



ISSM-ML: a modular framework for integrating machine learning models into the Ice-sheet and Sea-level System Model (ISSM v2026.4)

Yinmin Liu¹, Mathieu Morlighem¹, and Gong Cheng^{2, 3}

¹Department of Earth and Planetary Sciences, Dartmouth College, Hanover, NH 03755, USA

²Center for Spatial Information Science and Sustainable Development Applications, Tongji University, Shanghai, China

³College of Surveying and Geo-informatics, Tongji University, Shanghai, China

Correspondence: Gong Cheng (chenggong@tongji.edu.cn)

Abstract. Machine learning (ML) models are increasingly used in glaciology to replace computationally expensive model components or emulate specific processes. These approaches, however, are commonly developed as standalone tools or within application specific software and cannot be readily deployed within established ice-sheet modeling frameworks. We present ISSM-ML, a modular framework for integrating externally trained, Python-based ML models into the Ice-sheet and Sea-level System Model (ISSM). Model development and training remain within the Python ecosystem, while ISSM invokes selected ML modules at runtime and retains control of model initialization, numerical solvers, physical analyses, time stepping, and other procedures in its original workflow. We demonstrate the framework using a multilayer perceptron to replace the nonlinear velocity-dependent term in the Weertman basal-friction law. Two training strategies are evaluated, one using conventional scaling and the other using distribution-aware preprocessing. When the scaling-only model is coupled to ISSM, the stress balance solver fails to converge. The distribution-aware model, in contrast, allows ISSM to converge reliably throughout the transient simulation, with a good agreement with the analytical-friction reference. These results demonstrate that offline prediction accuracy alone is not sufficient for evaluating an ML component intended for numerical-model coupling. Their behavior must also be assessed within the simulation environment in which they operate, including convergence, stability, and feedback with evolving model states.

1 Introduction

Ice-sheet numerical models are essential tools for understanding glacier and ice-sheet evolution and for projecting their contributions sea-level rise in a changing climate (Goelzer et al., 2020; Seroussi et al., 2020). Advances in numerical methods, computational resources, and the incorporation of observations have enabled increasingly detailed simulations at regional to continental scales (Larour et al., 2012; Fischler et al., 2022; Cheng et al., 2022). Nevertheless, the predictive capability of ice-sheet models remains limited by uncertainties in the representation of key physical processes. Basal sliding, calving, ice-ocean interactions, and surface mass balance for example involve processes that remain poorly understood. They are commonly represented through simplified or empirically calibrated parameterizations, contributing substantial uncertainty to projections of



future ice-sheet evolution (Pattyn and Morlighem, 2020; Seroussi et al., 2020; Goelzer et al., 2020). Improving the representation and constraint of these processes remains a central challenge in ice-sheet modeling.

25 Machine learning (ML) provides a flexible and increasingly promising set of tools for learning complex relationships from observational and numerical simulation data. ML models can emulate computationally expensive calculations, construct data-driven parameterizations, extract information from large datasets, and incorporate physical constraints into predictive models (Willard et al., 2022; Karniadakis et al., 2021). These capabilities complement established approaches such as data assimilation, which uses observations to estimate poorly constrained model parameters (MacAyeal, 1993; Morlighem et al., 2010; Goldberg et al., 2011; Choi et al., 2023). In glaciology, ML has been applied to ice-flow emulation (Jouvet and Cordonnier, 2023; Jouvet, 2023), regional climate-model emulation (Schlager et al., 2026; Antwerpen et al., 2026) and surface-mass-balance downscaling (Bochow et al., 2026; Van Der Meer et al., 2023), calving-parameter calibration (Koo et al., 2025), and automated interpretation of remote-sensing imagery (Baumhoer et al., 2023; Cheng et al., 2021; Lai et al., 2020; Baumhoer et al., 2019). These studies demonstrate the potential of ML to augment numerical ice-sheet models by accelerating selected calculations, representing relationships that are difficult to parameterize explicitly, and making effective use of growing observational and simulation datasets.

Despite these advances, existing ML approaches for ice-sheet applications are commonly developed as standalone tools or within dedicated, application-specific software (Cheng et al., 2025; Jouvet and Cordonnier, 2023; Jouvet, 2023). A general mechanism for incorporating externally trained ML models into established ice-sheet modeling frameworks remains less developed. Established numerical ice-sheet models are indispensable because they provide mature and tested infrastructures for solving known physical equations and coordinating many interacting components, including ice flow solvers, ice front and grounding line dynamics, and surface and basal processes. Current ML models generally represent only specific processes or calculations rather than this complete modeling infrastructure. A practical pathway is therefore a hybrid modeling approach to incorporate ML models directly into established numerical models, where they can augment or replace selected calculation, while the numerical model retains control of the overall simulation and interactions among its components.

To demonstrate this approach, we introduce ISSM-ML, a framework for coupling pre-trained machine-learning models with the Ice-sheet and Sea-level System Model (ISSM). This framework provides reusable infrastructure for augmenting ISSM with data-driven modeling capabilities. Trained models can selectively replace ISSM calculations without modifying the overall simulation workflow. In Section 2, we describe the general structure and features of the framework. In Section 3, we illustrate its implementation in detail using a friction emulator as an example. In Section 4, we present the training results of friction emulator and conduct a transient simulation with the coupled model. In Section 5, we reflect on the framework design and the use of simulation-derived training data, and discuss the challenges and opportunities associated with coupling machine-learning models with numerical ice-sheet models.

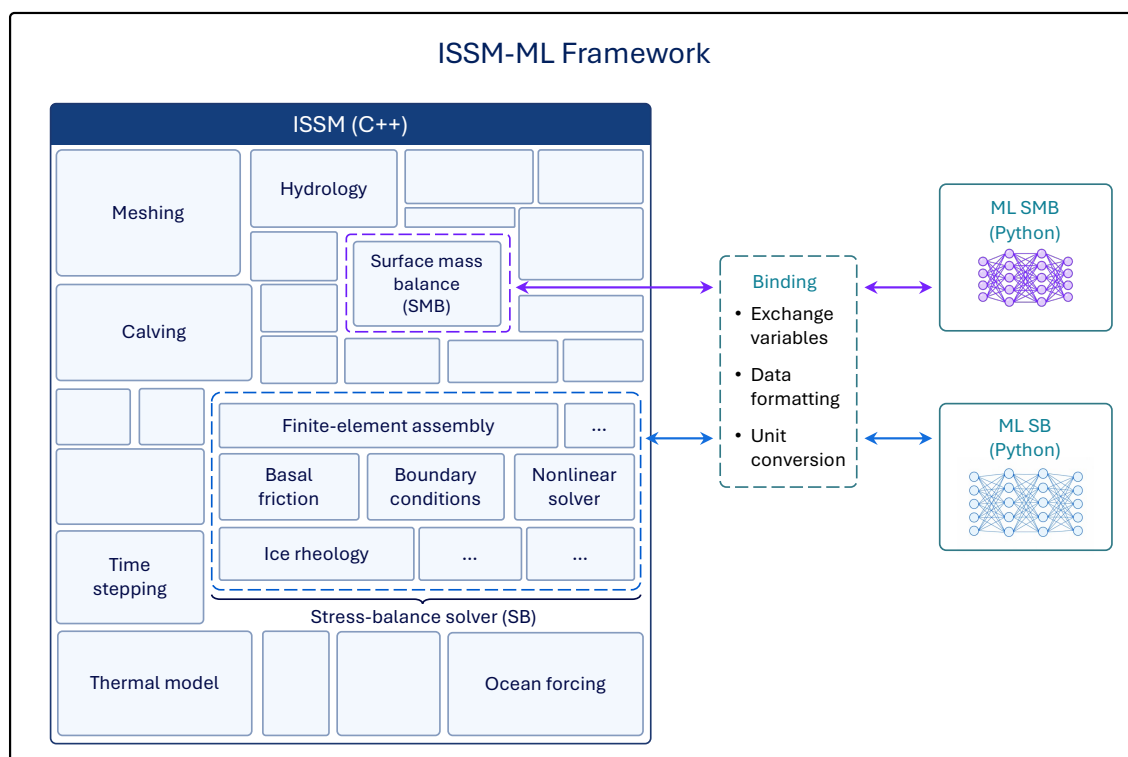


Figure 1. ISSM–ML framework for integrating externally trained machine-learning models into ISSM. The framework allows ML model to replace calculations at different levels of ISSM hierarchy, from individual functions to standalone modules, such as surface mass balance (purple dashed box), or multi-module components, such as the stress-balance solver (blue dashed box), through a common binding interface.

2 ISSM-ML framework

- ISSM’s core calculations are implemented in C++, while MATLAB and Python interfaces are provided for pre- and post-processing. In the ISSM-ML framework, trained ML models replace specific components in ISSM. As illustrated in Fig. 1, this substitution can occur at different levels of the model workflow. An ML model may represent a local calculation, such as a constitutive relation, or a larger collection of calculations, such as the solution of a differential equation. All unaffected components continue to use their standard ISSM implementations.
- The ISSM-ML binding layer provides a general, reusable coupling layer between ISSM and externally trained Python models. Emulators for different physical processes share the same underlying infrastructure and follow the same coupling pattern, and only a limited amount of process-specific handling is required to define the expected inputs, outputs, and conduct units conversion. At the beginning of a simulation, the selected model and its parameters are loaded into memory and then reused throughout the simulation, to avoid the computational overhead of repeated initializations. When the ML model is evaluated, the relevant ISSM variables are prepared and passed to the model, and the resulting predictions are then returned to the appropriate ISSM component. Because inference is performed using the model’s underlying Python ML library, an emulator



can also use CPU/GPU acceleration provided by libraries such as PyTorch or JAX. Users can configure an available emulator with a compatible trained model, while ML developers can retrain or replace such models within the Python ecosystem without the need of interact with ISSM core computing codes.

70 3 Friction law emulator

To illustrate the capability of the ISSM-ML framework, we develop and integrate an ML model for the nonlinear term in the friction law. This section describes the target relationship, the model architecture and training data, the user configuration, and the C++/Python binding used to initialize and evaluate the trained model during an ISSM simulation.

3.1 Friction law

75 We use the Weertman friction law (Weertman, 1957; Fowler, 1981), which assumes a power-law relationship between basal shear stress and basal velocity:

$$\tau_b = -C^2 |\mathbf{u}_b|^{\frac{1}{m}-1} \mathbf{u}_b, \quad (1)$$

where τ_b is the basal shear stress, $\mathbf{u}_b$ is the basal sliding velocity, C^2 is the friction coefficient representing bed properties, and $m = 3$ is the sliding exponent. Within ISSM, all friction laws are reformulated so that all the terms that multiply the ice sliding velocity are lumped into a single term, α^2 , such that $\tau_b = -\alpha^2 \mathbf{u}_b$. For the Weertman law, this pre-factor satisfies

$$\alpha^2 = C^2 |\mathbf{u}_b|^{\frac{1}{m}-1}. \quad (2)$$

Our friction emulator learns the mapping of Eq. (2), from the local friction parameter C^2 and basal-velocity magnitude $|\mathbf{u}_b|$ to the corresponding basal-drag pre-factor α^2 . Following the ISSM convention, the notations α^2 and C^2 are used to indicate that these two parameters are nonnegative.

85 3.2 Emulator preparation

We use a multilayer perceptron (MLP) to approximate this mapping. MLPs are feedforward neural networks composed of an input layer, one or more hidden layers, and an output layer, with nonlinear activation functions applied between successive layers (Rosenblatt, 1958; Rumelhart et al., 1986). Their ability to approximate continuous nonlinear functions makes them suitable for representing the target relation in the Eq. (2) (Hornik et al., 1989; Cybenko, 1989). Figure 2 shows the MLP architecture used for the friction emulator. We use two fully connected hidden layers, each with 32 neurons and ReLU activation functions (Nair and Hinton, 2010). We construct our training data from an ISSM simulation of Helheim Glacier using the Weertman friction law. The model setup and mesh are adapted from Cheng et al. (2022) and Koo et al. (2025). In total, we use 217,128 samples over the full transient simulation, consisting of the input variables C^2 and $|\mathbf{u}_b|$, together with the corresponding Weertman basal-drag pre-factor α^2 , at each nodal point.

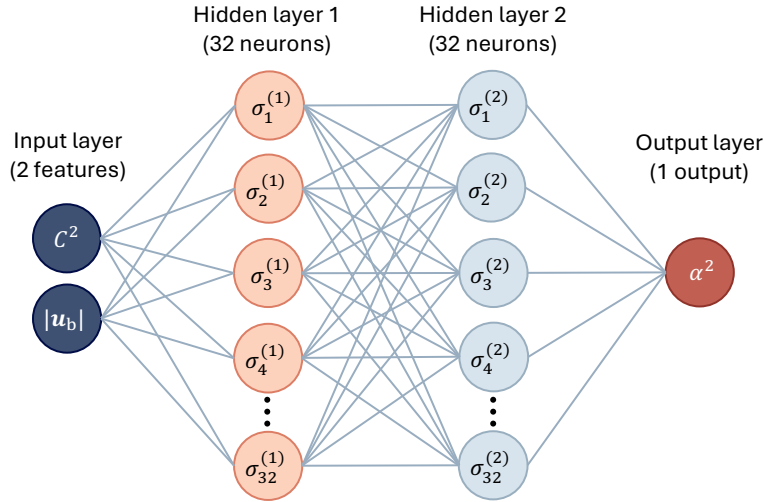


Figure 2. Architecture of the multilayer perceptron used to emulate the Weertman basal-drag pre-factor. The inputs are the squared friction coefficient, C^2 , and basal-velocity magnitude, $|u_b|$. Two fully connected hidden layers, each containing 32 neurons with ReLU activation, produce the predicted basal-drag pre-factor, α^2 . The symbols $\sigma_j^{(1)}$ and $\sigma_j^{(2)}$ denote the activations of neuron j in the first and second hidden layers, respectively.

95 We apply a preprocessing workflow consisting of data cleaning, variable transformation, weighted sampling, and normalization. First, samples with $|u_b|$ less than a threshold are removed. A square-root transformation is then applied to reduce distributional skewness. The remaining data are randomly divided into training, validation, and test sets using a 70% : 15% : 15% split. Sampling is then applied only to the training set. The two-dimensional parameter space defined by $\sqrt{C^2}$ and $\sqrt{|u_b|}$ is divided into a 30×30 grid using uniformly spaced bin edges between the minimum and maximum training values along
 100 each axis. If a joint bin contains n samples, each sample in that bin is assigned a weight proportional to $1/n$. Then, weighted random sampling is performed, increasing the probability of selecting samples from sparsely populated regions. A total of 30,000 samples are drawn from the 147,188 training samples; the validation and test sets retain their original distributions.

After training, the friction emulator is stored as an external model package containing a checkpoint file and a Python model file. The checkpoint stores the trained network parameters and architecture information. The Python file defines the MLP and
 105 provides the functions used to initialize the model and perform inference. The Listing 1 shows the major components of this model file. The `init_model` function loads the checkpoint, constructs the network using the stored architecture dimensions, and selects an available computational device. As shown in the line 13 of Listing 1, the Python file initializes the emulator on GPUs when CUDA devices are available in the computing environment. The `predict_alpha2_np` function converts the inputs to tensors, applies the normalization used during training, evaluates the initialized model, transforms the output back to
 110 its original scale, and returns the prediction as a NumPy array. The checkpoint file and the model file are supplied to ISSM through the configuration interface described in the following section.

```
1: class FrictionMLP(torch.nn.Module):
```



```

115 2:     def __init__(self, in_dim, h1_dim, h2_dim, out_dim):
3:         super().__init__()
4:         self.net = Sequential(
5:             Linear(in_dim, h1_dim), ReLU(),
6:             Linear(h1_dim, h2_dim), ReLU(),
7:             Linear(h2_dim, out_dim) )
120 8:
9:     def init_model(weights_path):
10:         state_dict, in_dim, h1, h2, out_dim = load(<checkpoint directory>)
11:         model = FrictionMLP(in_dim, h1_dim, h2_dim, out_dim)
12:         model.load_state_dict(state_dict)
125 13:         model.to("cuda" if cuda.is_available() else "cpu")
14:         model.eval()
15:
16:     def predict_alpha2_np(features_np):
17:         features_t = convert features_np to tensor on device
18:         features_norm = (features_t - x_mean) / x_std
130 19:         prediction_norm = model(features_norm)
20:         prediction = prediction_norm * y_std + y_mean
21:         prediction.cpu()
135 22:         return prediction.numpy()
    
```

Listing 1. Pseudocode for the friction emulator Python model file.

3.3 Emulator configuration

Users activate the friction emulator through the standard ISSM model configuration interface (either in MATLAB or Python). The emulator is selected by assigning the `frictionemulator` class to the `md.friction` field. The user then specifies the directory containing the model package, the checkpoint file name, the Python model file name, and the spatial friction field required by the emulator. Listing 2 shows this configuration. The field `md.friction.module_dir` specifies the directory containing the emulator files. The fields `md.friction.pt_name` and `md.friction.py_name` identify the checkpoint and Python files within that directory. The field `md.friction.C` supplies the spatially varying ISSM friction coefficient used to construct the input feature C^2 .

```

145 1: md.friction = frictionemulator();
2: md.friction.module_dir = <directory to the friction emulator folder>;
3: md.friction.pt_name = <checkpoint file name>;
4: md.friction.py_name = <ML model file name>;
150 5: md.friction.C2 = <friction coefficients>;
    
```

Listing 2. Configure the emulator on ISSM's MATLAB interface.



3.4 Emulator binding

The binding layer is implemented within the C++ level source code of ISSM. It consists of two parts: an emulator class and an emulator wrapper. The constructor of the emulator class receives the configuration values supplied by the user, stores the emulator file information, initializes the embedded Python environment, imports the specified Python module, and calls the module's
155 `init_model` Python function (see Section 3.2). The major components of the emulator class, named `EmulatorParam`, are listed in Listing 3. The member `enum_type` identifies the emulator type, which in this example is the friction emulator. The member `guard` manages the embedded Python interpreter. The imported Python module is stored in `mod`, allowing the initialized model to be reused during subsequent friction evaluations.

```
160 1: class EmulatorParam: public Param{  
2:     public:  
3:         int          enum_type; // identifier for the emulator type  
4:         char*        module_dir; // directory containing emulator files  
5:         char*        pt_name; // checkpoint file name  
165 6:         char*        py_name; // Python model file name  
7:         py::scoped_interpreter* guard; // embedded Python interpreter manage  
8:         py::module_    mod; // imported Python module  
9:  
10:     EmulatorParam(enum_type, module_dir, pt_name, py_name) {  
170 11:         this->guard = new py::scoped_interpreter();  
12:         this->enum_type=enum_type;  
13:         this->mod = py::module_::import(module_dir+py_name);  
14:         this->mod.attr("init_model")(pt_name); } // Constructor  
15:     ~EmulatorParam() // Destructor  
175 16:     ...
```

Listing 3. Major members of the emulator C++ class.

The emulator wrapper is used to evaluate the friction emulator during the repeated assembly of the stiffness matrix within the numerical solver. As described in Listing 4, the C++ friction routine retrieves the friction coefficient C^2 and basal-velocity magnitude $|\mathbf{u}_b|$ and constructs the feature array $[C^2, |\mathbf{u}_b|]$, following the same ordering as the input layer of the neural net-
180 work shown in Fig. 2. The feature array is converted from C++ object to a NumPy array and passed to the Python function `predict_alpha2_np`. The resulting prediction is then converted back to an ISSM scalar and used as the pre-factor α^2 in basal drag, thereby replacing the conventional evaluation of α^2 in the basal-friction calculation. In the present implementation, emulator inference is performed independently on each Message Passing Interface (MPI) process. All cross-language function calls and data-type conversions between C++ and Python are handled using the `pybind11` package (Jakob et al., 2017).

```
185 1: emulator_param = FindParamObject(FrictionEmulatorEnum); // Specify the emulator type  
2: emulator = (EmulatorParam*)emulator_param; // Create an emulator object  
3:  
4: GetAlpha2Emulator(IssmDouble* alpha2){  
190 5:     C2 = GetFrictionCoefficient();
```



```

6:   vmag = GetVelocityMagnitude();
7:   features_cpp = [C2, vmag] // Build C++ feature array
8:   features_np = cpp_to_numpy(features_cpp) // Convert to NumPy array
9:   prediction_np = emulator->mod.predict_alpha2_np(features_np) // Evaluate Python model
10:  *alpha2 = numpy_to_cpp(prediction_np) // Assign predicted alpha2
11: }
```

Listing 4. Pseudocode for the C++ wrapper that evaluates the friction emulator.

4 Results

This section evaluates the friction-law emulator from three complementary perspectives. First, we characterize the simulation-generated training data and examine how data preprocessing changes the coverage of the emulator input space. Second, we compare offline emulator performance between a scaling-only baseline and a comprehensive preprocessing workflow. Here, offline process refers to the model optimization and evaluation performed outside the ISSM simulation environment. Finally, we evaluate the emulator within a transient ISSM simulation to assess its behavior under coupled, time-evolving conditions.

4.1 Data distribution shifts

Figure 3 shows the distributions of the two emulator inputs, C^2 and $|\mathbf{u}_b|$, and the target output, α^2 , from the training dataset introduced in Section 3.2. Most C^2 samples are concentrated toward the lower part of the linear range, with progressively fewer samples at larger values. The basal-velocity distribution contains a large concentration of low-velocity samples and a broad range of higher velocities extending to approximately $7 \times 10^3 \text{ m} \cdot \text{a}^{-1}$. A small number of extremely low velocity values appear in the far-left tail of Fig. 3 (b). The α^2 distribution is also skewed, with most samples concentrated near the lower end of the linear range and a small number of values extending to approximately $1.4 \times 10^7 \text{ Pa} \cdot \text{a} \cdot \text{m}^{-1}$. Figure 4 shows the spatial distributions of the two input variables. The C^2 field spans several orders of magnitude, with values predominantly of order 3×10^3 to 3×10^4 across the glacier and localized reductions to approximately 10^0 to 3×10^2 near the margins and terminus. In contrast, basal speeds $|\mathbf{u}_b|$ remain below approximately $1000 \text{ m} \cdot \text{a}^{-1}$ over most of the interior, while speeds exceeding $3000 \text{ m} \cdot \text{a}^{-1}$ are concentrated along the fast-flowing outlet, reaching approximately 5000 to $6000 \text{ m} \cdot \text{a}^{-1}$ near the ice front. Based on the raw data distribution, the velocity threshold is set to be $1.58 \text{ m} \cdot \text{a}^{-1}$.

Figure 5 compares the joint training-data distribution before and after the weighted sampling method as described in Section 3.2. The sampled set provides broader coverage of the occupied input space, particularly for combinations of friction coefficient and basal velocity that occur infrequently in the original simulation data.

After the sampling, each variable is normalized using the mean (μ_{train}) and standard deviation (σ_{train}) calculated from the sampled training set, $x_{\text{norm}} = (x - \mu_{\text{train}})/\sigma_{\text{train}}$. The same transformation parameters are then applied to the validation and test sets. Figure 6 compares the distributions of the normalized, preprocessed variables for the sampled training set and the unchanged validation and test sets. The validation and test curves nearly overlap in all three panels. For normalized $\sqrt{C^2}$, the sampled training distribution is broader, with a peak probability density of approximately 0.53, compared with 1.15 for

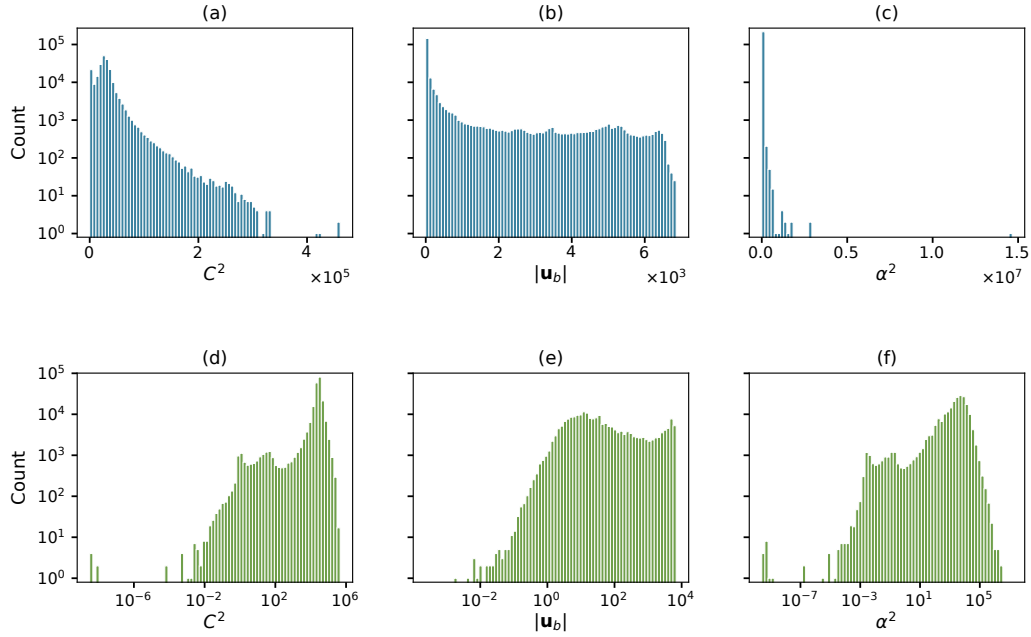


Figure 3. Distributions of the raw friction-emulator variables. Columns, from left to right, show C^2 ($\text{Pa} \cdot \text{a}^{1/3} \cdot \text{m}^{-1/3}$), $|u_b|$ ($\text{m} \cdot \text{a}^{-1}$), and α^2 ($\text{Pa} \cdot \text{a} \cdot \text{m}^{-1}$). Panels (a)–(c) display the three variables on linear horizontal axes, while panels (d)–(f) show the same distributions on logarithmic horizontal axes. The sample counts are plotted on logarithmic vertical axes in all panels. The variables span several orders of magnitude and have skewed distributions.

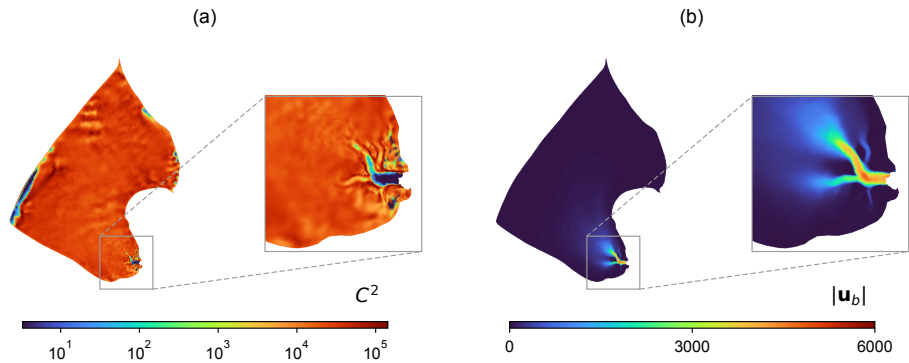


Figure 4. Spatial distributions of (a) the squared friction coefficient, C^2 ($\text{Pa} \cdot \text{a}^{1/3} \cdot \text{m}^{-1/3}$), and (b) basal-velocity magnitude, $|u_b|$ ($\text{m} \cdot \text{a}^{-1}$), over Helheim Glacier. Insets enlarge the glacier-terminus region.

validation and test. For normalized $\sqrt{|u_b|}$, sampling reduces the peak density near -1.2 from approximately 1.5 to 0.48, while increasing the density across intermediate and higher normalized values. Conversely, the sampled training distribution

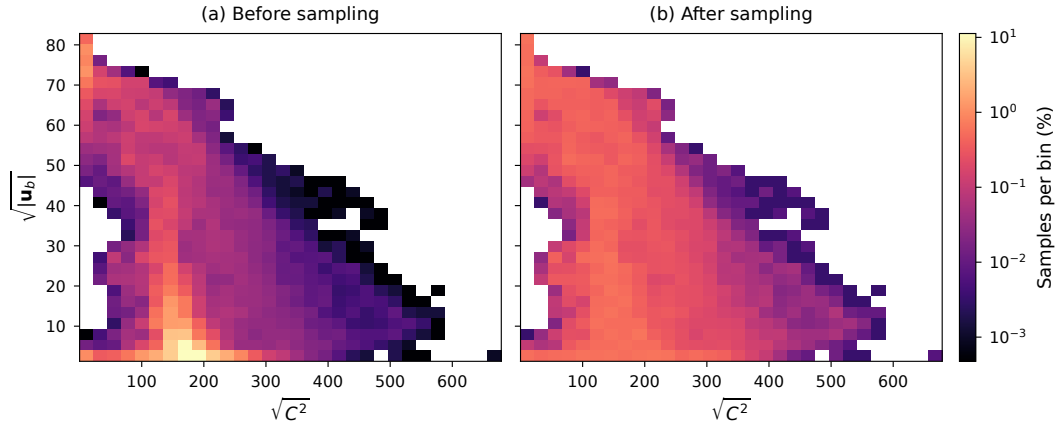


Figure 5. Joint distribution of the transformed training inputs (a) before and (b) after weighted sampling. The axes show $\sqrt{C^2}$ ($\text{Pa}^{1/2} \cdot \text{a}^{1/6} \cdot \text{m}^{-1/6}$) and $\sqrt{|u_b|}$ ($\text{m}^{1/2} \cdot \text{a}^{-1/2}$). Colors indicate the percentage of samples per bin on a logarithmic scale.

of normalized $\sqrt{\alpha^2}$ exhibits a sharper peak near -0.5, with a density of approximately 1.0, compared with peak densities of 0.35-0.37 for validation and test.

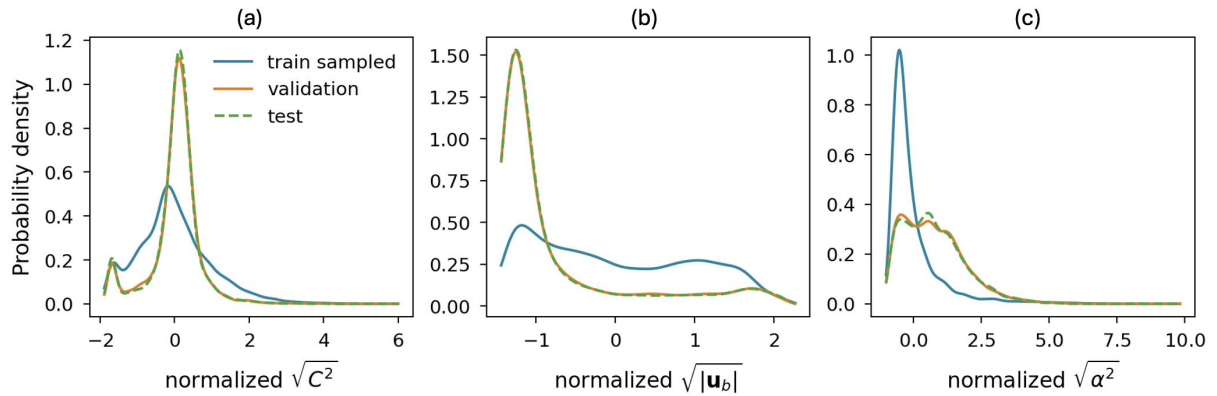


Figure 6. Probability-density distributions of the normalized emulator variables for the sampled training, validation, and test sets: (a) $\sqrt{C^2}$, (b) $\sqrt{|u_b|}$, and (c) $\sqrt{\alpha^2}$. All normalized variables are dimensionless. Only the training set is resampled.

4.2 Offline performance

Figure 7(a) compares the training and validation root mean square errors (RMSEs) of the scaling-only baseline and the full-
 230 preprocessing workflow. In the baseline, no square root transformation is applied, and 30,000 samples are selected uniformly at random, approximately preserving its original distribution. The full-preprocessing workflow instead selects 30,000 samples using the methods in Section 3.2. Both models exhibit a rapid initial reduction in error followed by slower convergence. After

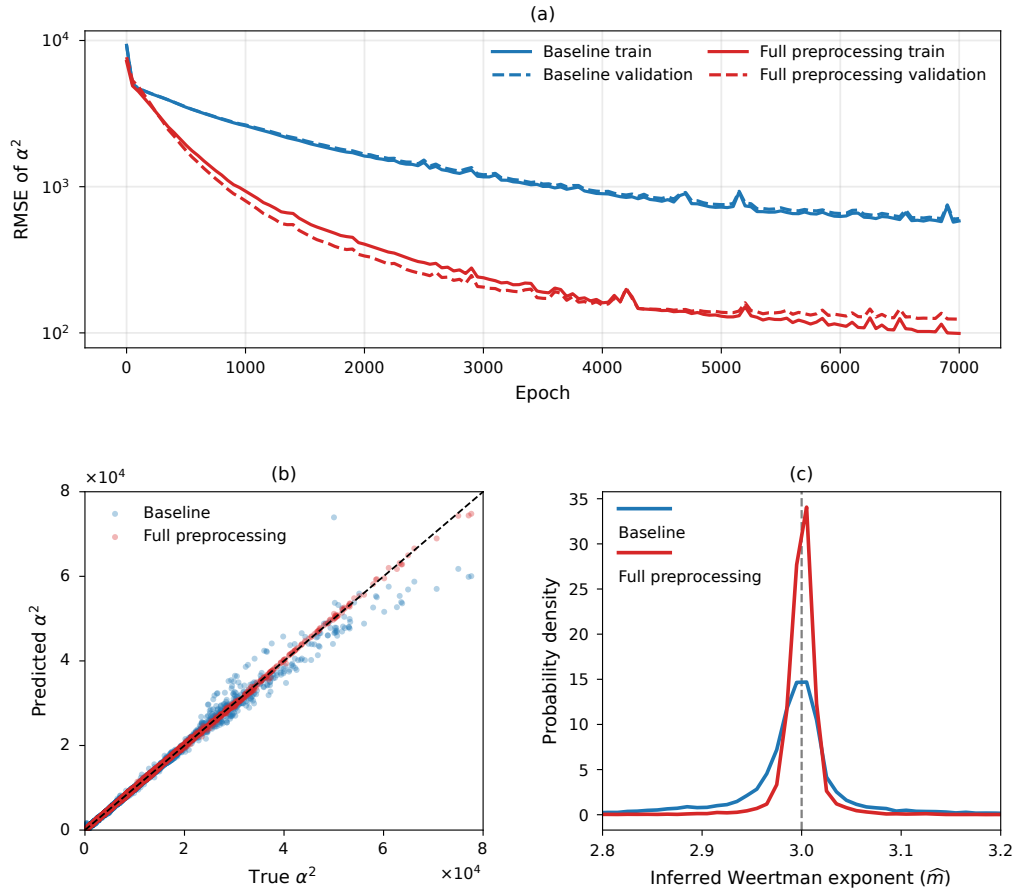


Figure 7. Offline evaluation of the scaling-only baseline (blue) and the emulator trained with the full preprocessing workflow (red). (a) Training (solid) and validation (dashed) RMSEs of α^2 ($\text{Pa} \cdot \text{a} \cdot \text{m}^{-1}$) over 7000 epochs, shown on a logarithmic vertical axis. (b) Predicted versus analytical α^2 ($\text{Pa} \cdot \text{a} \cdot \text{m}^{-1}$) for a subset of the common test set; the dashed black line indicates one-to-one agreement. (c) Probability-density distributions of the inferred Weertman exponent $\hat{m}$. The dashed gray line marks the prescribed value $m = 3$.

7000 epochs, the baseline reaches training and validation RMSEs of 5.83×10^2 and 6.05×10^2 $\text{Pa} \cdot \text{a} \cdot \text{m}^{-1}$, respectively, whereas the full-preprocessing model reaches 9.92×10^1 and 1.24×10^2 $\text{Pa} \cdot \text{a} \cdot \text{m}^{-1}$. Full preprocessing therefore reduces the final validation RMSE by approximately 79.5%. Neither model exhibits a sustained increase in validation error as training progresses.

Figure 7(b) compares the emulator predictions with the analytical values of α^2 . Both models capture the overall relationship, but the baseline case exhibits greater scatter and tends to underpredict the largest values, whereas the full-preprocessing predictions lie much closer to the one-to-one line. Evaluated on the same 31,541 test samples, the coefficient of determination



240 increases from $R^2 = 0.9931$ for the baseline to $R^2 = 0.9997$ for full preprocessing. The corresponding test RMSE decreases from 5.86×10^2 to $1.19 \times 10^2 \text{ Pa} \cdot \text{a} \cdot \text{m}^{-1}$.

We infer the effective sliding exponent from the emulator-predicted pre-factor $\widehat{\alpha}^2$ and the corresponding input variables by rearranging Eq. 2: $\widehat{m} = \left[1 + \log(\widehat{\alpha}^2 / C^2) / \log(|\mathbf{u}_b|) \right]^{-1}$. Figure 7(c) shows the resulting distributions for inferred exponents. Both distributions peak near the prescribed value $m = 3$, but full preprocessing produces a substantially sharper peak. Within
 245 this interval, the mean inferred exponent is 3.000 for full preprocessing and 2.987 for the baseline, and the corresponding standard deviations are 0.044 and 0.102.

4.3 Online prediction

The baseline emulator could not be evaluated over the transient period because the nonlinear stress-balance solver failed to converge after 20 Picard nonlinear solver iterations. The online comparison is therefore limited to the emulator trained with
 250 the full preprocessing workflow.

The emulator trained with the full preprocessing workflow is deployed in a transient simulation of Helheim Glacier from 2007 to 2022. The simulation setup is adapted from Cheng et al. (2022) and Koo et al. (2025) and is the same as that used to generate the dataset in Section 3.2. The emulator-coupled simulation is compared with a reference simulation that uses the analytical Weertman law directly. The two simulations use the same model configuration and differ only in how α^2 is evaluated
 255 (i.e., either using the analytical expression or the emulator trained to approximate this relationship). Their difference therefore represents the substitution error introduced by replacing the analytical relationship with the emulator.

Figure 8a shows the signed velocity difference (emulator minus reference) along the 50 km central flowline shown in Figure 8b. The involvement of the friction emulator generally leads to an underestimation of the velocity upstream, with mean differences of approximately $-40 \text{ m} \cdot \text{a}^{-1}$ over 0-20 km and $-119 \text{ m} \cdot \text{a}^{-1}$ over 30-36 km. Near the terminus, at 40-50 km, the
 260 mean bias is approximately $-80 \text{ m} \cdot \text{a}^{-1}$ during 2007-2010, followed by alternating underestimation and overestimation during 2012-2019, with positive excursions reaching approximately $210 \text{ m} \cdot \text{a}^{-1}$ (2.17%). During 2021-2022, downstream differences remain negative and average approximately $-112 \text{ m} \cdot \text{a}^{-1}$. The mean absolute relative error over 40-50 km is 0.85% across the full simulation period.

5 Discussions

265 5.1 Training with glacier data

The use of simulation-derived training data highlights an important distinction between the number of available samples and the coverage of the physically relevant state space. Samples extracted from a glacier model reflect the spatial distribution of glacier states and the region of parameter space most frequently occupied by the modeled system. Large, slow-flowing regions may therefore contribute many similar samples, whereas less extensive but dynamically distinct regions, such as fast-flowing
 270 outlets, margins, grounding zones, and ice fronts, may be represented by comparatively very few samples. Sample frequency

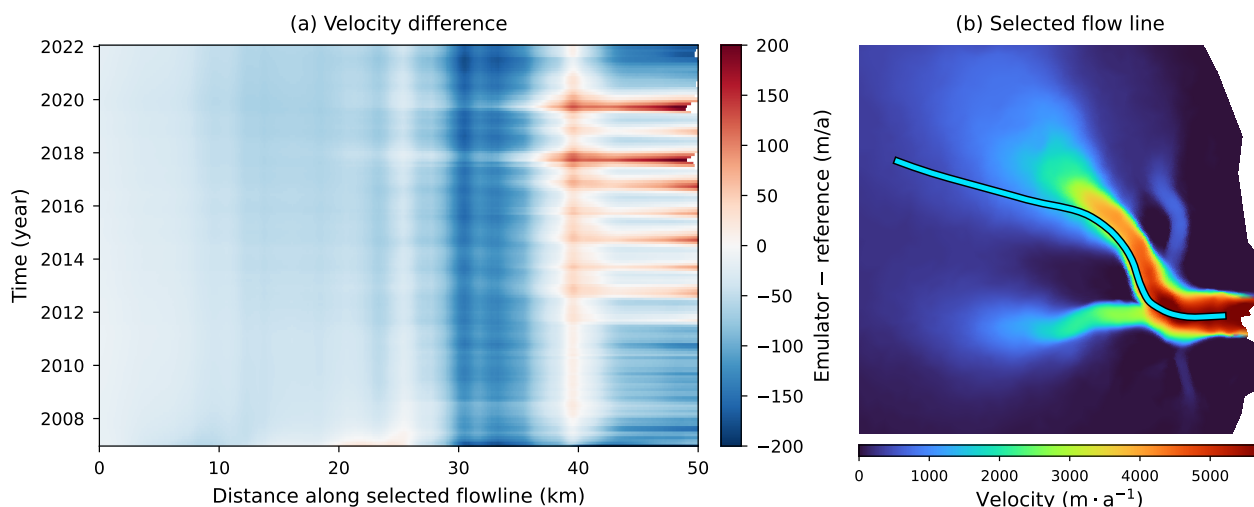


Figure 8. Time–distance evolution of the emulator error in basal-velocity magnitude along the central Helheim flowline. (a) Signed velocity difference, $\Delta|u_b| = |u_b|_{\text{emu}} - |u_b|_{\text{ref}}$ ($\text{m} \cdot \text{a}^{-1}$), as a function of distance along the flowline and time. Positive values indicate emulator overestimation and negative values indicate underestimation. (b) Location of the selected central flowline used for the time–distance comparison.

is thus not necessarily proportional to the dynamical importance of a particular regime. A large glacier derived dataset can consequently remain poorly balanced with respect to the range of states that an ML model may encounter during a simulation.

This issue extends beyond the friction law example considered here. Training data derived from either numerical simulations or observations will generally inherit the spatial and statistical structure of the glacier system from which they are sampled. For
 275 ML components intended to operate within ice sheet models, the construction of the training dataset should therefore consider the coverage of relevant dynamical regimes rather than data volume alone. In our experiment, increasing the representation of sparsely sampled regions of the input space improved the emulator’s ability to recover the prescribed relationship. More generally, this suggests that sampling and preprocessing strategies should be designed with the intended model application, particularly when comparatively rare states may have a disproportionate influence on the numerical solutions.

280 5.2 Coupling ML predictions with numerical solvers

ML predictions introduce new error patterns into numerical solvers. In the friction emulator case, after each nonlinear iteration of the stress balance, the updated velocities change the basal-drag pre-factor and therefore modify the finite element stiffness matrix, which in turn affect the next velocity update. Consequently, errors introduced by ML predictions can accumulate and make the nonlinear solver struggle to converge, as observed in the online test with the baseline workflow. In other
 285 ML–numerical solver coupling applications, errors may propagate in different ways. For instance, a stress-balance emulator coupled with an ice-sheet model may not cause numerical divergence but may instead accumulate errors silently over time. The



velocity predictions from the emulator is used by the mass-balance solver to update the ice thickness, which in turn provides the ice thickness back to the emulator. This feedback loop may remain stable while cumulatively altering the simulation results over time.

290 Ensuring the stability and accuracy of coupled ML–numerical solver models remains an open challenge. In ice-sheet modeling applications, as suggested in the previous section, appropriate data preprocessing is recommended. In addition, hard constraints on ML predictions based on physical understanding may be required to improve numerical stability and bound prediction errors.

5.3 Potential applications

295 The same framework can be applied to processes for which an ML model offers computational or scientific benefits. One potential application is a stress-balance emulator that replaces one of the most computationally expensive components, while ISSM continues to coordinate the remaining analyses and time stepping. Such an emulator could substantially reduce the computational cost of ice-sheet simulations, enabling higher-order, large-scale, long-term, and/or ensemble simulations. Another promising application is the integration of a regional climate-model (RCM) emulator for online surface-mass-balance evaluation. During a transient simulation, ISSM could provide the evolving ice-sheet geometry to the RCM emulator and receive
300 updated surface mass balance in return. Such coupling would allow surface mass balance forcing to respond to changes in ice-sheet geometry more frequently than in existing models that couple to a full RCM through annual exchanges (Paice et al., 2026; Delhasse et al., 2024; Le Clec’h et al., 2019). Other applications include data-driven representations of basal friction, calving, subglacial processes, and ice–ocean interactions, could also benefit from the framework presented in this work.

305 6 Conclusions

We developed ISSM-ML, a framework that allows trained ML models to become part of ISSM computations. We demonstrated the framework using an emulator of the nonlinear relationship in the Weertman friction law. We implemented a data-distribution-based preprocessing workflow that substantially improved offline prediction accuracy and enabled stable transient simulations. By contrast, the baseline emulator, which retained the raw data distribution, caused the nonlinear stress-balance
310 solver to fail to converge despite plausible offline performance. These results demonstrate that ML components intended for numerical-model coupling must be evaluated within the numerical environment in which they operate, rather than on offline prediction metrics alone. Future work is needed to better understand the impact of errors introduced into numerical solvers by ML predictions. We believe that ISSM-ML paves the way toward next-generation data-powered ice-sheet numerical modeling.

Code and data availability. The coupled model is available in the ISSM 2026.4 release (Morlighem et al., 2026). The friction emulator
315 training code is available in the FrictionEmulatorTrain v1.0.0 release (Liu, 2026).



Author contributions. YL designed the ISSM–ML framework, implemented the friction emulator application, performed the simulations and analysis, and drafted the manuscript. MM designed the EmulatorParam class and contributed to the interpretation of the results and revision of the manuscript. GC shaped the scope and direction of the study and contributed to discussions and revision of the manuscript.

Competing interests. The authors declare that they have no conflict of interest

320 *Acknowledgements.* We acknowledge support from the National Science Foundation (award no. 2118285), “HDR Institute: HARP- Harness-
ing Data and Model Revolution in the Polar Regions” and National Aeronautics and Space Administration (award no. 80NSSC24K1888),
“ROSES: Research Opportunities in Space and Earth Science”. GC is also supported by the Fundamental Research Funds for the Central
Universities. During the preparation of this manuscript, the authors used Claude (Anthropic) to assist with language editing and improving
the clarity of the text. The authors reviewed and edited all AI-assisted content and take full responsibility for the accuracy, originality, and
325 integrity of the manuscript.



References

- Antwerpen, R., Tedesco, M., Gentine, P., van de Berg, W. J., and Fettweis, X.: PIXAL: a physics-inspired explainable machine learning architecture for Greenland ice albedo modeling, *The Cryosphere*, 20, 3131–3149, 2026.
- Baumhoer, C. A., Dietz, A. J., Kneisel, C., and Kuenzer, C.: Automated extraction of Antarctic glacier and ice shelf fronts from Sentinel-1 imagery using deep learning, *Remote Sensing*, 11, 2529, <https://doi.org/10.3390/rs11212529>, 2019.
- Baumhoer, C. A., Dietz, A. J., Heidler, K., and Kuenzer, C.: IceLines: a new data set of Antarctic ice shelf front positions, *Scientific Data*, 10, 138, <https://doi.org/10.1038/s41597-023-02045-x>, 2023.
- Bochow, N., Hess, P., and Robinson, A.: Physics-constrained generative machine learning-based high-resolution downscaling of Greenland's surface mass balance and surface temperature, *The Cryosphere*, 20, 1841–1866, <https://doi.org/10.5194/tc-20-1841-2026>, verify publication status (may be in press), 2026.
- Cheng, D., Hayes, W., Larour, E., Mohajerani, Y., Wood, M., Velicogna, I., and Rignot, E.: Calving Front Machine (CALFIN): glacial termini dataset and automated deep learning extraction method for Greenland, 1972–2019, *The Cryosphere*, 15, 1663–1675, 2021.
- Cheng, G., Morlighem, M., Mouginot, J., and Cheng, D.: Helheim Glacier's terminus position controls its seasonal and inter-annual ice flow variability, *Geophysical Research Letters*, 49, e2021GL097085, <https://doi.org/10.1029/2021GL097085>, 2022.
- Cheng, G., Krishna, M., and Morlighem, M.: A Python library for solving ice sheet modeling problems using physics-informed neural networks, PINNACLE v1.0, *Geoscientific Model Development*, 18, 5311–5327, <https://doi.org/10.5194/gmd-18-5311-2025>, 2025.
- Choi, Y., Morlighem, M., Rignot, E., Wood, M., and Bondzio, J. H.: Impact of time-dependent data assimilation on ice flow model initialization: a case study of Kjer Glacier, Greenland, *The Cryosphere*, 17, 5499–5518, <https://doi.org/10.5194/tc-17-5499-2023>, 2023.
- Cybenko, G.: Approximation by superpositions of a sigmoidal function, *Mathematics of control, signals and systems*, 2, 303–314, 1989.
- Delhasse, A., Beckmann, J., Kittel, C., and Fettweis, X.: Coupling mar (modèle atmosphérique régional) with pism (parallel ice sheet model) mitigates the positive melt–elevation feedback, *The Cryosphere*, 18, 633–651, 2024.
- Fischler, Y., Rückamp, M., Bischof, C., Aizinger, V., Morlighem, M., and Humbert, A.: A scalability study of the Ice-sheet and Sea-level System Model (ISSM, version 4.18), *Geoscientific Model Development*, 15, 3753–3771, <https://doi.org/gmd-15-3753-2022>, 2022.
- Fowler, A.: A theoretical treatment of the sliding of glaciers in the absense of cavitation, *Philosophical Transactions of the Royal Society of London. Series A, Mathematical and Physical Sciences*, 298, 637–681, 1981.
- Goelzer, H., Nowicki, S., Payne, A., Larour, E., Seroussi, H., Lipscomb, W. H., Gregory, J., Abe-Ouchi, A., Shepherd, A., Simon, E., et al.: The future sea-level contribution of the Greenland ice sheet: a multi-model ensemble study of ISMIP6, *The Cryosphere Discussions*, 2020, 1–43, <https://doi.org/10.5194/tc-14-3071-2020>, 2020.
- Goldberg, D. N., Heimbach, P., Joughin, I., and Smith, B.: Data assimilation using a hybrid ice flow model, *The Cryosphere*, 5, 315–327, <https://doi.org/10.5194/tc-5-315-2011>, 2011.
- Hornik, K., Stinchcombe, M., and White, H.: Multilayer feedforward networks are universal approximators, *Neural networks*, 2, 359–366, 1989.
- Jakob, W., Rhineland, J., and Moldovan, D.: pybind11: Seamless operability between C++11 and Python, <https://github.com/pybind/pybind11>, version 2.10.4, 2017.
- Jouvet, G.: Deep learning speeds up ice flow modelling by several orders of magnitude, *Journal of Glaciology*, 69, 651–664, <https://doi.org/10.1017/jog.2022.120>, 2023.



- Jouvet, G. and Cordonnier, G.: Ice-flow model emulator based on physics-informed deep learning, *Journal of Glaciology*, 69, 1941–1955, <https://doi.org/10.1017/jog.2023.73>, 2023.
- Karniadakis, G. E., Kevrekidis, I. G., Lu, L., Perdikaris, P., Wang, S., and Yang, L.: Physics-informed machine learning, *Nature Reviews Physics*, 3, 422–440, <https://doi.org/10.1038/s42254-021-00314-5>, 2021.
- 365 Koo, Y., Cheng, G., Morlighem, M., and Rahnemounfar, M.: Calibrating calving parameterizations using graph neural network emulators: application to Helheim Glacier, East Greenland, *The Cryosphere*, 19, 2583–2599, <https://doi.org/10.5194/tc-19-2583-2025>, 2025.
- Lai, C.-Y., Kingslake, J., Wearing, M. G., Chen, P.-H. C., Gentine, P., Li, H., Spergel, J. J., and van Wessem, J. M.: Vulnerability of Antarctica’s ice shelves to meltwater-driven fracture, *Nature*, 584, 574–578, 2020.
- 370 Larour, E., Seroussi, H., Morlighem, M., and Rignot, E.: Continental scale, high order, high spatial resolution, ice sheet modeling using the Ice Sheet System Model (ISSM), *Journal of Geophysical Research: Earth Surface*, 117, <https://doi.org/10.1029/2011JF002140>, 2012.
- Le Clec’h, S., Charbit, S., Quiquet, A., Fettweis, X., Dumas, C., Kageyama, M., Wyard, C., and Ritz, C.: Assessment of the Greenland ice sheet–atmosphere feedbacks for the next century with a regional atmospheric model coupled to an ice sheet model, *The Cryosphere*, 13, 373–395, 2019.
- 375 Liu, Y.: Friction Emulator Training Code (v1.0.1), <https://doi.org/10.5281/zenodo.22756528>, 2026.
- MacAyeal, D. R.: A tutorial on the use of control methods in ice-sheet modeling, *Journal of Glaciology*, 39, 91–98, <https://doi.org/10.3189/S0022143000015744>, 1993.
- Morlighem, M., Rignot, E., Seroussi, H., Larour, E., Ben Dhia, H., and Aubry, D.: Spatial patterns of basal drag inferred using control methods from a full-Stokes and simpler models for Pine Island Glacier, West Antarctica, *Geophysical Research Letters*, 37, L14 502, <https://doi.org/10.1029/2010GL043853>, 2010.
- 380 Morlighem, M., Quinn, J., Gong, C., Park, I., NJSchlegel, caronlam, larour, hseroussi, Poinelli, M., Hew, J. K. J., Sagebaum, M., Liu, Y., Getraer, B., Hepburn, A., aleahsommers, Adhikari, S., Bird, L., Harvey, T., Krishna, M., dabele, Menemenlis, D., Mankoff, K., Brätner, A. G. M., and Shao, N.: ISSMteam/ISSM: 2026.4, <https://doi.org/10.5281/zenodo.23243038>, 2026.
- Nair, V. and Hinton, G. E.: Rectified linear units improve restricted boltzmann machines, in: *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814, 2010.
- 385 Paice, C. M., Fettweis, X., and Huybrechts, P.: Positive feedbacks drive the Greenland ice sheet evolution in millennial-length MAR–GISM simulations under a high-end warming scenario, *The Cryosphere*, 20, 309–332, 2026.
- Pattyn, F. and Morlighem, M.: The uncertain future of the Antarctic Ice Sheet, *Science*, 367, 1331–1335, <https://doi.org/10.1126/science.aaz5487>, 2020.
- 390 Rosenblatt, F.: The perceptron: a probabilistic model for information storage and organization in the brain., *Psychological review*, 65, 386, 1958.
- Rumelhart, D. E., Hinton, G. E., and Williams, R. J.: Learning representations by back-propagating errors, *nature*, 323, 533–536, 1986.
- Schlager, E., Scher, S., Mottram, R. H., and Langen, P. L.: Learning to melt: Emulating Greenland surface melt from a polar RCM with machine learning, *The Cryosphere*, 20, 3313–3343, 2026.
- 395 Seroussi, H., Nowicki, S., Payne, A. J., Goelzer, H., Lipscomb, W. H., Abe-Ouchi, A., Agosta, C., Albrecht, T., Asay-Davis, X., Barthel, A., et al.: ISMIP6 Antarctica: a multi-model ensemble of the Antarctic ice sheet evolution over the 21st century, *The Cryosphere*, 14, 3033–3070, <https://doi.org/10.5194/tc-14-3033-2020>, 2020.



400

- Van Der Meer, M., de Roda Husman, S., and Lhermitte, S.: Deep learning regional climate model emulators: a comparison of two downscaling training frameworks, *Journal of Advances in Modeling Earth Systems*, 15, e2022MS003 593, <https://doi.org/10.1029/2022MS003593>, 2023.
- Weertman, J.: On the sliding of glaciers, *Journal of glaciology*, 3, 33–38, 1957.
- Willard, J., Jia, X., Xu, S., Steinbach, M., and Kumar, V.: Integrating scientific knowledge with machine learning for engineering and environmental systems, *ACM Computing Surveys*, 55, 1–37, 2022.