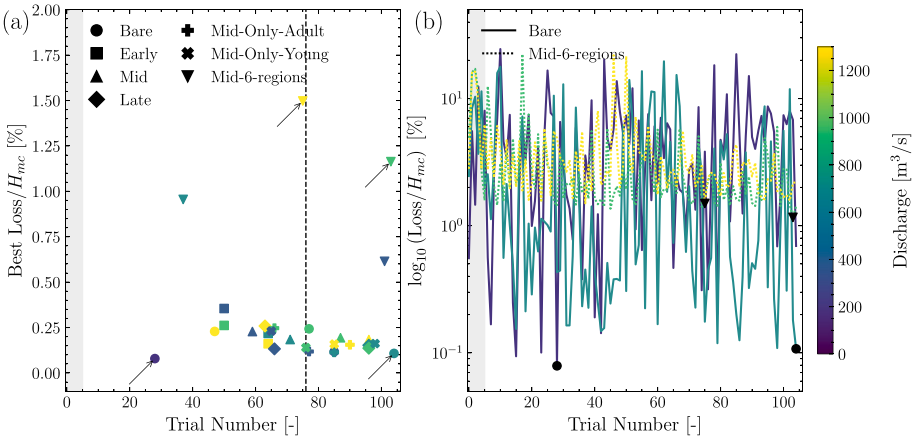

Fig. 3. Calibration-convergence analysis based on the normalized RMSE/H_{mc}: (a) best loss value (%) versus trial number; (b) loss-value trajectories for the *Bare* and *Early* cases at the trials highlighted by arrows in panel (a). The dashed line in panel (a) denotes the median number of trials required to obtain the optimal loss, and the black markers in panel (b) indicate the Corresponding optimal solution. The shaded light gray area represents the initial training phase. All simulations were performed on a Froude-scaled model at a 1:1 prototype scale. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

computational domain was equipped with two boundary conditions: (a) Froude inlet applied over the main channel and the transition regions, in which the prescribed discharge and bulk Froude number for each flow scenario were imposed, and (b) a uniform outlet covering the entire channel cross-section, in which a fixed longitudinal slope of 0.05% was prescribed. The simulations were run until the flow reached a steady state, at which point the numerical solution was deemed converged.

The hydraulic roughness was parametrized using the Strickler coefficient (k_{st}), spatially distributed over three regions: (1) the main channel and transition zone, (2) the floodplain, and (3) the forested area (Fig. 2a). *BOAR* was applied to calibrate the hydraulic roughness coefficients against measured water depths in two steps: (i) the main channel and floodplain were calibrated for the *Bare* scenario, defining a two-dimensional search space (dim = 2) and (ii) the main channel, floodplain and forest were calibrated (dim = 3). In order to showcase *BOAR*'s performance, we also split the forest into four divisions and calibrated the *Mid* vegetation with dim = 6, here called Mid-6-regions (Fig. 2b). The Strickler coefficient was independently calibrated for each vegetation and discharge condition, thus $k_{st}(Q, veg)$. In the *Bare* scenario, the first discharge corresponded to bankfull conditions; consequently, the vector dimension was reduced to dim = 1, as no flow occurred in the floodplain.

The loss function, ($RMSE(k_{st})$), was defined as the average root-mean-square error between the modeled and observed water-depth values at each measurement location:

$$\mathrm{RMSE}(k_{st}) = \frac{1}{A}\sum_{j=1}^{A}\mathrm{RMSE}_j,$$

$$\mathrm{RMSE}_j = \sqrt{\frac{1}{B}\sum_{i=1}^{B}\left(H_{\mathrm{num}} - H_{\mathrm{lab}}\right)^2}, \tag{8}$$

where H_{num} [m] is the simulated water depth at coordinates (x, y), H_{lab} [m] is the corresponding measured water depth, A is the number of files, B is the number of data points in each laboratory measurement file. In each case $A = 3$ files were recorded, each containing $B = 11$ measurement points. Because every file contains the same number of observations, the overall loss can be computed as the simple arithmetic mean of the individual RMSE values; no additional pooling or weighting is required, and the resulting aggregate loss remains unbiased.

BOAR was set to an initial population of $n_{initial} = 5$ training vectors, and a precision of 0.1 m$^{1/3}$s^{-1}. The optimization process was allowed to run up to `max_tested_vectors` = 100 (stop criterion 1, vide Section 2.1.2), with a `max_no_improvement` = 100 (stop criterion 2) and a tolerance of 0.001 m (stop criterion 3). Effectively, this implies that only criteria 1 and 3 could possibly be enforced. The model was calibrated using the BOAR engine. The vector sampler seed was set to 9, to enable results reproducibility. The search domain was limited between $k_{st} \in [20, 60]$ for the main channel and floodplain regions and $k_{st} \in [10, 35]$ for the forest regions. The hydraulic roughness vector was initialized with $k_{st} = 34.4$ m$^{1/3}$s^{-1} in all regions.

For the vegetated cases we introduced two inequality constraints on the Strickler coefficients: (i) $k_{st,forest} < k_{st,main\ channel}$ and (ii) $k_{st,forest} < k_{st,floodplain}$. Together, these constraints guarantee that the forest region is always rougher (i.e., has a lower Strickler coefficient) than the other hydraulic-roughness zones in the domain.

Fig. 3 shows the optimization convergence for all calibrated scenarios. The simulations agreed very well with the laboratory data, with $0.3\% < \mathrm{RMSE}(k_{st})/H_{mc} < 1.5\%$ across the full set of cases. All runs terminated when stopping criterion 1 was satisfied (`max_tested_vectors` = 100). The median convergence was typically reached within 71 iterations, corresponding to a total (5 training evaluations + optimization) of 76 hydrodynamic-model evaluations. Based on the global distribution of loss values versus trial numbers, we selected four representative calibration trajectories to illustrate: (1) the highest observed error, which is close to the median; (2) the slowest convergence for a vegetated scenario with vector dimension (dim = 6); (3) the slowest convergence; and (4) the fastest convergence for the *Bare* scenario with (dim = 2) (Fig. 3(b)). These trajectories show how the code explores the search space avoiding becoming trapped in local minima, as observed for the *Bare* case under bankfull conditions (Fig. 3(b)).

As an additional showcase, the calibration scenarios were reanalyzed using a modified tolerance of $0.015H_{\mathrm{mc}}$, corresponding to a normalized threshold of $\mathrm{RMSE}/H_{\mathrm{mc}} < 1.5\%$ (stop criterion 3). Fig. 4 shows the number of trials required to reach this loss threshold. Under this setting, *BOAR* converged after a median of four optimization iterations, corresponding to a total of nine hydrodynamic-model evaluations, including the five initial training evaluations. The observed loss ($\mathrm{RMSE}(k_{st})/H_{mc}$) ranged between 0.4 and 1.5%.

Overall, convergence speed was linked to the smoothness of the loss function: when gradients between parameter vectors were smaller, the optimization engine required more trials to explore the search space, thereby slowing convergence.

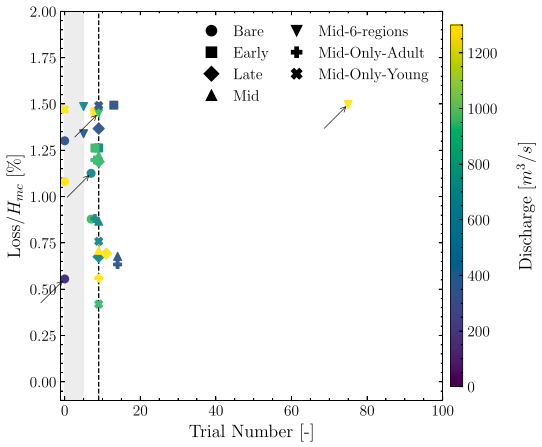

Fig. 4. Calibration-convergence analysis based on the normalized RMSE/H_{mc} (Loss) versus trial number, with an error threshold of Loss/$H_{mc} \leq 1.5\%$. The shaded light gray area represents the initial training phase. All simulations were performed on a Froude-scaled model at a 1:1 prototype scale.

4. Conclusions and impact

In this study, we present *BOAR*, a lightweight, open-source Python 3 framework designed to streamline the calibration of hydraulic roughness in two-dimensional depth-averaged hydrodynamic simulations. *BOAR* significantly reduces the temporal and computational costs of parameter optimization by pairing a Bayesian optimization engine with a flexible interface for managing ground-truth data, simulation configurations (JSON and HDF5), and user settings (YAML).

The impact of this tool extends beyond simple automation; by establishing a unified and standardized procedure for calibration, *BOAR* enables fully reproducible results in both academic research and practical engineering contexts. For instance, *BOAR* greatly facilitates mesh-independence analyses, which require the recalibration of the same model across varying mesh configurations, since hydraulic roughness is sensitive to domain discretization. Such sensitivity directly affects critical outputs, including flood-peak magnitude and propagation [1,9], which subsequently influence the reliability of flood hazard estimations and early-warning systems.

Although the current version of *BOAR* does not implement a full Bayesian inference framework, its modular architecture provides a foundation for extending the calibration workflow to additional river-process parameters and model outputs, such as sediment rating curves, water temperature, or habitat metrics. With a design focused on maintainability and extensibility, *BOAR* promotes open science and encourages the community to extend its functionality to other simulation codes. Future development should also focus on incorporating support for unsteady-flow ground-truth data, further broadening the tool's applicability.

CRediT authorship contribution statement

Luiz E.D. de Oliveira: Writing – review & editing, Writing – original draft, Validation, Software, Methodology, Data curation. **Mário J. Franca:** Writing – review & editing, Writing – original draft, Supervision, Funding acquisition, Conceptualization. **Nils P. Huber:** Writing – review & editing, Writing – original draft, Supervision, Funding acquisition. **Davide Vanzo:** Writing – review & editing, Writing – original draft, Supervision, Methodology, Conceptualization.

Declaration of competing interest

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

Acknowledgements

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors. We acknowledge Leonie Herlan for her assistance during the conduction of the laboratory experiments. We also thank Dr. Martin Utz for repository curation.

References

[1] Caviedes-Voullième D, García-Navarro P, Murillo J. Influence of mesh structure on 2D full shallow water equations and SCS curve number simulation of rainfall/runoff events. J Hydrol 2012;448-449:39–59. https://doi.org/10.1016/j.jhydrol.2012.04.006. https://www.sciencedirect.com/science/article/pii/S0022169412002697.

[2] Vanzo D, Peter S, Vonwiller L, Bürgler M, Weberndorfer M, Siviglia A, Conde D, Vetsch DF. basement v3: a modular freeware for river process modelling over multiple computational backends. Environ Model Softw 2021;143:105102. https://doi.org/10.1016/j.envsoft.2021.105102. https://www.sciencedirect.com/science/article/pii/S1364815221001456.

[3] Caponi F, Vetsch DF, Vanzo D. BASEveg: a Python package to model riparian vegetation dynamics coupled with river morphodynamics. Softwarex 2023;22:101361. https://doi.org/10.1016/j.softx.2023.101361. https://linkinghub.elsevier.com/retrieve/pii/S2352711023000572.

[4] Altmann M, Vanzo D, Valero D, Schalko I. A simple approach to simulate logjams in two-dimensional hydrodynamic models. J Hydraul Eng 2024;150(4):06024002. https://doi.org/10.1061/JHEND8.HYENG-13713

[5] Young A, Albertson JD, Moretti G, Orlandini S. Real-time flood inundation modeling with flow resistance parameter learning. Water Resour Res 2025;61(1):e2024WR038424, _eprint: https://doi.org/10.1029/2024WR038424. https://onlinelibrary.wiley.com/doi/pdf/10.1029/2024WR038424.

[6] Chanson H. The hydraulics of open channel flow. 2nd ed. Oxford: Butterworth-Heinemann; 2004.

[7] Armanini A. Principles of river hydraulics. Cham: Springer International Publishing; 2018. https://doi.org/10.1007/978-3-319-68101-6. http://link.springer.com/10.1007/978-3-319-68101-6.

[8] de Oliveira LED, Janzen JG, Folke F, Wittmann F, Huber NP, Franca MJ, Gualtieri C. Hydrodynamics of four moments in the life of a floodplain forest in compound channels. Water Resour Res 2026;62(4):e2024WR038968, _eprint: https://doi.org/10.1029/2024WR038968. https://agupubs.onlinelibrary.wiley.com/doi/pdf/10.1029/2024WR038968.

[9] Prior EM, Michaelson N, Czuba JA, Pingel TJ, Thomas VA, Hession WC. LiDAR DEM and computational mesh grid resolutions modify roughness in 2D hydrodynamic models. Water Resour Res 2024;60(7):e2024WR037165. https://doi.org/10.1029/2024WR037165

[10] Yang L, Shami A. On hyperparameter optimization of machine learning algorithms: theory and practice. Neurocomputing 2020;415:295–316. https://doi.org/10.1016/j.neucom.2020.07.061. https://www.sciencedirect.com/science/article/pii/S0925231220311693.

[11] Beckers F, Heredia A, Noack M, Nowak W, Wieprecht S, Oladyshkin S. Bayesian calibration and validation of a large-scale and time-demanding sediment transport model. Water Resour Res 2020;56(7):e2019WR026966, _eprint: https://doi.org/10.1029/2019WR026966. https://agupubs.onlinelibrary.wiley.com/doi/pdf/10.1029/2019WR026966.

[12] Ming X, Liang Q, Xia X, Li D, Fowler HJ. Real-time flood forecasting based on a high-performance 2-D hydrodynamic model and numerical weather predictions. Water Resour Res 2020;56(7):e2019WR025583, _eprint: https://doi.org/10.1029/2019WR025583. https://agupubs.onlinelibrary.wiley.com/doi/pdf/10.1029/2019WR025583.

[13] Kohlhaas R, Morales Oreamuno MF, Lacheim A. BayesValidRox 2.0.0. JoDaKISS - J Data- Knowl-integr Simul Sci Jul 2025. https://doi.org/10.46298/jodakiss.15337. https://jodakiss.episciences.org/15337.

[14] Frazier PI. A tutorial on Bayesian optimization, Issue: arXiv:1807.02811. Jul 2018.

[15] Hutter F, Kotthoff L, Vanschoren J, Eds. Automated machine learning: methods, systems, challenges, The Springer Series on Challenges in Machine Learning. Cham: Springer International Publishing; 2019. https://doi.org/10.1007/978-3-030-05318-5. http://link.springer.com/10.1007/978-3-030-05318-5.

[16] Rasmussen CE, Williams CKI. Gaussian processes for machine learning. The MIT Press; 2005. https://doi.org/10.7551/mitpress/3206.001.0001. https://direct.mit.edu/books/monograph/2320/Gaussian-Processes-for-Machine-Learning.

[17] Heard N. An introduction to Bayesian inference, methods and computation. Cham: Springer International Publishing; 2021. https://doi.org/10.1007/978-3-030-82808-0

[18] Balbi M, Lallemant DCB. Bayesian calibration of a flood simulator using binary flood extent observations. Hydrol Earth Syst Sci 2023;27(5):1089–108. https://doi.org/10.5194/hess-27-1089-2023. https://hess.copernicus.org/articles/27/1089/2023/.

[19] Tanim AH, Smith-Lewis C, Downey ARJ, Imran J, Goharian E. Bayes_Opt-SWMM: a Gaussian process-based Bayesian optimization tool for real-time flood modeling with SWMM. Environ Model Softw 2024;179:106122. https://doi.org/10.1016/j.envsoft.2024.106122. https://www.sciencedirect.com/science/article/pii/S136481522400183X.

[20] Oladyshkin S, Mohammadi F, Kroeker I, Nowak W. Bayesian3 active learning for the Gaussian process emulator using information theory. Entropy 2020;22(8):890. https://doi.org/10.3390/e22080890. https://www.mdpi.com/1099-4300/22/8/890.

[21] Mouris K, Acuna Espinoza E, Schwindt S, Mohammadi F, Haun S, Wieprecht S, Oladyshkin S. Stability criteria for Bayesian calibration of reservoir sedimentation models. Model Earth Syst Environ 2023;9(3):3643–61. https://doi.org/10.1007/s40808-023-01712-7

[22] Vetsch DF, Frei S, Halso MC, Schierjott JC, Bürgler M, Vanzo D. Basement v4—a multipurpose modelling environment for simulation of flood hazards and river morphodynamics across scales. In: Gourbesville P, Caignaert G, editors. Advances in Hydroinformatics—SimHydro 2023, vol. 1. Singapore: Springer Nature; 2024. p. 125–38. https://doi.org/10.1007/978-981-97-4072-7_8

[23] Snoek J, Larochelle H, Adams R. Practical Bayesian optimization of machine learning algorithms. In: Pereira F, Burges CJ, Bottou L, Weinberger K, editors. Advances in neural information processing systems, vol. 25. Curran Associates, Inc.; 2012. https://proceedings.neurips.cc/paper_files/paper/2012/file/05311655a15b75fab86956663e1819cd-Paper.pdf.

[24] Jones DR, Schonlau M, Welch WJ. Efficient global optimization of expensive black-box functions. J Glob Optim 1998;13(4):455–92. https://doi.org/10.1023/A:1008306431147

[25] McKay MD, Beckman RJ, Conover WJ. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. Technometrics 1979;21(2):239–45. https://doi.org/10.2307/1268522. https://www.jstor.org/stable/1268522.

[26] Sobol' IM. On the distribution of points in a cube and the approximate evaluation of integrals. USSR Comput Math Math Phys 1967;7(4):86–112. https://doi.org/10.1016/0041-5553(67)90144-9. https://www.sciencedirect.com/science/article/pii/0041555367901449.

[27] Nezu I, Onitsuka K, Iketani K. Coherent horizontal vortices in compound open channel flows. In: Hydraulic modeling: proceedings of the international conference on water, environment, ecology, socio-economics and health engineering. Seoul, Korea: Singh V, Seo I, Sonu J; 1999. p. 17–32.