Journal of Glaciology

Article

Deep learning speeds up ice flow modelling by several orders of magnitude

Guillaume Jouvet[1], Guillaume Cordonnier[2,3], Byungsoo Kim[2], Martin Lüthi[1], Andreas Vieli[1] and Andy Aschwanden[4,5]

[1]Department of Geography, University of Zurich, Zurich, Switzerland; [2]Department of Computer Science, Computer Graphics Laboratory, ETH Zurich, Switzerland; [3]Université Côte d'Azur and INRIA, Sophia-Antipolis, France; [4]Geophysical Institute, University of Alaska Fairbanks, Fairbanks, AK, USA and [5]Department of Glaciology and Climate, Geological Survey of Denmark and Greenland, Denmark

Cite this article: Jouvet G, Cordonnier G, Kim B, Lüthi M, Vieli A, Aschwanden A (2021). Deep learning speeds up ice flow modelling by several orders of magnitude. *Journal of Glaciology* 1–14. https://doi.org/10.1017/jog.2021.120

Received: 27 April 2021
Revised: 20 October 2021
Accepted: 20 October 2021

Keywords:
Glacier flow; glacier modelling; ice dynamics; ice velocity

Author for correspondence:
Guillaume Jouvet,
E-mail: guillaume.jouvet@geo.uzh.ch

Abstract

This paper introduces the Instructed Glacier Model (IGM) – a model that simulates ice dynamics, mass balance and its coupling to predict the evolution of glaciers, icefields or ice sheets. The novelty of IGM is that it models the ice flow by a Convolutional Neural Network, which is trained from data generated with hybrid SIA + SSA or Stokes ice flow models. By doing so, the most computationally demanding model component is substituted by a cheap emulator. Once trained with representative data, we demonstrate that IGM permits to model mountain glaciers up to 1000 × faster than Stokes ones on Central Processing Units (CPU) with fidelity levels above 90% in terms of ice flow solutions leading to nearly identical transient thickness evolution. Switching to the GPU often permits additional significant speed-ups, especially when emulating Stokes dynamics or/and modelling at high spatial resolution. IGM is an open-source Python code which deals with two-dimensional (2-D) gridded input and output data. Together with a companion library of trained ice flow emulators, IGM permits user-friendly, highly efficient and mechanically state-of-the-art glacier and icefields simulations.

Introduction

Glacier and ice-sheet models are valuable tools to assess their future evolution and the resulting sea-level rise under climate warming (Pattyn, 2018). In the past two decades, tremendous efforts have been made by the glaciological community to develop models to account for the most relevant underlying physical processes such as ice flow, thermodynamics, subglacial hydrology and their coupling with the atmosphere (e.g. climate-driven surface mass balance), the lithosphere, and the ocean (e.g. iceberg calving or subaquatic melt). However, the added complexity of these models comes with increasing computational cost, which cannot be offset entirely by recent advances in scalable numerical methods and increasing computing power.

Since the 1950s, ice is commonly treated as a viscous, non-Newtonian fluid (Glen, 1953) best described by the Stokes equations, which are computationally expensive to solve. Although numerical simulations of real small glaciers have been possible since the late 1990s (e.g. Gudmundsson, 1999), the simulation of large icefields and ice sheets at high spatial resolution and/or over multi-millennial time scales remains challenging with today's available computational resources, with a few exceptions: Seddik and others (2012) performed a 100-year simulation of the entire Greenland Ice Sheet while Cohen and others (2018) modelled a few millennia of the former Rhone Glacier at the Last Glacial Maximum with Elmer/Ice (Gagliardini and others, 2013). As a consequence, most models frequently solve computationally less expensive approximations to the Stokes equations.

The Shallow Ice Approximation (SIA; Hutter, 1983), a zeroth-order approximation to the Stokes equations, remains a reference model for many applications (e.g. Maussion and others, 2019; Višnjević and others, 2020), despite strongly-simplifying mechanical assumptions and applicability limited to areas where ice flow is dominated by vertical shearing (Greve and Blatter, 2009). As a compromise between mechanical accuracy and computational costs, a family of models of intermediate complexity has emerged such as the hybrid SIA + SSA (Shallow Shelf Approximation) (Bueler and Brown, 2009) implemented in the Parallel Ice Sheet Model (PISM) (Khroulev and the PISM Authors, 2020). This class of approximations reduces the stress balance to at most 2-D equations, thus facilitating simulations over time scales of glacial cycles (Seguinot and others, 2018). With the exception of sub-glacial hydrology models (Werder and others, 2013), the costs associated with the other glacier-related model components (e.g.mass balance, ice energy, lithospheric displacement, calving) remain computationally low. Well-established and state-of-the-art simulation tools such as PISM or Elmer/Ice already use efficient and scalable numerical solvers, thus limiting the potential for further improvements in computational efficiency. Yet computational efficiency in high-order ice flow modelling is crucial to (i) explore a large variety of model parameters, (ii) model long time scales in paleo applications, (iii) refine the spatial resolution when dealing with complex topography, and (iv) to reduce model biases where SIA-like models are used today due to computational constraints.

In recent years, Graphics Processing Units (GPU), which feature more but slower cores compared to Central Processing Units (CPU), have gained interest in ice flow modelling to overcome the aforementioned limitations, and to obtain significant speed-ups (Räss and others, 2020). The key for using GPUs efficiently is to implement numerical schemes that can be divided into several thousand parallel tasks; a challenge regarding the viscous behaviour of ice, and the underlying diffusion equations that describe its motion. To our knowledge, two approaches have been attempted to achieve this high level of parallelization. The first consisted of solving SIA (Višnjević and others, 2020) or Second Order SIA (Brædstrup and others, 2014), explicitly in time, while the second consisted of solving the Stokes equations using finite differences to take advantage of numerical stencil-based techniques (Räss and others, 2020). The first approach permitted a large ensemble or long time scales simulations with applications to invert climatic parameters from observed glacial extents at the last glacial maximum (Višnjević and others, 2020) and to model glacial erosion and landscape evolution over glacial cycle time scales (Egholm and others, 2017). The necessity of coding in a dedicated programming language (e.g. CUDA) has probably hindered the use of GPUs. However, the emergence of user-friendly Python libraries such as TensorFlow and PyTorch, which allow running relatively simple code on GPUs, will certainly contribute to popularize it in the coming years.

Compared to physic-based numerical models (referred here as instructor models), statistical emulators (or surrogate models), which mimic the behaviour of the simulator as closely as possible, are computationally cheap to evaluate. Surrogate models are solely constructed from intelligently chosen input-output data of the instructor model without any knowledge of its inner working. The arrival of machine learning has led to the development of deep learning-based surrogate models (Reichstein and others, 2019) to accelerate computational fluid dynamics (CFD) codes (e.g. Ladicky and others, 2015; Tompson and others, 2016; Kim and others, 2019; Obiols-Sales and others, 2020). The main idea is to take advantage of the large amount of modelling results (data) that a CFD solver can produce to train a neural network emulator delivering high fidelity solutions at much lower computational cost. The fidelity of the emulated solution, relative to the instructor model, is directly dictated by the emulator complexity and the quality of the training dataset, which must be representative of all dynamical states. This approach can therefore be seen as a way to compress a large number of model realizations (Kim and others, 2019), and to take benefit of this information for new model runs that are fairly close to the runs already performed in generating the training dataset. With the exception of Brinkerhoff and others (2021), this idea has never been exploited in ice flow modelling. Yet – from a CFD point of view – the most accurate and most expensive ice flow model, the Stokes model, shows no major difficulties for its numerical solving: it is a purely diffusive (although non-linear) non-advective, time-independent problem, and its solution solely depends on the geometry, and given fields such as the ice hardness or the basal sliding coefficient. Furthermore, the same solver often recomputes states which are close to those which were computed before for parameter studies (Aschwanden and others, 2019) or multi-glacial cycle applications (Sutter and others, 2019). In such cases, the strategy described above can be very beneficial to use previously computed ice flow modelled states and save this information to emulate a cheap model trained by deep learning. This paper intends to explore the potential of constructing ice flow model emulators.

In the machine learning paradigm, Artificial Neural Networks (ANNs) have been increasingly used in recent years to deal with various kinds of problems such as image classification (Simonyan and Zisserman, 2015), segmentation (Ronneberger and others, 2015) and domain transfer (Isola and others, 2017), whenever a large-scale training dataset is available (LeCun and others, 2015). Classical examples of classifiers are email filtering to identify spam (Dada and others, 2019), handwriting recognition (Cireşan and others, 2010)or image segmentation for biomedical applications (Yang and others, 2017). In the case of image analysis, a breakthrough occurred in the early 2010s (LeCun and others, 2015) thanks to successful Convolutional Neural Networks (CNN) that are today widely used in image recognition or classification. CNNs are especially good at recognizing image features or spatial patterns due to convolution operations, and their optimization is computationally tractable owing to a strategy of shared weights to reduce the number of tuning parameters.

In glaciology, ANNs have been used for estimating bed topography (Clarke and others, 2009; Leong and Horgan, 2020; Monnier and Zhu, 2021), to infer basal conditions at the bedrock (Brinkerhoff and others, 2021; Riel and others, 2021), to model mass balance (Bolibar and others, 2020) or to identify the calving fronts of tidewater glaciers from satellite images (Baumhoer and others, 2019; Mohajerani and others, 2019; Zhang and others, 2019; Cheng and others, 2021). Note that the last three studies all rely on semantic image segmentation using CNN. Recently, Brinkerhoff and others (2021) used a neural network to emulate an expensive coupled ice flow/subglacial hydrology model, and infer the optimal parameters that best reproduce the observed surface velocities using a Bayesian approach.

In this paper, we apply deep learning to ice flow modelling. Our approach consists of setting up a CNN that predicts ice flow from given topographic variables and basal sliding parametrization in a generic manner. By contrast, Brinkerhoff and others (2021) emulated a coupled ice flow-hydrology model for a specific glacier from a small-size ensemble of relevant parameters. Our neural network emulator is trained from a large dataset, which is generated from ice flow simulations obtained from two state-of-the-art models – the PISM (Khroulev and the PISM Authors, 2020) and CfsFlow (Jouvet and others, 2008, 2009) – equipped with two different mechanics (hybrid SIA + SSA and Stokes) and at two different spatial resolutions (2 km and 100 m). We integrate mass balance and mass conservation with our ice flow emulator to obtain a time evolution model, the 'Instructed Glacier Model' (IGM) written in Python, which permits highly efficient, and mechanically state-of-the-art ice flow simulations.

In the following, we first describe the IGM model with its neural network-based emulator and the generation of the training dataset. Then, we present our results in terms of fidelity and computational performance of the emulated ice flow model with respect to the instructor model, and of embedding the ice flow emulator into a time evolution model. Last, we compare and discuss the IGM results with state-of-the-art ice flow model results.

Methods

IGM couples the submodels of surface mass balance, mass conservation and ice dynamics as depicted in Figure 1. Specifically, IGM models the ice flow by a CNN emulator, which is trained by PISM and CfsFlow realizations.

From now on, we denote $b(x, y)$, $h(x, y, t)$ and $s(x, y, t) = b(x, y) + h(x, y, t)$ bedrock elevation (assumed to be fixed in time), ice thicknessand ice surface elevation, respectively (Fig. 2). We call $\bar{\mathbf{u}} = (\bar{u}, \bar{v})$ the vertically-averaged horizontal ice velocity field, and SMB the Surface Mass Balance function, which consists of yearly-average accumulation (when positive) or ablation (when negative). Given $\bar{\mathbf{u}}$ and the SMB function, and ignoring basal melt, the evolution in ice thickness h is determined by the mass

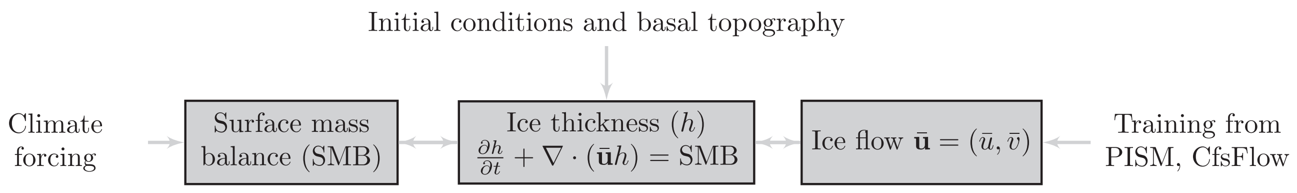
Initial conditions and basal topography
Climate forcing
Surface mass balance (SMB)
Ice thickness (h)
$\frac{\partial h}{\partial t} + \nabla \cdot (\bar{\mathbf{u}} h) = \text{SMB}$
Ice flow $\bar{\mathbf{u}} = (\bar{u}, \bar{v})$
Training from PISM, CfsFlow

Fig. 1. Interactions between the model components and the input data of IGM.

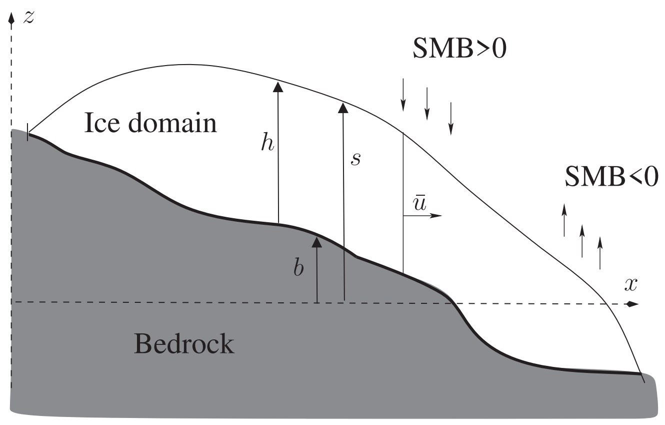

Fig. 2. Cross-section of a glacier with notations.

conservation equation:

$$\frac{\partial h}{\partial t} + \nabla \cdot (\bar{\mathbf{u}} h) = \text{SMB}, \tag{1}$$

which states the balance between the change in ice thickness $\partial h/\partial t$, the dynamical thinning/thickening $\nabla \cdot (\bar{\mathbf{u}} h)$ due to the ice flux $\bar{\mathbf{u}} h$, and the adding/removal of ice on the top surface by the SMB (Fig. 2). Here $\nabla \cdot$ denotes the divergence operator with respect to horizontal variables (x, y).

We assume the horizontal model domain to be a rectangle subdivided by a regular 2-D grid with uniform spacing Δx in x and y – the variables h, s, b, $\bar{u}$, $\bar{v}$, SMB being defined at the centre of each cell. Let t^0 be the initial time, $\{t_k\}_{k=0,1,\ldots}$ be a discretization of time with variable steps $\Delta t^{k+1} = t^{k+1} - t^k$. We denote by h^k an approximation of h at time $t = t^k$, and similarly for all other variables. Given an initial ice thickness h^0, IGM updates the ice flow $(\bar{u}^k, \bar{v}^k)$, the surface mass balance SMB^k, and the ice thickness h^k sequentially as follows:

(I) **Ice flow:** Given the ice thickness h^k and the surface slope fields $(\partial s^k/\partial x, \partial s^k/\partial y)$, the ice flow emulator provides the vertically-integrated ice flow $(\bar{u}^{k+1}, \bar{v}^{k+1})$.

(II) **Surface mass balance**: Given the ice surface elevation $b + h^k$, the surface mass balance SMB^k is computed using two models: a simple one based on given Equilibrium Line Altitude (ELA) or a combined accumulation-Positive Degree-Day (PDD) model (cf. Hock, 2003) based on climate data (see Appendices B and C for further details).

(III) **Ice thickness:** Given the vertically-averaged ice flow $(\bar{u}^{k+1}, \bar{v}^{k+1})$, and the surface mass balance SMB^k, update the ice thickness h^{k+1} by solving the mass conservation equation (1) using the first-order explicit upwind finite-volume scheme on a staggered grid (e.g. Lipscomb and others, 2019), which has the advantage to be mass-conserving. To ensure stability of the scheme, the time step Δt must satisfy the CFL condition:

$$\Delta t \leq C \times \frac{\Delta x}{\|\bar{\mathbf{u}}\|_{L^\infty}}, \tag{2}$$

where $C < 1$ and Δx is the grid cell spacing. Condition (2) simply ensures that ice is never transported over more than one cell distance in one time step. Here, one uses CFL number C between 0.3 and 0.5. Further details about solving of (1) is reported in Appendix A.

In the following, we focus on step (I) as it is the key innovation of the paper; steps (II) and (III) are standard modelling practices.

Ice flow emulator

Our ice flow emulator predicts vertically averaged horizontal velocity from ice thickness, surface slope and basal sliding coefficient c, which is defined later in Eqn. (6):

$$\mathcal{M}: \underset{\mathbb{R}^{N_X \times N_Y \times 4}}{\left\{h, \frac{\partial s}{\partial x}, \frac{\partial s}{\partial y}, c\right\}} \underset{\longrightarrow}{\longrightarrow} \underset{\mathbb{R}^{N_X \times N_Y \times 2}}{\{\bar{u}, \bar{v}\}} \tag{3}$$

where input and output are 2-D fields, which are defined over the discretized computational domain (or subparts) of size $N_X \times N_Y$.

We approximate $\mathcal{M}$ by means of an ANN $\mathcal{M}_\mathbf{p}$ (see Appendix E for a digest on ANN and LeCun and others (2015) for an in-depth review). ANNs map input to output variables using a sequence of network layers connected by trainable linear and non-linear operations with weights $\mathbf{p} = [p_1, \ldots, p_N]$, which are adjusted (or trained) to a dataset (realizations of an instructor model in the present case). Here we use a CNN (Long and others, 2015), which is a special type of ANN that additionally includes local convolution operations to extract translation-invariant features as trainable objects and then learn spatially-variable relationships from given fields of data (LeCun and others, 2015). CNNs are therefore suitable to learn from a high-order ice flow model, which determines the velocity solution from topographical variables and their spatial variations. Note that in contrast to high-order models, the SIA determines the ice flow from the local topography without using further spatial variability information in the vicinity.

Our CNN consists of N_{lay} 2-D convolutional layers between the input and output data (Fig. 3). Passing from one to the next layer consists of a sequence of linear and non-linear operations:

(1) Convolutional operations multiply input matrix with a trainable kernel matrix (or feature map) of size $N_{\text{ker}} \times N_{\text{ker}}$ to produce an output matrix (Fig. 4). A padding is used to conserve the frame size through the convolution operation. Convolutional operations are repeated using a sliding window with one stride across the input frame, and for N_{feat} the number of feature maps.

(2) As a non-linear activation function, we use leaky Rectified Linear Units (Maas and others, 2013), which was found more robust than the standard ReLU.

Only for the last convolutional layer, we instead use a linear activation function. Various combinations of $(N_{\text{lay}}, N_{\text{feat}}, N_{\text{ker}})$ are tested later on (Table 1), and optimal parameter sets in terms of model fidelity to computational performance are retained.

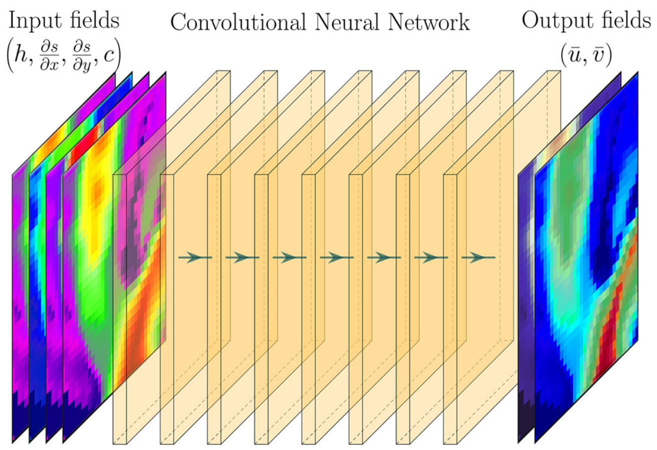
Input fields
$\left(h, \frac{\partial s}{\partial x}, \frac{\partial s}{\partial y}, c\right)$
Convolutional Neural Network
Output fields
$(\bar{u}, \bar{v})$

Fig. 3. The function we aim to emulate by learning from hybrid SIA + SSA or Stokes realizations maps geometrical fields (thickness and surface slopes) and basal sliding parametrization to ice flow fields.

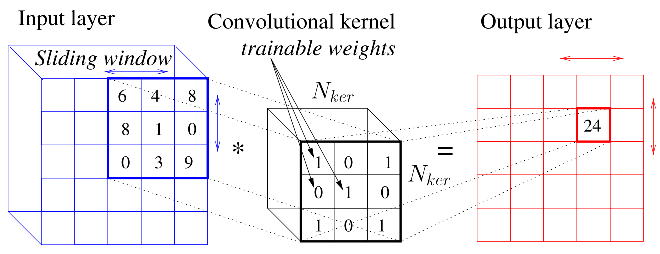

Fig. 4. Illustration of one convolution operation between two layers: the elements of the input matrix and the kernel matrix are multiplied and summed to construct the output entry. The operation is repeated with a one stride sliding window to fill the output layer. The frame size is conserved using a padding, which consists of augmenting the input matrix by zeros on the border (not shown).

An advantage of CNNs is that the size of the input/output may vary as our network consists of successive convolution operations, which are inherent to the window size. One can therefore train and evaluate it with various sizes $N_X \times N_Y$. The only requirement is that for training, the window $N_X \times N_Y$ must be sufficiently large to carry the relevant information for prediction. In this paper, we use $N_X = N_Y = 32$.

Our CNN has N trainable weights $\mathbf{p} = [p_1, \ldots, p_N]$, which increases with parameters N_{lay}, N_{feat} and N_{ker}, and need to be adjusted to data. This stage – called training – is performed by minimizing a loss (or cost) function, which measures the misfit between the predicted ice flow, $\bar{\mathbf{u}}_{\mathrm{P}}$, and the reference ice flow, $\bar{\mathbf{u}}_R$. While there are several possible choices of loss (e.g. the mean squared error L_2, the mean absolute error L_1or more general L_p errors), we opted for the error L_1 loss function by simply imitating existing neural networks (Kim and others, 2019; Thuerey and others, 2020) designed for emulating CFD solutions:

$$\|\bar{\mathbf{u}}_{\mathrm{P}} - \bar{\mathbf{u}}_{\mathrm{R}}\|_{L^1} = \frac{1}{|\Omega|}\int |\bar{\mathbf{u}}_{\mathrm{P}} - \bar{\mathbf{u}}_{\mathrm{R}}|_1 \, \mathrm{d}\Omega. \tag{4}$$

The loss function is minimized on GPU using a mini-batch gradient descent method, namely the Adam optimizer (Kingma and Ba, 2015) with a learning rate of 0.0001, and a batch size of 64. Let us note that we use a norm clipping to protect against gradient vanishing or exploding behaviour. To reach a satisfying level of convergence, we usually iterate between 100 and 200 epochs, which means that the optimization algorithm passes 100–200 times over the entire dataset (Appendix D).

Data generation for training and validation

To train our ice flow emulator and assess its accuracy and performance, we perform an ensemble of simulations to generate large datasets using two ice flow models of variable complexity: the PISM (Khroulev and the PISM Authors, 2020), and CfsFlow (Jouvet and others, 2008). The goal is to construct diverse states to obtain a heterogeneous dataset that a large variety of possible glaciers (large/narrow, thin/thick, flat/steep, long/small, fast/slow, straight/curved glaciers, …) that can be met in future modelling. For simplicity, we assume the ice to be always at the pressure melting point to focus on the dynamics in this study.

PISM and CfsFlow simulate the evolution of the ice thickness combining ice flow and mass-balance models for given basal topography, initial conditions and climate forcing as depicted in Figure 1. In both models, the ice deformation is modelled by Glen's flow law (Glen, 1953):

$$\dot{D} = A\tau^n, \tag{5}$$

where $\dot{D}$ is the strain rate tensor, τ is the deviatoric stress tensor, $A = 78\ \mathrm{MPa}^{-3}\ \mathrm{a}^{-1}$ is the rate factor for temperate ice (Cuffey and Paterson, 2010), and n is Glen's exponent, which is taken equal to 3. Basal sliding is modelled with a non-linear sliding law – known as Weertman's law (Weertman, 1957):

$$u_{\mathrm{b}} = c\tau_{\mathrm{b}}^{1/m}, \tag{6}$$

Table 1. Fidelity and performance results of our neural network trained from the icefield (PISM) and glacier (CfsFlow) simulations for different network parameters. The L_1 validation loss (m a^{-1}) is the mean absolute discrepancy between the neural network and the reference velocity solutions. For convenience, we also provide the L_1 misfit relative (in %) defined by Eqn. (7). Selected (optimal) models used in the remainder of the paper are marked with bold numbers and tagged in the leftmost column. The performance consists of the average time to compute an entire ice flow field using GPU (NVIDIA Quadro P3200 GPU card with 1792 1.3 GHz cores) and CPU (Intel(R) Core(TM) i7-8850H CPU with 6 2.6 GHz cores)

					Icefields/PISM			Glaciers/CfsFlow		
	Neural network parameters				Fidelity	Performance		Fidelity	Performance	
Selected models for	# trainable weights N	N_{feat}	N_{ker}	N_{lay}	L_1 valid. loss (m a^{-1} (%))	GPU (sec)	CPU (sec)	L_1 valid. loss (m a^{-1} (%))	GPU (sec)	CPU (sec)
	9000	8	3	16	5.4 (17)	0.033	0.094	7.6 (19)	0.006	0.014
	35000	16	3	16	3.3 (12)	0.039	0.142	4.5 (11)	0.007	0.018
	140000	32	3	16	2.6 (8)	0.055	0.366	3.2 (8)	0.008	0.030
	556000	64	3	16	2.4 (7)	0.105	1.082	2.7 (7)	0.012	0.090
	140000	32	3	16	2.6 (8)	0.055	0.366	3.2 (8)	0.008	0.0306
	386000	32	5	16	3.1 (9)	0.139	0.803	3.0 (8)	0.017	0.072
	28000	32	3	4	3.3 (12)	0.030	0.086	8.4 (20)	0.005	0.010
Icefields ↠	65000	32	3	8	**2.7 (9)**	**0.038**	**0.176**	4.9 (12)	0.007	0.016
Glaciers ↠	140000	32	3	16	2.6 (8)	0.055	0.366	**3.2 (8)**	**0.008**	**0.030**
	288000	32	3	32	2.8 (9)	0.089	0.713	3.4 (8)	0.012	0.058

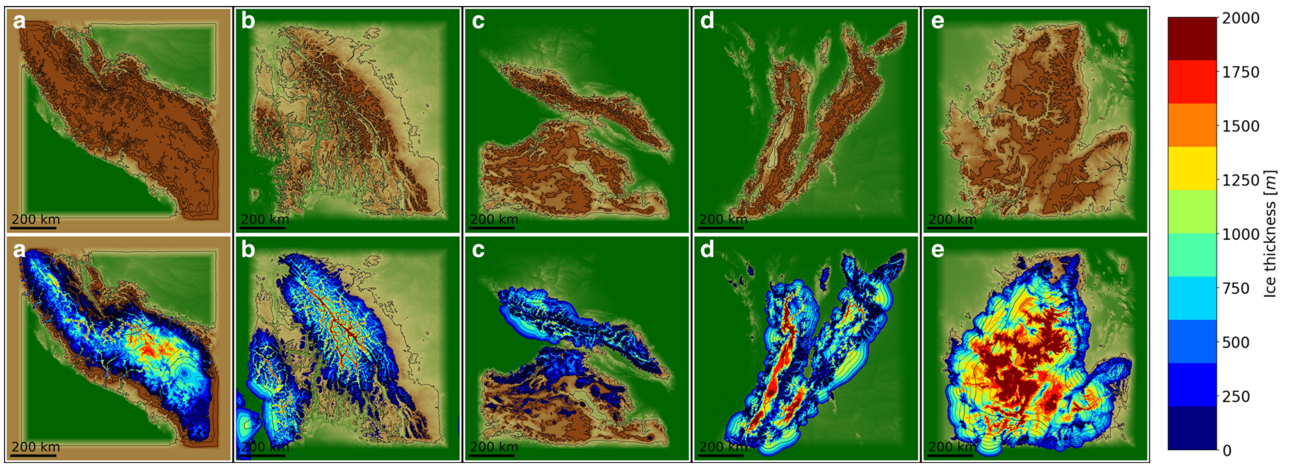

Fig. 5. Topographies of the fives 1024 km × 1024 km selected tiles of mountain ranges (upper panel) used to produce icefield simulations with PISM and simulated maximal state (bottom panel).

where u_b is the norm of the basal velocity, τ_b is the magnitude of the basal shear stress, $m = 1/3$ is a constant parameter and c is the sliding coefficient. The main difference between CfsFlow PISM simulations concerns the solving of the momentum balance. While CfsFlow solves the full set of equations, PISM uses a linear combination of the SIA for the vertical shearing and the SSA for the longitudinal ice extension. This low-order hybrid approach is a trade-off between mechanical accuracy and computational cost that permits the model to run over long time scales (e.g. glacial cycles Seguinot and others, 2018) – a task not achievable with the more mechanically complete CfsFlow.

We perform two types of simulations – icefield-scale with PISM and individual glaciers with CfsFlow. In both cases, the models are initialized with ice-free conditions, and the mass balance is chosen to produce a glacier advance followed by retreat to span over a wide panel of glacier shapes:

(1) **Icefields with PISM:** We simulate the time evolution of five synthetic icefields inspired by the geometry of today's Alaska icefields (Ziemen and others, 2016). As bedrock topography, we take five 1024 km × 1024 km tiles of existing mountainous range worldwide (Table 5 and Fig. 5). Basal sliding is modelled with law (6) with fixed parameter $c = 70$ km MPa^{-3} a^{-1}. For each tile, we run PISM at 2 km resolution with a combined accumulation-PDD model (cf. Hock, 2003) and initialize it with present-day climate. Then we gradually reduce air temperatures (up to 12–20°C) for ≈1000–2000 years to obtain sufficiently large glaciers, and then warm it again until all glaciers disappear. As a result, our five simulations produce a large number of different glacier shapes from very small individual glaciers to large ice caps during time scales of 3 millennia (Fig. 5). The dynamical and topographical results are recorded each 20 years, providing a representation of many states relevant for training IGM. In total, they consist of $\approx 5 \times 150$ snapshots of grid size 512×512 of glaciated mountain ranges. Further modelling details are given in Appendix B.

(2) **Valley glaciers with CfsFlow:** We simulate 200 years time evolution of 41 glaciers that are artificially built on existing topographies. For that purpose, we take valleys from the European Alps and New Zealand, that are today ice-free but were likely covered by ice during the last glaciation. For each valley, we run CfsFlow at 100 m horizontal resolution and force equilibrium line altitudes in a simple mass-balance model to simulate a 100-year-long advance followed by a 100-year-long retreat. The results are recorded every 2 years to provide a wide range of dynamical states roughly representative of real-world temperate glacier behaviour since the Little Ice Age, consisting of $\approx 41 \times 100$ snapshots. Then, for a chosen subset of ten glaciers (Fig. 6 and Section 'Data selection') among the 41 glaciers, we carry the same simulation varying parameter c in $\{0, 6, 12, 25, 70\}$ km MPa^{-3} a^{-1} to explore different basal sliding conditions. We additionally carry out two further simulations of existing glaciers – Aletsch and Rhone glaciers (Switzerland) – for 250 and 200 years, respectively, that we use for assessing the method accuracy. Further modelling details are given in Appendix C.

In all simulations, the fields of ice thickness, ice surface, sliding coefficient and depth-average velocity, covering various domains are recorded on a structured grid at different times. The transformation of raw modelling data to be usable for training our neural network emulator (Eqn. (3)) consists of the following three steps (the two last are illustrated in Fig. 7):

(1) First, the data are normalized to fit the interval $[-1, 1]$ by dividing by a typical value over the entire dataset for each variable independently. Such a normalization is widespread in deep learning to deal with different ranges of data and homogenize their values prior learning (Raschka and Mirjalili, 2017).

(2) Second, patches of dimension $N_X \times N_Y$ are randomly extracted from all slices of the dataset. The motivation to work patch-wise instead of the entire domain is that (i) ice flow at a given location is only determined by predictors within a certain neighbourhood, it is therefore unnecessary to carry irrelevant distant information for model prediction, (ii) it gives higher flexibility to exclude ice-free patches, which do not carry any relevant information for training, (iii) to apply data augmentation (see next item). Here we found that a $N_X \times N_Y = 32 \times 32$ patch size was suitable for all applications, and we therefore always used this value.

(3) Last, data augmentation (Raschka and Mirjalili, 2017) is applied after patching. For that purpose, we randomly apply 90°, 180°, 270° rotations, as well as vertical and horizontal flipping to the patches shortly after being picked in the dataset. This augmentation increases the amount of data and permits to regularize the underlying optimizing problem,

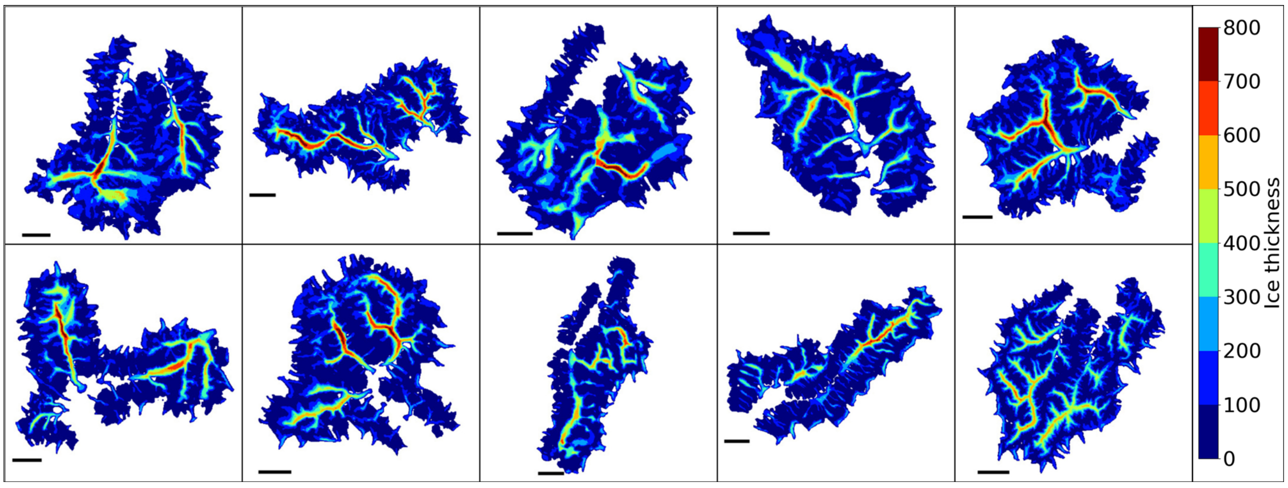

Fig. 6. The ten selected glaciers used for individual glacier simulations with CfsFlow at 100 m resolution used to train IGM's ice flow emulator with various sliding coefficients *c*. The horizontal bar represents 5 km to give the scale of each glacier.

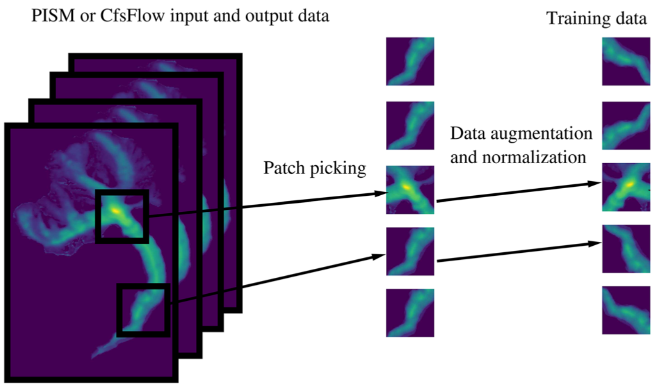

Fig. 7. Illustration of data preparation steps to train IGM including patch extraction and data augmentation.

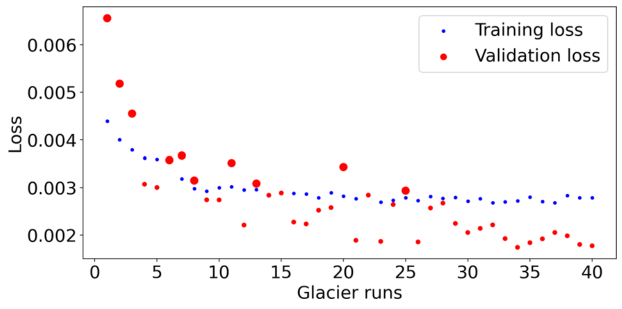

Fig. 8. Evolution of the training and test/validation losses during the iterative selection procedure. Only glaciers with greater validation than training loss were kept in the pool (large dots). The maximum ice thickness of the ten selected glaciers is depicted in Figure 6.

and is supported by the fact that the process we wish to capture – the ice flow – is invariant to these transformations.

Data selection

To reduce redundant data generated by CfsFlow and explore a variety of sliding parameters *c*, we have applied a strategy to select a pool of glaciers that are the most relevant for training (Figs 8 and 6). For that purpose, we first consider the 41 glacier simulations obtained with a constant sliding parameter $c = 0$. We sort the 41 glaciers by ice volume from the most to the least voluminous one at its maximum state. The selection worked as follows: We train the emulator on the first glacier and use the second one for testing. If the test loss is greater than the training loss, it means these data have some added value and we include it to the training pool, otherwise we exclude it. Then we apply the same strategy to the third and loop over the entire set of all available glacier runs. This iterative selection proved to be efficient to minimize the size of data while keeping it heterogeneous (Fig. 8), and to be relatively insensitive to the choice of $c = 0$ (not shown). As a result, only ten glaciers are kept from the 41 original ones (Fig. 8), allowing further simulations to be carried out with varying parameters *c*. Note that this is mostly the largest/thickest glaciers (or the leftmost in Fig. 8) that were kept as they naturally carry more information. Indeed, a small glacier is similar to a large glacier in an early state of advance or a late state of retreat, explaining why it naturally brings little added value to the dataset, and justifying our choice of starting from the most voluminous glaciers.

Implementation

IGM is implemented in Python with the Tensorflow (Abadi and others, 2015) library to evaluate the neural network emulator, solve the mass balance and the mass conservation equation (1) in parallel on CPU or GPU. The training of the neural network emulator was performed separately on GPU using the Keras library (Chollet and others, 2015) with a Tensorflow backend. The Python code as well as some ice flow emulators are publicly available at https://github.com/jouvetg/igm.

Results

In this section, we demonstrate that our trained neural network can generate ice flow solutions with high fidelity and at cheaper computational cost compared to instructors. We then show the same features prevail with the time evolution model (referred as 'Instructed Glacier Model' or IGM) that combines the trained neural network as ice flow model emulator and mass conservation.

Ice flow field

We conducted several experiments with various network parameters to seek for the optimal parameters in terms of model accuracy versus model evaluation costs. For that purpose, we split our dataset in two parts: one for training and one for

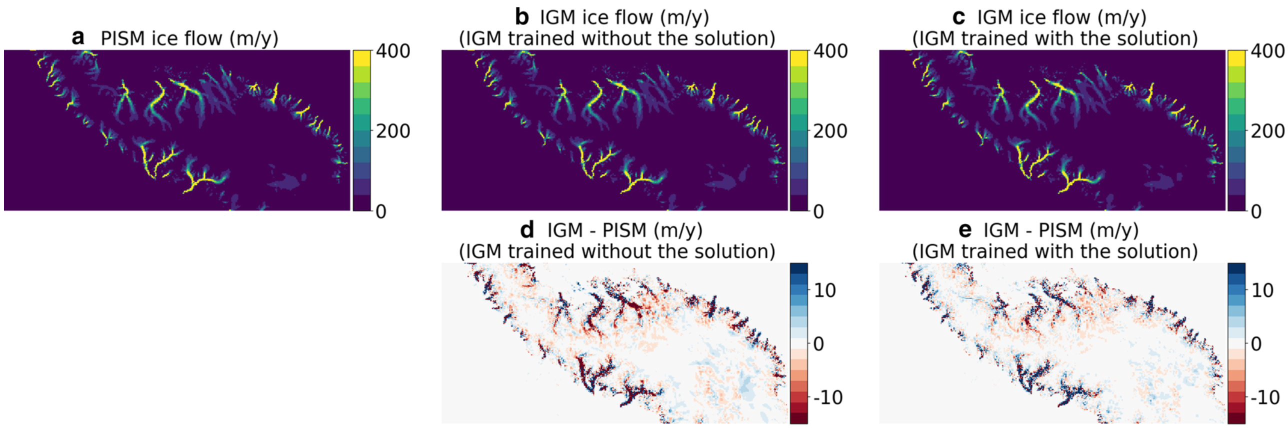
a PISM ice flow (m/y)
b IGM ice flow (m/y) (IGM trained without the solution)
c IGM ice flow (m/y) (IGM trained with the solution)
d IGM - PISM (m/y) (IGM trained without the solution)
e IGM - PISM (m/y) (IGM trained with the solution)
400
200
0
10
0
-10

Fig. 9. Vertically-averaged ice flow magnitude of Icefield A at its maximum state: PISM reference solution (a), IGM solution trained without (b) and with (c) the solution, and the difference between IGM and PISM solutions (d and e).

validation. Appendix D shows the evolution of training and validation losses with respect to epochs. Unless specified differently, we always reserved Icefield A (resp. Aletsch and Rhone glacier with $c = 12$) for validation and take the others for training for icefield (resp. glacier) simulations. Note that the validation loss is always lower than the training loss demonstrating that our choice of validation data remains safely within the hull of the training dataset (Appendix D).

Fidelity

The fidelity is measured by taking the rescaled L_1 validation loss (Eqn. (4)) between the predicted ice flow field $\bar{\mathbf{u}}_P$ and the reference one $\bar{\mathbf{u}}_R$, as well as the L_1 relative error (taken only where the L_1 norm of the ice velocity is above 10 m a^{-1}):

$$\|\bar{\mathbf{u}}_P - \bar{\mathbf{u}}_R\|_{L^1,\mathrm{rel}} = \frac{1}{|\Omega|}\int_{|\bar{\mathbf{u}}_R|_1>10} \frac{|\bar{\mathbf{u}}_P - \bar{\mathbf{u}}_R|_1}{|\bar{\mathbf{u}}_R|_1} d\Omega. \quad (7)$$

Table 1 displays the fidelity results of the trained ice flow emulator to reproduce the ice flow of Icefield A performed with PISM and of Aletsch glacier (with fixed sliding parameter $c = 12$) performed with CfsFlow varying network parameters such as the number of layers N_{lay}, output filters N_{feat} or kernel size N_{ker}. It shows that increasing the number of filters N_{feat} improves the fidelity of the solution. This is also true for the number of layers, however, only up to $N_{\mathrm{lay}} = 16$ layers as the fidelity deteriorates when deepening the network. Last, increasing the kernel size N_{ker} from 3 (the most common value, see Fig. 4) does not improve the solution in all cases. Opting for large N_{lay}, N_{ker} and N_{feat} numbers increases the number of network parameters, and thus the computational cost. Therefore, we selected two optimal parameter sets (one for each kind of dataset) that lead to fidelity levels above 90% while keeping low evaluation costs: (i) the parameter set (N_{feat}, N_{ker}, N_{lay}) = (32, 3, 8) for PISM icefield data leads to a neural network that has ≈65 k trainable parameters (it stores into a 880 KB file), and (ii) the parameter set (N_{feat}, N_{ker}, N_{lay}) = (32, 3, 16) for CfsFlow glacier data leads to a neural network that has ≈140 k trainable parameters (it stores into a 2 MB file).

Figures 9 and 10 compare the flow speeds of Icefield A and Aletsch glacier at the maximum ice extent to the reference simulations with the respective selected models. The fidelity of the emulated solution, compared to the reference PISM or CfsFlow solutions (which were not used in the training stage), is high and of the same order in all cases with mean relative errors below 9% (Table 1). This shows the neural network's ability to learn from data from different topographies and to apply this knowledge to new ones, even with a relative small model size (65 and 140 k of trainable parameters, respectively).

A closer look at the emulated ice flow field of Aletsch glacier (Fig. 10) shows that the ice flow is well reproduced for the three accumulation basins, however, larger discrepancies occur on the tongue and at Konkordiaplatz where the three tributary basins merge into a single tongue. While the training dataset generated by CfsFlow contains many examples of single channelized flow (Fig. 6), merging ice flows of similar sizes such as at Konkordiaplatz as well as thick ice (up to 800 m) is much less common in the training dataset, explaining why the flow in this region is less well reproduced than anywhere else. By contrast, single-branch and thinner valley glaciers such as Rhone Glacier

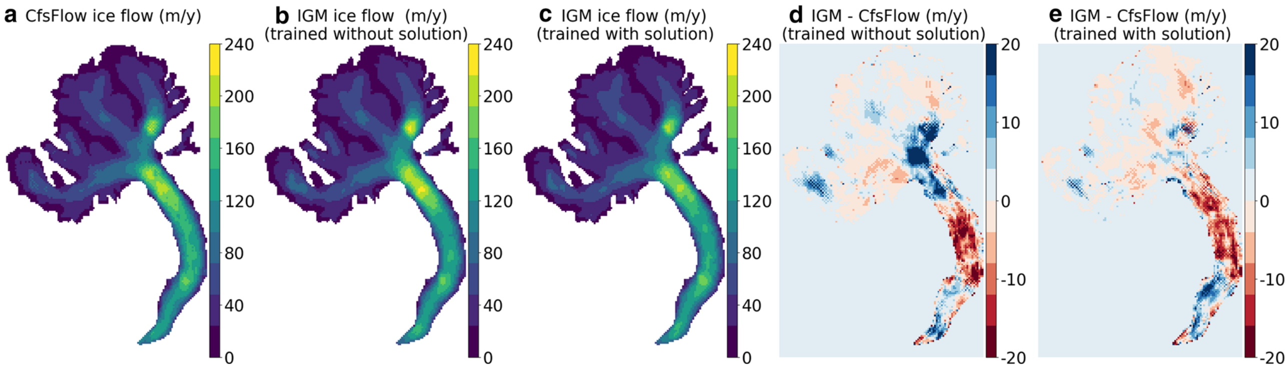
a CfsFlow ice flow (m/y)
b IGM ice flow (m/y) (trained without solution)
c IGM ice flow (m/y) (trained with solution)
d IGM - CfsFlow (m/y) (trained without solution)
e IGM - CfsFlow (m/y) (trained with solution)
240
200
160
120
80
40
0
20
10
0
-10
-20

Fig. 10. Vertically-averaged ice flow magnitude of the Aletsch glacier at its maximum state: CfsFlow reference solution (a), the IGM solution trained without (b) and with (c) the solution, and the difference between IGM and CfsFlow solutions (d and e).

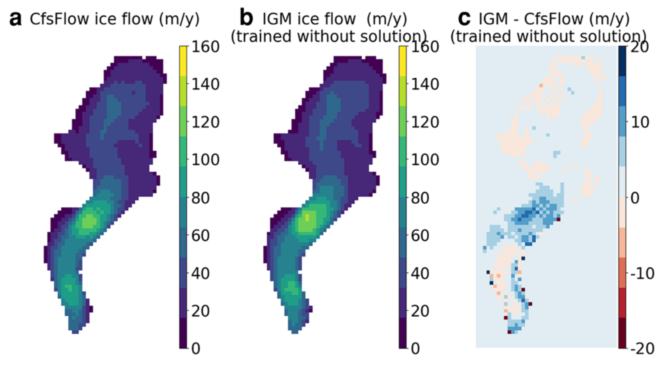
a CfsFlow ice flow (m/y)
b IGM ice flow (m/y) (trained without solution)
c IGM - CfsFlow (m/y) (trained without solution)
160
140
120
100
80
60
40
20
0
20
10
0
-10
-20

Fig. 11. Vertically-averaged ice flow magnitude of the Rhone glacier at its maximum state: CfsFlow reference solution (a), IGM/test solutions (b), and the difference between IGM and CfsFlow solutions (c).

are much more common and better represented in the training dataset. As a result (Fig. 11) the emulated ice flow is in better agreement with the reference CfsFlow one.

The L_1 error (or loss) between a predicted and a reference ice flow has two components, a 'network' component that measures the performance of the network to compress and recover solutions and a 'data' component that measures the error related to the abundance/lack of relevant training data. These two components can be distinguished by including the solution we wish to predict to the training dataset to measure the sole 'network' component (or compression error). Here we found that this error component is ~7% in both cases, which is ~1−2% less than the L_1 original validation loss (~8−9%). We therefore attribute the 7% to network errors and the remainder (~1−2%) to a lack of representative data. The reference solution, and the predicted solutions with or without the reference in the training are displayed in Figures 9 and 10. In the case of Aletsch glacier, the spatial error patterns show that including the reference solution in the training locally resolves the discrepancy at the convergence area (Konkordiaplatz) of the three ice flow branches (Fig. 10), confirming that this particular source of error mostly lies on the data side (i.e. the absence of analogues in the training dataset) unlike the remaining errors on the main glacier trunk.

Up to now, we have tested the learning of ice flow from different geometries, but leaving the sliding coefficient c unchanged ($c = 12$ km MPa^{-3} a^{-1}). Table 2 gives the L_1 validation loss (m a^{-1}) at the maximum extent of Rhone and Aletsch glacier simulations with different sliding coefficients c, including the values used for training $\{0, 6, 12, 25\}$, as well as intermediate values $\{3, 9, 18\}$ to assess the interpolation accuracy. As a result, the values used for training yield to similar fidelity levels (<9%) while intermediate values yield to slightly deteriorated levels (<13%).

Computational performance

Along with the fidelity results, Table 1 also displays the time needed to evaluate a single full ice flow field (after training) with the neural network and varying parameters using both computational resources of the same laptop (Thinkpad Lenovo P52): a GPU (NVIDIA Quadro P3200 GPU card with 1792 1.3 GHz cores) or a CPU (Intel(R) Core(TM) i7-8850H CPU with 6 2.6 GHz cores). For convenience, all CPU and GPU performance results given in this paper relate to this hardware. As a result, GPU and CPU show dramatically different performances. GPU always outperforms CPU (speed-ups between 2 and 8), however, the increasing computational cost coming from the model size and the domain resolution (larger for icefield than for glaciers) scales differently between GPU and CPU. While GPU and CPU have close performance for models with few trainable parameters and small domains, the GPU is especially advantageous for complex models (with many weights) and for large size domains.

While a key advantage of neural networks is their low evaluation cost, their training necessarily comes with substantial upstream computation costs, which must be invested for each set of ice flow settings. Table 3 lists these upstream costs for generating the data from traditional models and training the model itself. As a result, the generation of the full dataset was the most computationally expensive task: ≈7–26 days for icefield simulations with PISM and glacier simulations with CfsFlow on CPU, respectively, while the training of the optimal models took 8–24 h on GPU.

Table 2. L_1 validation loss (m a^{-1}) and relative misfit (in %) defined by Eqn. (7) for Rhone and Aletsch glaciers with different sliding coefficients h including values that have been used for training (in bold), as well as intermediate values

Sliding coeff. c km MPa^{-3} a^{-1}	Rhone Glacier m a^{-1} (%)	Aletsch Glacier m a^{-1} (%)
0	2.3 (9)	3.7 (10)
3	3.1 (13)	3.8 (12)
6	2.3 (7)	3.0 (8)
9	2.7 (7)	3.3 (8)
12	2.8 (8)	3.6 (8)
18	4.8 (12)	5.0 (12)
25	4.4 (9)	5.0 (9)

Table 3. Computational costs required for the generation of all datasets on the CPU and for the training of the emulator on the GPU

Simulation	Data (CPU) generation	Emulator (GPU) training
Icefield	~150 h	~8 h
Glacier	~625 h	~24 h

Icefield and glacier evolutions

We now assess the fidelity and performance of the time-evolution model (Fig. 1) by replicating Icefield A, Aletsch and Rhone glacier transient simulations with IGM. It must be stressed that none of original Icefield A, Aletsch and Rhone glacier simulations were used for training IGM.

Fidelity

Overall, IGM shows good skill at reproducing ice thickness of Icefield A (Fig. 12), and Aletsch and Rhone glaciers at maximum ice extent (Fig. 13). The time-integrated root-mean-square error in terms of ice thickness is ≈20 m in all cases. IGM also reproduces the evolution of ice volume, ice extent and mean ice flow speed with high fidelity (Fig. 14) although small discrepancies arise, e.g. the ice flow of Aletsch Glacier is slightly overestimated with IGM. In the latter case, the cumulative errors remain very limited despite systematic errors in the ice flow (e.g. at the convergence area of the three ice flow branches, Fig. 10).

Computational performance

Table 4 compares the computational times required for Icefield A and Aletsch glacier simulations with all models on CPU and GPU. Note that a direct comparison is only possible on CPU as none of PISM or CfsFlow runs on GPU. Additionally, we forced IGM to use a single core for comparison with CfsFlow as the latter only runs in serial. It must be stressed that the given speed-up does not include the training costs (data generation and training itself summarized in Table 3).

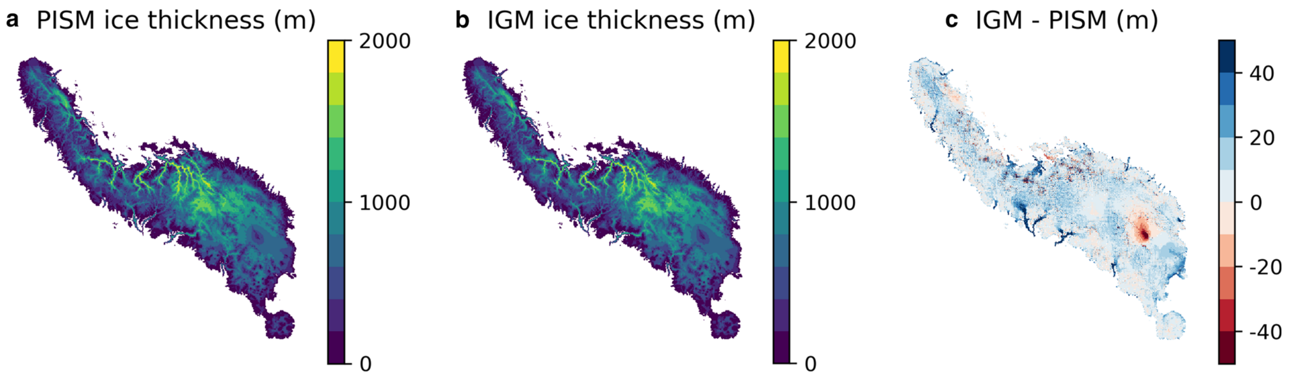

Fig. 12. Maximum ice thickness of Icefield A modelled with PISM (a) and IGM (b), and the difference between the two (c).

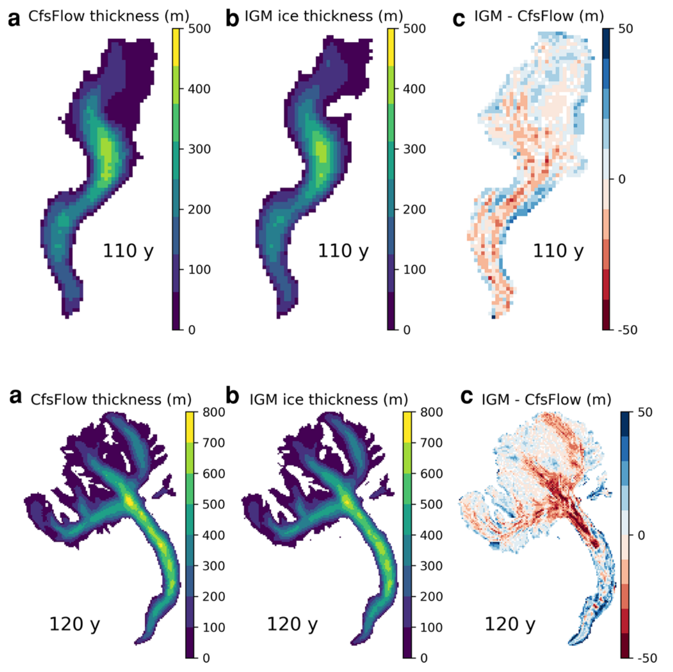

Fig. 13. Ice thickness fields of Rhone (top panels) and Aletsch (bottom panels) glaciers after 110 and 120 years with CfsFlow (a) and IGM (b), as well as the difference between the two (c).

Table 4 shows that IGM outperforms PISM for the Icefield A simulation by a factor of ~20 on CPU, with an additional $\sim 10\times$ speed-up on GPU. Here, the high resolution of the icefield computational domain (512×512) takes full advantage of the parallelism and explains the superiority of the GPU over the CPU. Note that while most of the time is taken by the ice flow model in glacier simulations, the PDD mass-balance model (Appendix B) in the icefield simulation requires significant aside computational effort (even on the GPU) as it loops over each grid cell at short time-scale intervals.

On the other hand, IGM outperforms CfsFlow for the simulation of Aletsch Glacier by a factor of ~600 on CPU (using a single core), with an additional $\sim 10\times$ speed-up on GPU, but only $\sim 2\times$ when comparing to a six-core CPU run (not shown). Here, the GPU remains superior due to the large size of the Stokes ice flow neural network as noted before, however, the added value is less important than before as the size of the computational domain is much lower (185×121). As an additional benchmark in Table 4, we have compared the computational time to simulate the full retreat of Aletsch glacier for 150 years from today's state with another Stokes model – Elmer/Ice (Gagliardini and others, 2013) – and IGM, which was trained from Elmer/Ice simulations. As a result, we found a $\sim 1000\times$ speed-up when comparing IGM and Elmer/Ice runs on CPU (both using all six cores), with an additional speed-up of $\sim 2\times$ on GPU.

Discussion

In ice flow modelling, the choice of a stress balance model is often the result of a compromise between mechanical complexity, spatial resolution and computational affordability. Shallow models rely on simplifications to make them computationally tractable but at the expense of mechanical loss of accuracy (e.g. Greve and Blatter, 2009). In contrast, the Stokes model is computationally expensive, and it is therefore challenging to use it for large domains and long time scales. The transfer of these models directly to GPUs offers promising speed-ups (Räss and others, 2020), however this usually requires a full reimplementation of the numerical model. As an added value to traditional modelling, our results demonstrate that substituting a well-trained neural network emulator running on GPUs for a traditional hybrid or Stokes solver permits to speed up by several orders of magnitude if one excludes a minor loss in accuracy and training costs, which are invested a single time for a given ice flow setting.

Network compression capability

We find that a relatively simple CNN composed of tens of layers is capable of learning from realizations of two ice flow models of different complexity levels and of retrieving composite solutions with high and similar fidelity (above 90%) with respect to the instructor models. The model size in terms of trainable parameters (10^5–10^6) – or equivalently in terms of data storage (1–2 MB) – is very small compared to the amount of data that was used for training (above 1 GB). The misfit between emulated and reference solutions due to the neural network itself (i.e. the compressibility error) is close to 7–8% for the emulation of both CfsFlow and PISM. This is likely much smaller than data and model uncertainties for many glaciological applications. By contrast, the error related to the richness of the dataset to include a large diversity of relevant dynamical states is fairly small (<2%). This therefore demonstrates the ability to learn the relationship between topographic and ice flow variables in a generic manner, i.e. to translate the knowledge acquired on some glaciers to others. Most importantly, a model as complex as the 3-D Stokes equations can be learnt and emulated very efficiently by a neural network, which maps 2-D fields.

Because using standard network parameters already results in a high fidelity emulator, we have not explored further parameter choices, thus leaving room for future improvements. For instance, we used the L_1 norm as our loss function, which is a common choice in deep learning accelerating CFD (Kim and others,

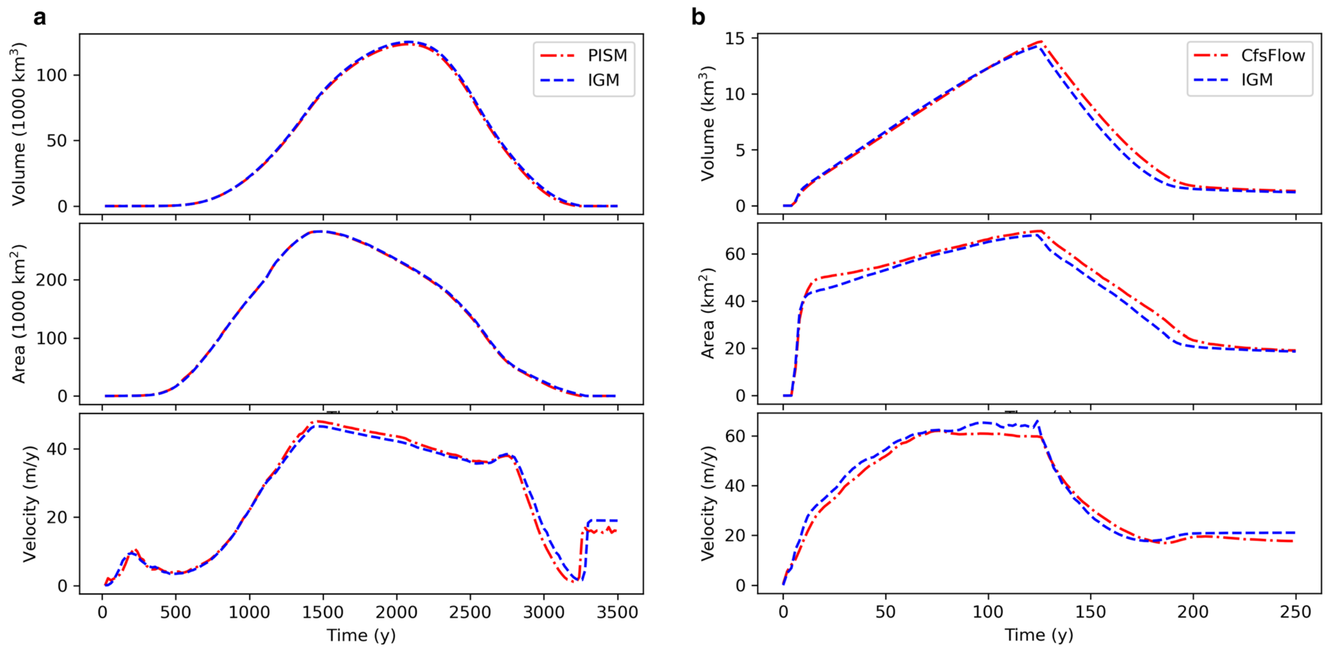

Fig. 14. Modelled evolution of ice volume, glaciated area and mean velocity field for Icefield A (a) and Aletsch glacier (b) simulations performed with IGM, PISM and CfsFlow.

Table 4. Overall computational time to achieve the simulation of Icefield A with PISM and IGM using CPU and GPU at 2 km resolution (top table) and of Aletsch glacier with CfsFlow and IGM using CPU and GPU (middle table). The bottom panel compares the computational times to simulate the retreat of Aletsch glacier for 150 years from today's state with another Stokes model Elmer/Ice and IGM, which was trained from Elmer/Ice simulations. The proportion of this time taken by the ice flow and mass-balance model components is additionally given when they are significant

Icefield A simulation (3500 years)		
Model	CPU	GPU
# cores	6	1792
PISM (all)	**36200** s	–
IGM (all)	1522 s	185 s
– Ice flow component	77%	58%
– Mass-balance component	21%	37%
Aletsch glacier simulation (200 years)		
Model	CPU	GPU
# cores	1	1792
CfsFlow (all)	**49860** s	–
IGM (all)	90 s	8 s
– Ice flow	98%	85%
Additional Aletsch glacier retreat simulation (150 years)		
Model	CPU	GPU
# cores	6	1792
Elmer/ice (all)	**12600** s	–
IGM (all)	13 s	5 s
– Ice flow component	95%	80%

2019; Thuerey and others, 2020), future studies could explore other loss/misfit functions such as L_2, L_p, … or other gradient-based norms such as $W^{1,p}$ with $p = 1 + 1/n$, which is the natural norm in which to analyse the existence and uniqueness of solutions of the Stokes problem in its variational (or minimization) form (Jouvet and Rappaz, 2011).

Optimality of the dataset

We have found that our original dataset made of arbitrarily chosen glacier/valley topographies to train IGM was far from optimal (e.g. containing redundant data). Criteria for the optimality of the dataset (in terms of model fidelity to data size) were investigated.

Table 5. List of tiles used to generate icefields simulations with PISM at 2 km resolution. The longitude and the latitude correspond to the location of the centreof each squared tiles

Icefield	Lon.	Lat.	Geo. location
A	−72.6	−14.70	Andes
B	−117.5	51.0	Canada
C	44.0	42.0	Caucasus
D	−75.0	5.0	Colombia
E	38.0	10.0	Ethiopia

First, we attempted (not shown) to use principal component analysis to preprocess and reduce the dimensionality of data while preserving the original structure and relevant relationships (e.g. Brinkerhoff and others, 2021). However, our iterative selection scheme was found to be more efficient to filter non-relevant data: the glacier dataset was reduced by ≈75% without affecting the emulator accuracy.

Varying basal sliding conditions

Real-world glacier modelling requires the tuning of ice flow parameters to observational data. For that purpose, our Stokes emulator can take the basal sliding coefficient as input thanks to an exploration of this parameter in the training dataset. Our results have shown that the interpolation errors between the training states remain fairly low (Table 2). Note that while we used spatially constant sliding coefficients for training, our emulator can take a spatially variable sliding coefficient (e.g. resulting from surface data assimilation) as input field. In such a case, it would be desirable to assess the emulator accuracy against reference solutions based on spatially sliding coefficients to evaluate whether the training data should be augmented with such solutions.

Embedding into the time evolution model

While even a small <10% inaccuracy of the ice flow emulator could raise concerns of cumulative errors in the time evolution model due to the feedback between ice flow and elevation-dependent mass balance, the good match between time evolution variables (Figs 12–14) suggests instead that the errors compensate. Additional investigations have shown that the errors of the divergence of the ice flux (not shown), which matters in the time-advancing scheme, are much more unevenly distributed than ice flow errors, leading to probable error compensation. This suggests that embedding our neural network emulator iteratively into the mass conservation equation is a safe operation, which keeps a high level of fidelity.

Speed-ups

The significant speed-up obtained here is the combined result of two key ingredients: (i) neural networks (once trained) are very cheap compared to the direct solving of non-linear diffusion equations describing the ice flow; (ii) the evaluation of such a neural network, as well as other expensive IGM model component tasks (like the PDD mass balance) runs well in parallel and can therefore take advantage of GPUs. Our performance results demonstrated that (i) a direct speed-up of ≈20 and 600–1000 for hybrid and Stokes mechanics on our examples related to CPUs, (ii) GPU can give another substantial speed-up, which is harder to quantify as it depends on modelling features and GPU performance. Indeed, GPUs outperform CPUs in general for large computations when all GPU cores available are used. Here, GPUs will show great advantage when the neural network size is large (such as the one selected to emulate Stokes) or/and when the size of the modelled domain is high. We can therefore anticipate a great added value of GPU for emulating Stokes and/or large size computations (e.g. the modelling of large ice sheets in high resolution). Finally, we used here a single laptop integrated GPU card (NVIDIA Quadro P3200 GPU), which features moderate performance (<2000 1.3 GHz cores and 6 GB of memory). There is therefore room for further speed-up by switching to the latest available GPUs, considering that the price performance of GPUs currently doubles roughly every two years.

Advantages and limitations of IGM

Besides the computational efficiency to obtain the ice flow solution at near Stokes accuracy, IGM has a certain number of practical advantages:

(1) Although it can be trained on 3-D ice flow models, IGM deals with 2-D regular grids facilitating the management of input and output data.
(2) IGM only takes gridded bedrock and surface elevations as topographic inputs, and does not require to identify any catchment or centre flow line, in contrast to flowline-based models.
(3) IGM users can directly do simulations by picking already trained model emulators (which vary according to spatial resolution) from a model collection library that comes along IGM's code.
(4) Although IGM performs better on GPU, it runs across both CPU and GPU, and switching from one to another architecture is trivial.

On the other hand, IGM has the following limitations, which call for further development:

(1) One must keep in mind that the applicability of IGM is dictated by the dataset used for training the emulator. For instance, the CfsFlow-trained emulator presented here will not be able to model the ice flow of glaciers, whose dimensions exceed the hull defined by the set of training glaciers (Fig. 6).
(2) The current emulator assumes isothermal ice for simplicity, i.e. it ignores the effect of ice temperature on ice deformation and basal sliding.
(3) In our approach, we have natural inflow and outflow conditions at the border of the computational domain, and no specific other boundary conditions can be prescribed.
(4) IGM's ice flow emulator works only with regular gridded data, and using unstructured meshes is incompatible with this strategy.

Perspectives

Potential for applications

The computational efficiency of IGM opens perspectives in paleo ice flow modelling, which involves very long time scales and relatively high-resolution computational domains such as: (i) the simulation of large icefields for multi-glacial cycles for reconstruction purposes (Seguinot and others, 2018) or to study glacial erosion and landscape evolution (Egholm and others, 2017) – an application that requires high-order mechanics to capture basal sliding, (ii) the inference of paleo climatic patterns from geomorphological evidence using an inverse modelling (Višnjević and others, 2020).

The ability of IGM to learn ice mechanics from an ensemble of glaciers simulated with a state-of-the-art physical instructor model and translate knowledge acquired from some glaciers may also be exploited in global modelling. Indeed, today's global models all rely on highly simplified SIA-based models contributing to model uncertainties. Providing a training set over a representative sample of glaciers with a range of ice flow parameters, IGM would permit to model a massive number of glaciers with a near Stokes accuracy.

Generalizing the emulator

It is straightforward to generalize our CNN (Fig. 3) with further relevant spatially variable inputs (e.g. ice hardness) to emulate a more generic ice flow model. In the same way, adding information on buoyancy as input would permit to learn from an ice shelf dynamical model, and then generalize the emulator to floating ice. Instead, the main challenge here is to provide a larger ensemble of training data.

Data assimilation

While the current paper focussed on emulating a cheap ice flow emulator from a mechanical model, it does not overcome the necessary step of calibrating ice flow parameters to observational data. Besides their low computational evaluation costs, neural networks rely on automatic differentiation, which is a strong asset for inverse modelling. The optimization of basal conditions (bedrock location and basal sliding) from surface observations through the inversion of an ice flow emulator such as the one we used should be investigated.

In contrast, training a network similar to ours with real observations is a tempting alternative to directly include data assimilation. However, this strategy comes with a number of challenges that should be tackled, including (i) the availability of abundant observational data necessary for training a deep network (especially for the ice thickness) (ii) the embedding of possibly contradictory observational data (explaining real ice flow requires more than ice thicknesses and surface slopes, i.e. basal data which are hard to observe) and (iii) the effect of the data noise due to contradictory observational data on the iterative time evolution scheme. Future research is therefore needed to explore this potential.

Conclusions

We have introduced a new type of glacier model, which computes the ice flow using a deep learning emulator trained from hybrid

SIA + SSA or Stokes mechanical models. Our strategy permits us to compress the dynamical states produced by these models and to use this information to substitute the expensive ice flow model by a cheap emulator and therefore speed up the overall time evolution model considerably. We have demonstrated that the resulting model IGM, after appropriate training, models the flow and evolution of large icefields over millennia and of individual mountain glaciers over centuries up to 20 and 1000 faster than traditional hybrid and Stokes on CPUs with fidelity levels above 90%, and up to 200 and 2000 faster by switching to GPUs. IGM has potential for application in global glacier modelling as well as in paleo and modern ice-sheet simulations. The IGM code and the trained ice flow emulators are publicly available at https://github.com/jouvetg/igm.

Acknowledgements. We acknowledge Ed Bueler, Fabien Maussion and an anonymous referee for their valuable comments on the original manuscript. We are thankful to Constantine Khroulev and Olivier Gagliardini for support with PISM and Elmer/Ice, respectively. The Python code PyPDD by J. Seguinot has greatly helped the integration of the PDD model in IGM. The emulator was originally motivated to speed-up an ice flow model within the framework of project 200021-162444 supported by the Swiss National Science Foundation (SNSF). Michael Imhof is acknowledged for providing the script that served to produce Fig. 5.

Author contributions. G.J. conceived the study, performed the simulations, designed the ice flow emulator with support from G.C. and B.K., wrote the paper with support from all co-authors, and wrote the IGM code with contributions and support from G.C. for the Tensorflow implementation.

References

Abadi M and 39 others (2015) TensorFlow: large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Aschwanden A and 7 others (2019) Contribution of the greenland ice sheet to sea level over the next millennium. *Science Advances* **5**(6), 1–11.

Baumhoer CA, Dietz AJ, Kneisel C and Kuenzer C (2019) Automated extraction of antarctic glacier and ice shelf fronts from sentinel-1 imagery using deep learning. *Remote Sensing* **11**(21), 2529.

Bolibar J and 5 others (2020) Deep learning applied to glacier evolution modelling. *Cryosphere* **14**(2), 565–584.

Brædstrup CF, Damsgaard A and Egholm DL (2014) Ice-sheet modelling accelerated by graphics cards. *Computers & Geosciences* **72**, 210–220.

Brinkerhoff D, Aschwanden A and Fahnestock M (2021) Constraining subglacial processes from surface velocity observations using surrogate-based Bayesian inference. *Journal of Glaciology* **67**, 1–19. doi: 10.1017/jog.2020.112.

Bueler E and Brown J (2009) Shallow shelf approximation as a "sliding law" in a thermomechanically coupled ice sheet model. *Journal of Geophysical Research – Earth Surface* **114**(F3), 1–21.

Cheng D and 6 others (2021) Calving front machine (CALFIN): glacial termini dataset and automated deep learning extraction method for Greenland, 1972–2019. *The Cryosphere* **15**(3), 1663–1675. doi: 10.5194/tc-15-1663-2021.

Chollet F and others (2015) Keras. https://github.com/fchollet/keras.

Cireşan DC, Meier U, Gambardella LM and Schmidhuber J (2010) Deep, big, simple neural nets for handwritten digit recognition. *Neural Computation* **22**(12), 3207–3220.

Clarke GK, Berthier E, Schoof CG and Jarosch AH (2009) Neural networks applied to estimating subglacial topography and glacier volume. *Journal of Climate* **22**(8), 2146–2160.

Cohen D, Gillet-Chaulet F, Haeberli W, Machguth H and Fischer UH (2018) Numerical reconstructions of the flow and basal conditions of the Rhine Glacier, European Central Alps, at the last glacial maximum. *The Cryosphere* **12**(8), 2515–2544.

Cuffey K and Paterson W (2010) *The Physics of Glaciers*. Amsterdam: Academic Press.

Dada EG and 5 others (2019) Machine learning for email spam filtering: review, approaches and open research problems. *Heliyon* **5**(6), e01802.

Egholm DL and 7 others (2017) Formation of plateau landscapes on glaciated continental margins. *Nature Geoscience* **10**(8), 592–597.

Fick SE and Hijmans RJ (2017) Worldclim 2: new 1-km spatial resolution climate surfaces for global land areas. *International Journal of Climatology* **37**(12), 4302–4315.

Gagliardini O and 14 others (2013) Capabilities and performance of Elmer/Ice, a new-generation ice sheet model. *Geoscientific Model Development* **6**(4), 1299–1318.

Glen JW (1953) Rate of flow of polycrystalline ice. *Nature* **172**, 721–722.

Greve R and Blatter H (2009) *Dynamics of Ice Sheets and Glaciers*. Berlin: Springer Verlag.

Gudmundsson GH (1999) A three-dimensional numerical model of the confluence area of unteraargletscher, Bernese Alps, Switzerland. *Journal of Glaciology* **45**(150), 219–230.

Hock R (2003) Temperature index melt modelling in mountain areas. *Journal of hydrology* **282**(1–4), 104–115.

Hutter K (1983) *Theoretical Glaciology*. Tokyo: Terra Scientific Publishing Company, D. Reidel Publishing Company.

Isola P, Zhu JY, Zhou T and Efros AA (2017) Image-to-image translation with conditional adversarial networks. *CVPR*, 1125–1134.

Jouvet G, Huss M, Blatter H, Picasso M and Rappaz J (2009) Numerical simulation of rhonegletscher from 1874 to 2100. *Journal of Computational Physics* **228**, 6426–6439. doi: 10.1016/j.jcp.2009.05.033

Jouvet G, Picasso M, Rappaz J and Blatter H (2008) A new algorithm to simulate the dynamics of a glacier: theory and applications. *Journal of Glaciology* **54**(188), 801–811. doi: 10.3189/002214308787780049

Jouvet G and Rappaz J (2011) Analysis and finite element approximation of a nonlinear stationary stokes problem arising in glaciology. *Advances in Numerical Analysis* **2011**, 1–24.

Khroulev C and the PISM Authors (2020) PISM, a Parallel Ice Sheet Model v1.2: User's Manual.

Kim B, Azevedo VC, Thuerey N, Kim T, Gross M and Solenthaler B (2019) Deep fluids: a generative network for parameterized fluid simulations. *Computer Graphics Forum* **38**(2), 59–70. doi: 10.1111/cgf.13619.

Kingma DP and Ba JL (2015) Adam: a method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015 – Conference Track Proceedings*, San Diego, vol. 5.

Ladický L, Jeong S, Solenthaler B, Pollefeys M and Gross M (2015) Data-driven fluid simulations using regression forests. *ACM Transactions on Graphics* **34**(6), 1–9, ISSN 15577368. doi: 10.1145/2816795.2818129

LeCun Y, Bengio Y and Hinton G (2015) Deep learning. *Nature* **521**(7553), 436–444.

Leong WJ and Horgan HJ (2020) DeepBedMap: a deep neural network for resolving the bed topography of Antarctica. *The Cryosphere* **14**(11), 3687–3705. doi: 10.5194/tc-14-3687-2020.

Lipscomb WH and 14 others (2019) Description and evaluation of the community ice sheet model (cism) v2. 1. *Geoscientific Model Development* **12**(1), 387–424.

Long J, Shelhamer E and Darrell T (2015) Fully convolutional networks for semantic segmentation. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3431–3440.

Maas AL, Hannun AY and Ng AY (2013) Rectifier nonlinearities improve neural network acoustic models. *Proc. ICML*, vol. 30, p. 3.

MacAyeal DR (1997) EISMINT: Lessons in ice-sheet modeling. *Department of Geophysical Sciences, University of Chicago, Chicago, IL* **1832**, 1839.

Maussion F and 14 others (2019) The open global glacier model (oggm) v1. 1. *Geoscientific Model Development* **12**(3), 909–931.

Mohajerani Y, Wood M, Velicogna I and Rignot E (2019) Detection of glacier calving margins with convolutional neural networks: a case study. *Remote Sensing* **11**(1), 74.

Monnier J and Zhu J (2021) Physically-constrained data-driven inversions to infer the bed topography beneath glaciers flows. Application to East Antarctica. *Computational Geosciences* **25**(5), 1793–1819.

Obiols-Sales O, Vishnu A, Malaya N and Chandramowlishwaran A (2020) Cfdnet: a deep learning-based accelerator for fluid simulations. arXiv:2005.04485.

Pattyn F (2018) The paradigm shift in antarctic ice sheet modelling. *Nature Communications* **9**(1), 1–3.

Pattyn F and 18 others (2012) Results of the marine ice sheet model intercomparison project, mismip. *The Cryosphere* **6**(3), 573–588.

Raschka S and Mirjalili V (2017) *Python Machine Learning*. Birmingham: Packt Publishing Ltd.

Räss L, Licul A, Herman F, Podladchikov YY and Suckale J (2020) Modelling thermomechanical ice deformation using an implicit pseudo-transient method (FastICE v1. 0) based on graphical processing units (GPUs). *Geoscientific Model Development* **13**(3), 955–976.

Reichstein M and 6 others (2019) Deep learning and process understanding for data-driven Earth system science. *Nature* **566**(7743), 195–204. doi: 10.1038/s41586-019-0912-1

Riel B, Minchew B and Bischoff T (2021) Data-driven inference of the mechanics of slip along glacier beds using physics-informed neural networks: case study on Rutford ice stream, Antarctica. *Journal of Advances in Modeling Earth Systems* e2021MS002621.

Ronneberger O, Fischer P and Brox T (2015) U-net: convolutional networks for biomedical image segmentation. *International Conference on Medical image computing and computer-assisted intervention*. Springer, pp. 234–241.

Seddik H, Greve R, Zwinger T, Gillet-Chaulet F and Gagliardini O (2012) Simulations of the Greenland Ice Sheet 100 years into the future with the full Stokes model Elmer/Ice. *Journal of Glaciology* **58**(209), 427–440.

Seguinot J and 5 others (2018) Modelling last glacial cycle ice dynamics in the Alps. *The Cryosphere* **12**(10), 3265–3285.

Simonyan K and Zisserman A (2015) Very deep convolutional networks for large-scale image recognition. *International Conference on Learning Representations.*

Sutter J and 6 others (2019) Modelling the Antarctic Ice Sheet across the mid-Pleistocene transition – implications for oldest ice. *The Cryosphere* **13**(7), 2023–2041.

Thuerey N, Weißenow K, Prantl L and Hu X (2020) Deep learning methods for Reynolds-averaged Navier–Stokes simulations of airfoil flows. *AIAA Journal* **58**(1), 25–36.

Tompson J, Schlachter K, Sprechmann P and Perlin K (2019) Accelerating Eulerian fluid simulation with convolutional networks. *5th International Conference on Learning Representations, ICLR 2017 – Workshop Track Proceedings*, vol. 70, pp. 3424–3433.

Višnjević V, Herman F and Prasicek G (2020) Climatic patterns over the European Alps during the LGM derived from inversion of the paleo-ice extent. *Earth and Planetary Science Letters* **538**, 116185.

Weertman J (1957) On the sliding of glaciers. *Journal of Glaciology* **3**(21), 33–38. doi: 10.3189/S0022143000024709

Werder MA, Hewitt IJ, Schoof CG and Flowers GE (2013) Modeling channelized and distributed subglacial drainage in two dimensions. *Journal of Geophysical Research: Earth Surface* **118**(4), 2140–2158.

Yang L, Zhang Y, Chen J, Zhang S and Chen DZ (2017) Suggestive annotation: a deep active learning framework for biomedical image segmentation. *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer, pp. 399–407.

Zhang E, Liu L and Huang L (2019) Automatically delineating the calving front of Jakobshavn Isbræ from multitemporal TerraSAR-X images: a deep learning approach. *The Cryosphere* **13**(6), 1729–1741.

Ziemen FA and 6 others (2016) Modeling the evolution of the Juneau icefield between 1971 and 2100 using the parallel ice sheet model (PISM). *Journal of Glaciology* **62**(231), 199–214.

Appendix A: Solving mass conservation

The mass conservation equation (1) is solved using a first-order upwind finite-volume scheme similar as the one described in Lipscomb and others (2019). Here we use a staggered grid, i.e. the ice thickness is defined at the centre of each grid cell, but the ice flow velocities used in the scheme are defined at the middle of the grid edges by a simple averaging. The key advantage of using such a staggered grid is that solving Eqn. (1) by finite volumes becomes natural as mass of ice is allowed to move from cell to cell (where the thickness is defined) from edge-defined fluxes (inferred from depth-average velocities). The transport of mass is governed following an upwind scheme for stability reasons. The resulting scheme is fully explicit (and therefore runs well in parallel), however, subject to CFL condition (2). Further details about this scheme can be found in Lipscomb and others (2019, Section 5.5.3).

Appendix B: Icefield simulations with PISM

To generate icefield simulation data, we selected five 1024 km × 1024 km tiles of existing topographically diverse mountainous environments worldwide (Table 5 and Fig. 5). For each tile, we extracted the publicly available NASA Shuttle Radar Topographic Mission (SRTM, http://srtm.csi.cgiar.org/) Digital Elevation Model (DEM), resampled at 2 km resolution to be used as basal topography in PISM. Details about the ice flow and mass-balance models used in PISM are now described in turn:

(1) Basal sliding was modelled with law (6) with parameter c = 70 km MPa^{-3} a^{-1}, which was chosen so that a basal shear stress of 80 kPa corresponds to a sliding velocity of ≈35 m a^{-1} (Pattyn and others, 2012).

(2) Surface mass balance (difference between accumulation and ablation) is calculated from the monthly mean surface air temperature, monthly precipitation and daily variability of surface air temperature. Accumulation is equal to solid precipitation when the temperature is below 0°C, and decreases to zero linearly between 0 and 2°C. Ablation is computed proportionally to the number of positive degree days (Hock, 2003) with factors f_i = 8 mm K^{-1} d^{-1} w.e. for ice and f_s = 3 mm K^{-1} d^{-1} w.e. for snow, which are taken from the EISMINT intercomparison experiments for Greenland (MacAyeal, 1997).

As today's climate forcing, we took the monthly temperature and precipitation from the WorldClim dataset (Fick and Hijmans, 2017). Starting from ice-free conditions, we linearly decreased temperature at a rate of 1°C per century until a large icefield covering the tile was built. Then, the cooling was reversed symmetrically into warming at the same rate until the ice has completely disappeared.

Appendix C: Glacier simulations with CfsFlow

To generate glacier simulation data, we have picked 41 diverse existing valleys from the European Alps and from New Zealand with drainage basins ranging from 80 to 700 km^2. For each one, we have extracted the publicly available NASA Shuttle Radar Topographic Mission (SRTM, http://srtm.csi.cgiar.org/) Digital Elevation Model (DEM) to be used as bedrock, and built a 100 m resolution two-dimensional structured mesh of the DEM. A three-dimensional mesh was then vertically extruded with 100 of 10 m thick layers. In more detail, we use CfsFlow with the following specific settings:

(1) Basal sliding was modelled by (6) with constant coefficient c taken among {0, 6, 12, 25, 70} km MPa^{-3} a^{-1}.

(2) As mass balance, we use a simple parametrization based on given Equilibrium Line Altitude (ELA) z_{ELA}, accumulation β_{acc} and ablation β_{abl} vertical gradients, and maximum accumulation rates $a_{\max}$:

$$\mathrm{SMB}(z) = \begin{cases} \min\left(\beta_{\mathrm{acc}}(z - z_{\mathrm{ELA}}), a_{\max}\right), & \text{if } z \geq z_{\mathrm{ELA}} \\ \beta_{\mathrm{abl}}(z - z_{\mathrm{ELA}}), & \text{otherwise.} \end{cases}$$

with $\beta_{\mathrm{abl}} = 0.009$ a^{-1}, $\beta_{\mathrm{acc}} = 0.005$ a^{-1}, and $a_{\max} = 2$ m a^{-1}.

We initialized the model with ice-free conditions. For the first 100 years, the ELA was chosen constant with the 20% quantile value of the original glacier top surface elevation to allow for glacier advance. For the next and last 100 years, the ELA was raised linearly until it reached the 90% quantile value of the original glacier top surface elevation to allow for glacier retreat.

Appendix D: Learning curve

Figure 15 shows the learning curve, i.e. the evolution L_1 loss/misfit function against the number of epochs (an epoch corresponds to one pass over the entire training dataset), during the training of the two datasets. As a result, the convergence of the training was always found efficient and fairly smooth. The validation loss was always found below the training loss.

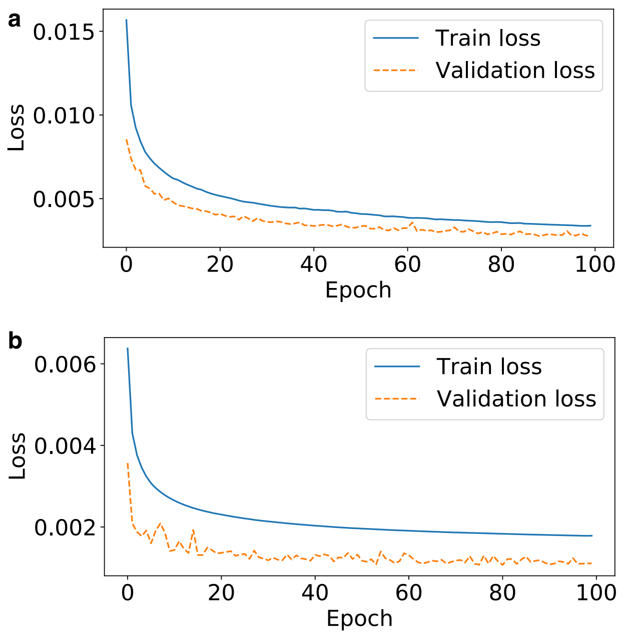

Fig. 15. Evolution of the train and the validation loss while training the ice flow emulator from the data generated with PISM (top) and CfsFlow (bottom).

Appendix E: Digest of artificial neural networks

Artificial neural networks approximate a mathematical function that maps input to output variables through a sequence of layers that contain nodes (or neurons). Mathematically, the output of each neuron is computed by some non-linear function (called activation) as the sum of its inputs. Neurons and connections have weights that are adjusted during the training (i.e. learning) stage. The optimal weights of the network are found by minimizing a certain cost function (often referred as loss function), which is most often defined as the misfit between the ground truth of the training data and the output of the network. As the number of weights of an effective neural network can be very high (typical $10^6 - 10^8$), the availability of a large data set is the key to determine the optimal weights and prevent against underdetermined systems. The optimization (or training) of the neural network often relies on a batch gradient descent method, which consists of sequentially adjusting weights by computing descent directions on small subsets (called batches) of the training data from the error between the model and data outputs. While the model evaluation happens sequentially from the first to the last layer (feed forward), the training happens sequentially in the opposite direction (back propagation) from the error computed at the last stage to the first input layer. In that case, the gradients required to optimize each weight are computed by the simple chain rule for derivatives. We refer to LeCun and others (2015) for an in-depth review on deep learning.