Journal of Computational Physics 537 (2025) 114093

Contents lists available at ScienceDirect

Journal of Computational Physics

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

Predicting nonlinear-flow regions in highly heterogeneous porous media using adaptive constitutive laws and neural networks

Chiara Giovannini [a,b,c,*], Alessio Fumagalli [a], Francesco Saverio Patacchini [b]

[a] MOX, Department of Mathematics, Politecnico di Milano, p.za Leonardo da Vinci 32, 20133, Milano, Italy
[b] IFP Energies nouvelles, 1 et 4 avenue de Bois-Préau, 92852, Rueil-Malmaison, France
[c] Department of Engineering Mechanics, KTH Royal Institute of Technology, Brinellvägen 8, 114 28, Stockholm, Sweden

ARTICLE INFO

Keywords:
Porous medium
Heterogeneous medium
Adaptive constitutive law
Neural network

ABSTRACT

In a porous medium featuring heterogeneous permeabilities, a wide range of fluid velocities may be recorded, so that significant inertial and frictional effects may arise in high-speed regions. In such parts, the link between pressure gradient and velocity is typically made via Darcy's law, which may fail to account for these effects; instead, the Darcy–Forchheimer law, which introduces a nonlinear term, may be more adequate. Applying the Darcy–Forchheimer law globally in the domain is very costly numerically and, rather, should only be done where strictly necessary. The question of finding a priori the subdomain where to restrict the use of the Darcy–Forchheimer law was recently answered in Fumagalli and Patacchini (2023) by using an adaptive model: given a threshold on the flow's velocity, the model locally selects the more appropriate law as it is being solved. At the end of the resolution, each mesh cell is flagged as being in the Darcy or Darcy–Forchheimer subdomain. Still, this model is nonlinear itself and thus relatively expensive to run. In this paper, to accelerate the subdivision of the domain into low- and high-speed regions, we instead exploit the adaptive model from Fumagalli and Patacchini (2023) to generate partitioning data given an array of different input parameters, such as boundary conditions and inertial coefficients, and then train neural networks on these data classifying each mesh cell as Darcy or not. Two test cases are studied to illustrate the results, where cost functions, parity plots, precision-recall plots and receiver operating characteristic curves are analyzed.

1. Introduction

When modeling fluid flow in highly heterogeneous porous media, characterized by irregularly distributed permeability profiles, such as they may arise when studying underground storage and management of CO_2, geothermal energy, nuclear waste or freshwater [2–5], Darcy's law, which is widespread in this context, may be challenged as noted in many studies [6–8]. Indeed, Darcy's law being a linear relation between fluid velocity (or, more precisely, seepage flux) and pressure gradient, it ignores inertial and frictional effects possibly arising in highly permeable areas, such as macropores, fractures or conduits, where high speeds may develop. Theoretical studies [9–13] provide admissible corrections in this particular setting, which consist in adding a nonlinear term to Darcy's law. The choice we make here is the so-called *Darcy–Forchheimer law* (cf. later for a generalization of it), which has the form

$$-\nabla p = \left(1 + c_{\mathrm{F}} \frac{\sqrt{k}}{\mu} \|\boldsymbol{u}\|\right) \frac{\nu}{k} \boldsymbol{u},$$

* Corresponding author.
E-mail address: chigio@kth.se (C. Giovannini).

https://doi.org/10.1016/j.jcp.2025.114093
Received 1 August 2024; Received in revised form 26 February 2025; Accepted 14 May 2025
Available online 17 May 2025

where $\boldsymbol{u}$ is the seepage flux, μ and ν are, respectively, the fluid's dynamic and kinematic viscosities, k is the medium's permeability and c_{F} is a dimensionless empirical inertial term which we refer to as the *Forchheimer coefficient*. The notation $\|\cdot\|$ stands for the Euclidean norm. Note that, for simplicity, we decide here to focus on scalar permeabilities, although tensor generalizations of this law are available [14].

The Darcy–Forchheimer law is more accurate than Darcy's law thanks to its nonlinear term, but it is also more costly to solve numerically because of this same nonlinearity. Therefore, it may not be reasonable resource-wise to solve it everywhere in the domain, but, rather, it may be better to select appropriately a partition of the domain into two subdomains, where in one subdomain Darcy's law is still solved thanks to a slow flow and in the other the Darcy–Forchheimer law is solved because of a fast regime that does not allow to neglect inertial and frictional terms. To implement this, we choose to determine the partition a priori and then, once it is known, to use domain-decomposition techniques [15–17] and thus obtain a numerical strategy which is faster than a global Darcy–Forchheimer approach but still more precise than the global Darcy counterpart. A posteriori strategies are also envisageable [18,19] but not investigated here. In this paper, we focus on the first step of this methodology, that is, the a priori calculation of the domain partition, the domain-decomposition step being left for future examination.

One possible way of achieving a priori knowledge of the Darcy and Darcy–Forchheimer regions may be found in the adaptive model presented in Fumagalli and Patacchini [1,20] and further tested numerically in Fumagalli and Patacchini [21]. There, the authors study a constitutive law that, given a fixed threshold on the magnitude of the seepage flux, automatically and iteratively selects Darcy's law or the Darcy–Forchheimer law: if the local flux has smaller magnitude than the threshold, Darcy's law is selected, otherwise the Darcy–Forchheimer law is. As a result of the simulation, one obtains a classification of each mesh cell as belonging to either the Darcy or the Darcy–Forchheimer subdomain. However, although the authors show that such a model is well posed, it is discontinuous at the transitions between the laws, making it rather unsuitable for numerical resolution via a classical, explicit fixed-point algorithm. Instead, in Fumagalli and Patacchini [1], they propose a smoothed version of this adaptive model, where, thanks to a convolution, the law is regularized around the transitions. As a result, this regularized model yields approximations of the sought-after partition but is better suited for numerical purposes. The regularized model is, nevertheless, nonlinear and thus still expensive to run.

To circumvent this, we propose here to combine the regularized model, already available from Fumagalli and Patacchini [1], with machine-learning techniques to provide a fast way of obtaining the partition. The regularized model is used to generate data with varying parameters, such as the Forchheimer coefficient and the boundary conditions and other physical parameters, and then neural networks are trained on these data. The networks output classifications of the mesh cells, by deciding for each cell whether it is Darcy or not. The validation of the networks is carried out by analyzing cost functions, parity plots, precision-recall plots and receiver operating characteristic curves. The codes for the regularized model and the neural networks are written in Python and available at Fumagalli and Patacchini [22], Giovannini [23], while the underlying discretization and equation resolution are performed using the open source Python simulation tool PorePy [24,25]. Note that, instead of restricting our neural networks to class outputs, we could directly design them to predict the complete and continuously valued velocity and pressure fields; this, however, would bring about the issue of mass conservation, which is not guaranteed in a neural-network output unless appropriately encoded [26] Our domain-identification approach circumvent this fundamental issue by relying on the resolution of a conservative model for the decomposition.

We start by giving the details of the physical framework and equations at stake and by deriving the threshold on the magnitude of the seepage flux (Section 2); we then describe our overall machine-learning methodology and the two test cases we apply it to, the first coming from Winter et al. [4] and applicable to landfill management and the second extracted from the SPE10 benchmark reservoir scenario [27] (Section 3); and, finally, we discuss the results on these test cases (Section 4). In Appendix A, we recall and discuss the basics of deep learning which we use in our methodology.

2. Physical framework

We position ourselves within the modeling context established by Fumagalli and Patacchini [1]; to ensure comprehensive understanding, we provide a summary here. To begin, let us elucidate some notations. We denote by $\Omega \subset \mathbb{R}^d$, $d \in \mathbb{N}$, the porous medium and assume it is open, bounded, and possesses a Lipschitz boundary $\partial\Omega$. The outward normal unit vector of $\partial\Omega$ is written $\boldsymbol{n}$. Additionally, we use $\mathbb{R}_+ = [0, \infty)$ and $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$.

The unknowns in the problems discussed throughout this paper are the fluid's pressure $p : \Omega \to \mathbb{R}$ and the seepage flux $\boldsymbol{u} : \Omega \to \mathbb{R}^d$. The seepage flux is defined by $\boldsymbol{u} = \rho\phi\boldsymbol{V}$, where ϕ is the medium's porosity, and ρ and $\boldsymbol{V}$ are the fluid's density and velocity, respectively. Furthermore, we use μ, ν and k to represent the fluid's dynamic viscosity and kinematic viscosity and the medium's scalar permeability, respectively. We assume that $\rho, \mu, \nu, k : \Omega \to (0, \infty)$ and $\phi : \Omega \to (0, 1)$ are space-dependent knowns of the problem.

2.1. Underlying model equations

In the porous material under consideration, we wish to solve the time-independent problem given by the conservations of mass and momentum. The former conservation is expressed as

$$\operatorname{div} \boldsymbol{u} = q \quad \text{in } \Omega; \tag{2.1}$$

here $q : \Omega \to \mathbb{R}$ is a known fluid mass source. For the latter conservation, we choose what is often called the *generalized Forchheimer (GF) law* [28–32]:

$$-\nabla p + \boldsymbol{f} = \left(1 + c_{\mathrm{F}}\left(\frac{\sqrt{k}}{\mu}\right)^{m} \|\boldsymbol{u}\|^{m}\right)\frac{\nu}{k}\,\boldsymbol{u}, \tag{2.2}$$

where $\boldsymbol{f} : \Omega \to \mathbb{R}^d$ is a known vector of external body forces, such as gravity, $c_{\mathrm{F}} : \Omega \to (0, \infty)$ is the Forchheimer coefficient (also referred to as the *Ergün number* in the literature [33,34], containing information on the porosity and tortuosity of the medium [11]) and $m \in [1, \infty)$ may be called the *Forchheimer exponent*. When $m = 1$, we recover the Darcy–Forchheimer (DF) law, as already discussed; qualitatively, Darcy's linear law is retrieved by imposing $m = 0$ or $c_{\mathrm{F}} \equiv 0$. Here, $\|\cdot\|$ stands for the Euclidean norm in $\mathbb{R}^d$. The conservation of momentum is also referred to as the *constitutive law* or *seepage law*.

With this pair of equations, we associate the following boundary conditions:

$$\begin{cases} \boldsymbol{u} \cdot \boldsymbol{n} = u_0 & \text{on } \Sigma_{\mathrm{v}}, \\ p = p_0 & \text{on } \Sigma_{\mathrm{p}}, \end{cases} \tag{2.3}$$

where $\Sigma_{\mathrm{v}}, \Sigma_{\mathrm{p}} \subset \partial\Omega$ are relatively open in $\partial\Omega$ (i.e., Σ_{v} and Σ_{p} are each the intersection of an open subset of $\mathbb{R}^d$ with $\partial\Omega$) and satisfy $\partial\Omega = \overline{\Sigma_{\mathrm{v}} \cup \Sigma_{\mathrm{p}}}$ and $\Sigma_{\mathrm{v}} \cap \Sigma_{\mathrm{p}} = \emptyset$. Here, $u_0 : \Sigma_{\mathrm{v}} \to \mathbb{R}$ and $p_0 : \Sigma_{\mathrm{p}} \to \mathbb{R}$ are functions setting the conditions on the boundary for the flux and pressure.

Eqs. (2.1)–(2.3) form our model of interest. The underlying theoretical framework, including the appropriate functional spaces and weak formulations, are described in detail in Fumagalli and Patacchini [1,20].

2.2. Adaptive model

We can discern two distinct components within the GF law in (2.2): a linear component given by $(\nu/k)\boldsymbol{u}$ and a nonlinear one represented by $c_{\mathrm{F}}(k/\mu)^m\|\mathbf{u}\|^m(\nu/k)\boldsymbol{u}$. When the velocity, or seepage flux, is relatively low, addressing the nonlinear part becomes unnecessary and may be safely disregarded; this amounts to the ubiquitous Darcy law. Conversely, when the velocity is relatively high, the nonlinear part should be taken into account if one wants to study the inertial and frictional aspects of the flow; in this case, $m \geq 1$ and, as already mentioned, the particularly relevant case $m = 1$ yields the well known DF law.

In practice, one could always treat the nonlinear component since it adds precision to Darcy's law in any range of velocities. However, because its benefits are extremely limited in low-speed regions, i.e., in most part, of the medium, the gain in precision in these parts of the domain does not outweigh the numerical cost brought by the nonlinearity. It is thus reasonable only to solve the nonlinear model where strictly necessary, that is, where the flux magnitude is above a certain threshold and then keep Darcy's model where the magnitude is below this same threshold. The derivation of this a priori threshold is given in Section 2.3 below.

Denoting by $\bar{u}$ the threshold, the dichotomy arising from this approach can be formulated as

$$-\nabla p + \boldsymbol{f} = \begin{cases} \frac{\nu}{k}\,\boldsymbol{u} & \text{if } \|\boldsymbol{u}\| < \bar{u}, \\ \left(1 + c_{\mathrm{F}}\left(\frac{\sqrt{k}}{\mu}\right)^{m} \|\boldsymbol{u}\|^{m}\right)\frac{\nu}{k}\,\boldsymbol{u} & \text{if } \|\boldsymbol{u}\| > \bar{u}. \end{cases} \tag{2.4}$$

We refer to this law as the *adaptive* law, which, combined with (2.1) and (2.2) constitutes the adaptive model. The terminology stems from the fact that this model adapts naturally the constitutive law considered depending on the local Euclidean norm of the flux magnitude. What happens at the transition set $\{\|\boldsymbol{u}\| = \bar{u}\}$ is a delicate matter which we do not delve into here. Indeed, it requires the introduction of a multivalued reformulation of the adaptive law which does not enter the qualitative scope of the present discussion; we refer the reader to Fumagalli and Patacchini [1] for the complete details of the theory. The well-posedness of such an adaptive model is obtained by a straightforward generalization of [1, Assumption 4.1 and Corollary 4.4] to space-dependent physical parameters μ, ν, k and c_{F}, which we further assume to be bounded below and above by strictly positive constants in Ω.

Note that the adaptive law (2.4) can be rewritten as

$$-\nabla p + \boldsymbol{f} = \left(\mathbb{1}_{\{\|\boldsymbol{u}\| < \bar{u}\}} + c_{\mathrm{F}}\left(\frac{\sqrt{k}}{\mu}\right)^{m} \|\boldsymbol{u}\|^{m}\mathbb{1}_{\{\|\boldsymbol{u}\| > \bar{u}\}}\right)\frac{\nu}{k}\,\boldsymbol{u}, \tag{2.5}$$

where $\mathbb{1}_A$ stands for the indicator function of set A and $\{\|\boldsymbol{u}\| < \bar{u}\}$ (respectively, $\{\|\boldsymbol{u}\| > \bar{u}\}$) is a short-hand notation for $\{\boldsymbol{x} \in \Omega | \|\boldsymbol{u}(\boldsymbol{x})\| < \bar{u}\}$ (respectively, $\{\boldsymbol{x} \in \Omega | \|\boldsymbol{u}(\boldsymbol{x})\| > \bar{u}\}$).

2.3. Flux threshold

We now explain how to choose a priori the flux threshold, and then use this opportunity to define formally the Darcy and GF subdomains and introduce the Reynolds and Forchheimer numbers, which we shall use later. These notions are discussed in details for even more general seepage laws in Fumagalli and Patacchini [21].

2.3.1. Threshold derivation

Given a flux $\boldsymbol{u}$, the forces $-\boldsymbol{\nabla} p_{\mathrm{D}} + \boldsymbol{f}$ and $-\boldsymbol{\nabla} p_{\mathrm{GF}} + \boldsymbol{f}$ resulting from the Darcy and GF laws are, respectively, deducted from the following constitutive laws:

$$\begin{cases} -\boldsymbol{\nabla} p_{\mathrm{D}} + \boldsymbol{f} = \frac{\nu}{k}\,\boldsymbol{u}, \\ -\boldsymbol{\nabla} p_{\mathrm{GF}} + \boldsymbol{f} = \left(1 + c_{\mathrm{F}}\left(\frac{\sqrt{k}}{\mu}\right)^{m} \|\boldsymbol{u}\|^{m}\right)\frac{\nu}{k}\,\boldsymbol{u}. \end{cases} \tag{2.6}$$

Subtracting the equations in (2.6) and using the second one as reference, we arrive at the following definition of *local error* $d[\boldsymbol{u}] : \Omega \to [0,1)$, which can be found in Zeng and Grigg [35], Macini et al. [36], Winter et al. [4] for the case $m = 1$:

$$d[\boldsymbol{u}] = \frac{c_{\mathrm{F}}\left(\frac{\sqrt{k}}{\mu}\right)^{m} \|\boldsymbol{u}\|^{m}}{1 + c_{\mathrm{F}}\left(\frac{\sqrt{k}}{\mu}\right)^{m} \|\boldsymbol{u}\|^{m}}. \tag{2.7}$$

Note that there exists a function $\Delta : \Omega \times \mathbb{R}_{+} \to [0,1)$ such that, for all $\boldsymbol{x} \in \Omega$, there holds

$$d[\boldsymbol{u}](\boldsymbol{x}) = \Delta(\boldsymbol{x}, \|\boldsymbol{u}(\boldsymbol{x})\|), \tag{2.8}$$

and $\Delta(\boldsymbol{x}, \cdot)$ is an increasing bijection. Then, let $\delta \in [0,1)$ and deduce that there is a unique $w : \Omega \to \mathbb{R}_{+}$ so that $\Delta(\boldsymbol{x}, w(\boldsymbol{x})) = \delta$ for all $\boldsymbol{x} \in \Omega$. We can then define the flux threshold $\bar{u}$:

$$\bar{u} = \inf_{\Omega} w \in \mathbb{R}_{+}.$$

Then, using (2.7) and (2.8) and inverting Δ, one finds

$$w = \left(\frac{\delta}{1-\delta}\right)^{1/m} \frac{\mu}{c_{\mathrm{F}}^{1/m}\sqrt{k}}, \tag{2.9}$$

which yields

$$\bar{u} = \left(\frac{\delta}{1-\delta}\right)^{1/m} \inf_{\Omega} \frac{\mu}{c_{\mathrm{F}}^{1/m}\sqrt{k}}. \tag{2.10}$$

In particular, when $m = 1$, i.e., the reference law is DF, we get

$$\bar{u} = \frac{\delta}{1-\delta} \inf_{\Omega} \frac{\mu}{c_{\mathrm{F}}\sqrt{k}}.$$

2.3.2. Subdomains

Given again a flux $\boldsymbol{u}$, an error tolerance $\delta \in [0,1)$ and the corresponding flux threshold $\bar{u}$, we define the Darcy and GF subdomains as

$$\Omega_{\mathrm{D}}[\boldsymbol{u}] = \{\boldsymbol{x} \in \Omega \,|\, \|\boldsymbol{u}(\boldsymbol{x})\| < \bar{u}\} \quad \text{and} \quad \Omega_{\mathrm{GF}}[\boldsymbol{u}] = \{\boldsymbol{x} \in \Omega \,|\, \|\boldsymbol{u}(\boldsymbol{x})\| > \bar{u}\},$$

which we may respectively refer to as the *slow* and *fast* subdomains.

If $\boldsymbol{x} \in \Omega_{\mathrm{D}}[\boldsymbol{u}]$, we know that solving at $\boldsymbol{x}$ Darcy's law (cf. equation in (2.6)) does not generate a local error greater than δ relative to solving at $\boldsymbol{x}$ the GF law (cf.d equation). Indeed, suppose $\boldsymbol{x} \in \Omega_{\mathrm{D}}[\boldsymbol{u}]$; then, recalling (2.8), (2.9) and (2.10), we get

$$\|\boldsymbol{u}(\boldsymbol{x})\| < \bar{u} \leqslant w(\boldsymbol{x}),$$

so that

$$d[\boldsymbol{u}](\boldsymbol{x}) = \Delta(\boldsymbol{x}, \|\boldsymbol{u}(\boldsymbol{x})\|) < \Delta(\boldsymbol{x}, w(\boldsymbol{x})) = \delta.$$

Hence, by solving Darcy's law in $\Omega_{\delta}[\boldsymbol{u}]$ and the GF law in $\Omega_{\mathrm{GF}}[\boldsymbol{u}]$, we make sure that the local error stays everywhere below δ.

2.3.3. Reynolds and Forchheimer numbers

Given a flux $\boldsymbol{u}$, we define the *Reynolds number* $\mathrm{Re}[\boldsymbol{u}] : \Omega \to \mathbb{R}_{+}$ by

$$\mathrm{Re}[\boldsymbol{u}] = \frac{\sqrt{k}}{\mu}\|\boldsymbol{u}\|.$$

Thanks to this notation, together with the notions of subdomains given in the previous section, given an error tolerance, we may rewrite the adaptive law (2.5) as

$$-\boldsymbol{\nabla} p + \boldsymbol{f} = \left(\mathbb{1}_{\Omega_{\mathrm{D}}[\boldsymbol{u}]} + c_{\mathrm{F}}\mathrm{Re}[\boldsymbol{u}]^{m}\mathbb{1}_{\Omega_{\mathrm{GF}}[\boldsymbol{u}]}\right)\frac{\nu}{k}\,\boldsymbol{u}. \tag{2.11}$$

We also define the *mth Forchheimer number* $\mathrm{Fo}_{m}[\boldsymbol{u}] : \Omega \to \mathbb{R}_{+}$ by

$$\mathrm{Fo}_{m}[\boldsymbol{u}] = c_{\mathrm{F}}^{1/m}\frac{\sqrt{k}}{\mu}\|\boldsymbol{u}\| = c_{\mathrm{F}}^{1/m}\mathrm{Re}[\boldsymbol{u}].$$

The dimensionless number $\mathrm{Fo}[\boldsymbol{u}]^m$ quantifies the weight of the nonlinear term in the GF law relative to 1, i.e.tive to the linear term. The first Forchheimer number, with $m = 1$, resulting from considering the DF law as reference, is simply referred to as the Forchheimer number in the literature [4,35,36] and denoted by $\mathrm{Fo}[\boldsymbol{u}]$. Given an error tolerance $\delta \in [0, 1)$, one can further rewrite (2.11) as

$$-\boldsymbol{\nabla} p + \boldsymbol{f} = \left(\mathbb{1}_{\Omega_{\mathrm{D}}[\boldsymbol{u}]} + \mathrm{Fo}_m[\boldsymbol{u}]^m \mathbb{1}_{\Omega_{\mathrm{GF}}[\boldsymbol{u}]} \right) \frac{\nu}{k} \boldsymbol{u}. \tag{2.12}$$

Moreover, the local error in (2.7) can be reformulated as

$$d[\boldsymbol{u}] = \frac{\mathrm{Fo}_m[\boldsymbol{u}]^m}{1 + \mathrm{Fo}_m[\boldsymbol{u}]^m}.$$

Also note that if at $\boldsymbol{x} \in \Omega$ we have $\mathrm{Fo}_m[\boldsymbol{u}](\boldsymbol{x}) > (\delta/(1-\delta))^{1/m}$, then $\boldsymbol{x} \in \Omega_{\mathrm{DF}}[\boldsymbol{u}]$; thus, the value $(\delta/(1-\delta))^{1/m}$ may be seen as a critical Forchheimer number above which nonlinear effects are surely nonnegligeable.

2.4. Regularized model

The adaptive law, found in its different forms in (2.4), (2.5), (2.11) and (2.12), is discontinuous in the flux variable. Indeed, across the transition zone $\{\boldsymbol{x} \in \Omega | \|\boldsymbol{u}(\boldsymbol{x})\| = \bar{u}\}$, the law jumps from Darcy's to the GF law. This discontinuity is not well suited for numerical resolution, especially when using a fixed-point algorithm. To circumvent this, the authors in Fumagalli and Patacchini [1] propose to use the variational, or energetic, formulation of the adaptive model to regularize the law via a convolution of the underlying energy functional. For completeness, we describe the main idea below, although we refer the reader to Fumagalli and Patacchini [1] for the details and the rigorous framework.

As just mentioned, the regularization of the adaptive law is based on an energetic consideration. Indeed, it is shown there that a pair $(\boldsymbol{u}, p)$ is solution to the adaptive model (2.1)–(2.3)–(2.12) if and only if it is a saddle point of the energy functional $\mathcal{E}$ defined by

$$\mathcal{E}(\boldsymbol{\varphi}, \psi) = \int_\Omega \boldsymbol{f} \cdot \boldsymbol{\varphi} - \int_\Omega q\psi + \int_{\Sigma_{\mathrm{v}}} u_0 \psi - \int_\Omega \boldsymbol{\nabla}\psi \cdot \boldsymbol{\varphi} - \mathcal{D}(\boldsymbol{\varphi}),$$

where

$$\mathcal{D}(\boldsymbol{\varphi}) = \frac{1}{2} \int_\Omega \left(\mathbb{1}_{\Omega_{\mathrm{D}}[\boldsymbol{\varphi}]} + \frac{2}{m+2} \mathrm{Fo}_m[\boldsymbol{\varphi}]^m \mathbb{1}_{\Omega_{\mathrm{GF}}[\boldsymbol{\varphi}]} \right) \frac{\nu}{k} \|\boldsymbol{\varphi}\|^2.$$

The term D is called the *dissipation* and is minimized on a certain restricted function space by a certain component of the solution $\boldsymbol{u}$ (cf.tFP21,FP23). The discontinuity of the adaptive law is reflected in the discontinuity of the integrand of D with respect to the variable $\|\boldsymbol{\varphi}\|$. In fact, $\mathcal{D}$ is not differentiable and merely has a subdifferential, which is equal to the right-hand side of the adaptive law in (2.12). To make the dissipation smooth, its integrand is convolved with a Gaussian kernel G_ε of mean 0 and standard deviation some fixed $\varepsilon > 0$. The convolution is applied on the space of flux magnitudes and yields the regularized dissipation $\mathcal{D}_\varepsilon$, defined as

$$\mathcal{D}_\varepsilon(\boldsymbol{\varphi}) = \frac{1}{2} \int_\Omega G_\varepsilon * \left[\left(\mathbb{1}_{\Omega_{\mathrm{D}}[\boldsymbol{\varphi}]} + \frac{2}{m+2} \mathrm{Fo}_m[\boldsymbol{\varphi}]^m \mathbb{1}_{\Omega_{\mathrm{GF}}[\boldsymbol{\varphi}]} \right) \frac{\nu}{k} \|\boldsymbol{\varphi}\|^2 \right],$$

and the corresponding energy $\mathcal{E}_\varepsilon$, given by

$$\mathcal{E}_\varepsilon(\boldsymbol{\varphi}, \psi) = \int_\Omega \boldsymbol{f} \cdot \boldsymbol{\varphi} - \int_\Omega q\psi + \int_{\Sigma_{\mathrm{v}}} u_0 \psi - \int_\Omega \boldsymbol{\nabla}\psi \cdot \boldsymbol{\varphi} - \mathcal{D}_\varepsilon(\boldsymbol{\varphi}).$$

The regularized model then consists in finding a pair $(\boldsymbol{u}_\varepsilon, p_\varepsilon)$ which is a saddle point of $\mathcal{E}_\varepsilon$. Equivalently, after differentiation of the regularized dissipation, it consists in solving the model given by (2.1) and (2.3) together with the following law:

$$-\boldsymbol{\nabla} p_\varepsilon + \boldsymbol{f} = G'_\varepsilon * \left[\left(\mathbb{1}_{\Omega_{\mathrm{D}}[\boldsymbol{u}_\varepsilon]} + \frac{2}{m+2} \mathrm{Fo}_m[\boldsymbol{u}_\varepsilon]^m \mathbb{1}_{\Omega_{\mathrm{GF}}[\boldsymbol{u}_\varepsilon]} \right) \frac{\nu}{k} \|\boldsymbol{u}_\varepsilon\|^2 \right] \boldsymbol{u}_\varepsilon, \tag{2.13}$$

where G'_ε is the derivative of the Gaussian kernel G_ε. The smooth model (2.1)–(2.3)–(2.13) is well posed thanks to [1, Corollary 5.3] and we decide to use Raviart–Thomas finite elements and an explicit fixed-point algorithm to generate data on which to train our neural networks (cf.ref[Sections]3 and 4).

Of course, when solving the regularized model as opposed to the adaptive one, we obtain a different partition into slow and fast subdomains, so that $\Omega_{\mathrm{D}}[\boldsymbol{u}]$ differs from $\Omega_{\mathrm{D}}[\boldsymbol{u}_\varepsilon]$ and so does $\Omega_{\mathrm{GF}}[\boldsymbol{u}]$ from $\Omega_{\mathrm{GF}}[\boldsymbol{u}_\varepsilon]$. Nevertheless, thanks to the weak convergence result [1, Theorem 5.4], we expect that the regularized subdomains are not far from those stemming from the adaptive model for a reasonably small choice of ε, such as $\varepsilon = 0.1$, which is the value we take for the numerical experiments below.

Remark 2.1. Proving the abovementioned subdomain convergence is delicate and out of the scope of this paper. Still, for completeness, let us give a hint on how we may proceed. If the extra condition

$$\int_\Omega \mathbb{1}_{\Omega_{\mathrm{D}}[\boldsymbol{u}_\varepsilon]} g \to \int_\Omega \mathbb{1}_{\Omega_{\mathrm{D}}[\boldsymbol{u}]} g, \quad \text{as } \varepsilon \to 0 \text{ for all } g \in L^s(\Omega), \tag{2.14}$$

held, that is, if the indicator functions $(\mathbb{1}_{\Omega_{\mathrm{D}}[\boldsymbol{u}_\varepsilon]})_{\varepsilon>0}$ converged weakly to $\mathbb{1}_{\Omega_{\mathrm{D}}[\boldsymbol{u}]}$ in an appropriate Lebesgue space $L^s(\Omega)$, $1 \le s < \infty$, then the result would follow from Camilli [37, Proposition 3.1] applied to our indicator functions; indeed, we would reach the conclusion that

$$\mathcal{L}^d(\Omega_{\mathrm{D}}[\boldsymbol{u}_\varepsilon] \Delta \Omega_{\mathrm{D}}[\boldsymbol{u}]) \to 0 \quad \text{as } \varepsilon \to 0,$$

where $\mathcal{L}^d$ stands for the d-dimensional Lebesgue measure and $A \Delta B := (A \setminus B) \cup (B \setminus A)$ is the set difference of sets A and B. The difficulty thus lies in showing that the condition (2.14) holds, which is untrue in a general weak-convergence setting but which one may be able to show knowing that $(\boldsymbol{u}_\varepsilon)_{\varepsilon>0}$ and $\boldsymbol{u}$ satisfy particular PDEs in our setting. One possible way would be to find conditions under which strong convergence actually holds.

3. Methodology and test cases

In our pursuit of an a prioritioning of the domain into Darcy and Generalized Forchheimer (GF) zones, we leverage neural networks to predict the categorization of cells within a given mesh. Specifically, we employ a binary classification scheme, where 0 (or positive) denotes a GF cell and 1 (or negative) a Darcy cell.

3.1. Overview of methodology

For details on the fundamental notions of deep learning used to set up our neural networks, tune hyperparameters and validate performance, we refer the reader to Appendix A. To summarize, for each test case (cf. Section 3.2), having chosen a mesh and a set of input features, such as boundary conditions, our methodology is based on generating a dataset using the codebase [22], which uses the regularized model described in Section 2 to obtain a domain partition into Darcy and GF regions, on training a dense feed-forward neural network with two or three hidden layers and cross-entropy as cost function, on using cross-validation and recall to tune the underlying hyperparameters, and on evaluating the train and test results with recall, precision and various performance plots as metrics. Fig. 1 provides a flowchart of this methodology, where the ordering of the above steps is specified, as well as the train-test split ratio, the number of cross-validation folds, and the value of the ε parameter from the regularized model.

3.2. Test cases

We apply the above methodology to two test cases, whose results are given in Section 4 below. Both are two-dimensional, the former coming from an application in landfill management [4], the second taken from Layer 35 of the SPE10 benchmark reservoir scenario [27]. We present here the way the data are generated for these test cases as well as the set of hyperparameters optimized.

3.2.1. Case 1: landfill management [4]

Landfills serve as critical repositories for waste disposal, necessitating efficient management strategies to mitigate environmental impact. Within this context, understanding the dynamics of rainwater infiltration is pivotal. In this study, we focus on the case where the influx is at the upper boundary acting like rainwater permeating soil. We impose zero-flux conditions on the left and right boundaries, and hydrostatic pressure at the bottom. In Fig. 2, the vertical axis is aligned with the gravitational force, and we notice the presence of highly permeable channels, or macropores, of which we study two possible configurations: one has two channels traversing the whole domain vertically, exactly as in Winter et al. [4], and the other features a network of seven intersecting channels.

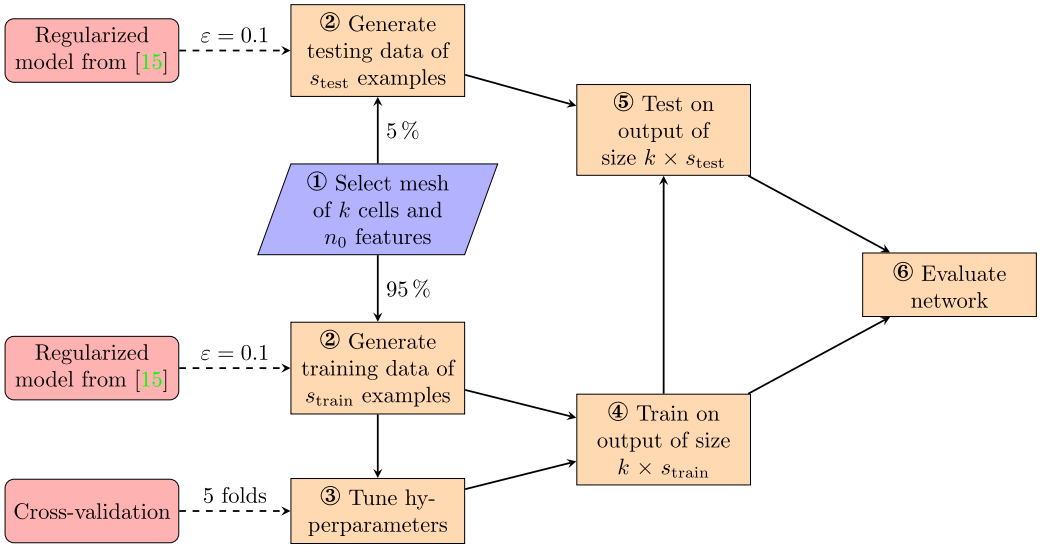
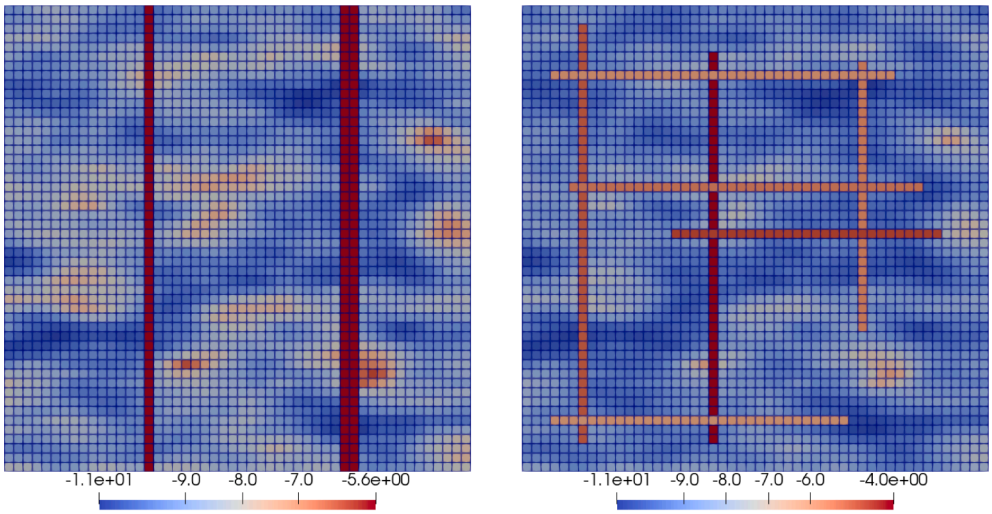


Fig. 1. Flowchart of used methodology, inspired by Muller and Guido [38].



(a) 2 channels. (b) 7 channels.

Fig. 2. Log-permeability in the landfill.

Data generation. In this configuration, our dataset comprises $s := s_{\text{train}} + s_{\text{test}} = 2352$ examples, each characterized by a 12-element input vector and a corresponding 2500-element output vector, $n_0 = 12$ being the number of input features and $k = 2500$ the mesh size (cf. Fig. 1 for the notation). The input vector includes the influx at the top boundary (u_0, in $\mathrm{kg\,m^{-2}\,s^{-1}}$; cf. Section 2.1), the Forchheimer coefficient (c_{F}, dimensionless; cf. Section 2.1), the Forchheimer exponent (m, dimensionless; cf. Section 2.1), the error tolerance (δ, dimensionless; cf. Section 2.3), and the number of channels in the domain (2 or 7) and their associated porosity values. To translate porosities into permeabilities, we use the Kozeny–Carman formula, given by

$$K = K_{\text{ref}} \frac{\phi^3 (1 - \phi_{\text{ref}})^2}{\phi_{\text{ref}}^3 (1 - \phi)^2},$$

where ϕ_{ref} and K_{ref} are respectively the reference porosity and reference permeability, which are calculated based on the grain-size distribution WT1 from the nuclear power plant Würgassen [4]. The calculation of these parameters is detailed in Winter et al. [4] and yields $\phi_{\text{ref}} = 0.35$ and $K_{\text{ref}} = 1.01 \times 10^{-9}\mathrm{m}^2$.

To generate diverse values for the input features, we employ specific probability distributions. For the influx, we use a normal distribution with mean, standard deviation and cut-off interval given by the following values:

$$\mu_{u_0} = 0.0105, \quad \sigma_{u_0} = 0.0035, \quad I_{u_0} = [0.001, 0.1].$$

Similarly, the Forchheimer coefficient follows a normal distribution with

$$\mu_{c_{\mathrm{F}}} = 0.55, \quad \sigma_{c_{\mathrm{F}}} = 0.4, \quad I_{c_{\mathrm{F}}} = [0.1, 0.9].$$

These ranges are inspired by values in Winter et al. [4]. The Forchheimer exponent is too generated from a normal distribution with

$$\mu_m = 1, \quad \sigma_m = 1.5, \quad I_m = [1, 4].$$

We restrict the interval to $m \geq 1$ which corresponds to the Darcy–Forchheimer seepage law (cf. Section 2.1). The error tolerance δ is selected from a uniform distribution within the interval $[0.01, 0.25]$.

Seven values are randomly selected for each parameter. These values are then clipped, resulting in six distinct c_{F}, seven δ, seven u_0, and four m. Subsequently, we compute the product with every possible combination of these values. The dataset is further diversified by using either 2 or 7 channels, effectively doubling its size, yielding the 2532 examples. For the porosity values in each channel, we generate a uniformly distributed vector of 7×1176 values ranging from 0.9 to 0.99; from this vector, we select either the first 2×1176 values for the two-channel case or the entire vector for the seven-channel case.

By adopting these distributions, our aim is to align the dataset with the patterns and insights highlighted in Winter et al. [4] and ensure a meaningful and contextually relevant dataset for experimentation. Using these values, we input data into the code from Fumagalli and Patacchini [22], Giovannini [23], generating a 2500-cell output vector for each input vector, i.e., an output of size $k \times s$, representing the Darcy and GF binary mesh values.

Hyperparameters. Using cross-validation with 5 folds based on recall as metric, we tune the learning rate in the range $[0.001, 0.1]$ and the number of nodes per hidden layer across the two networks given in Table 1.

Table 1
The layer structure of the two networks used for Case 1.

Network	Layers and #nodes
1	[12, 256, 512, 2500]
2	[12, 512, 256, 2500]

3.2.2. Case 2: layer 35 of SPE10 [27]

Layer 35 of the SPE10 model plays a key role in reservoir simulation due to the intricate interplay between injection and production dynamics, and due to the highly permeable paths it features. In the context of our investigation, this horizontal layer is characterized by a single injection well through which water is pumped at the center of the domain, along with four production wells at the corners (cf. Fig. 3). From the known depth of Layer 35, namely, 21.336 meters, we make the assumption of hydrostatic pressure on the entire boundary for this particular scenario, although no-flux boundary conditions could also be considered.

Data generation. Here, our dataset encompasses $s = 3888$ examples, each characterized by a 4-element input vector and a corresponding 13200-element output vector, where $n_0 = 4$ is the number of input features and $k = 13200$ is the mesh size (cf. Fig. 1). Similar to the preceding case, the input vector incorporates the Forchheimer coefficient, the Forchheimer exponent and the error tolerance. Additionally, the parameter Q (measured in $\mathrm{kg\,s^{-1}}$) is introduced, corresponding to an external source (cf. Section 2.1) and used to assign values to distinct injection and production wells within the mesh. Common values are retained for the shared parameters across cases; for Q, a normal distribution is employed, characterized by

$$\mu_Q = 100, \quad \sigma_Q = 30, \quad I_Q = [10, 300].$$

Nine values are randomly selected for each parameter. These values are then clipped, resulting in eight distinct c_{F}, nine δ, nine Q, and six m. Subsequently, we compute every possible combination of these values, reaching the 3888 examples.

Hyperparameters. Using again cross-validation with 5 folds and recall as performance metric, we tune the learning rate in the range $[0.001, 0.1]$ and the number of hidden layers and nodes across the two networks given in Table 2.

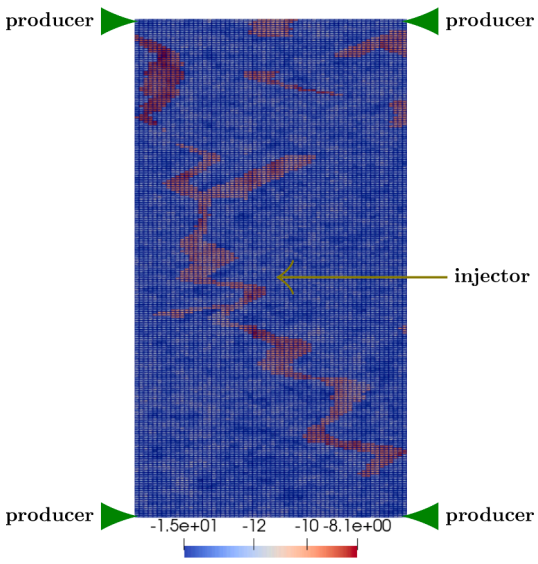


Fig. 3. Log-permeability in Layer 35 of SPE10.

Table 2
The layer structure of the two networks used for Case 2.

Network	Layers and #nodes
1	[4, 256, 512, 13200]
2	[4, 512, 1024, 2048, 13200]

4. Results

We discuss now the results for the two test cases described in Section 3.

4.1. Case 1: Landfill management [4]

4.1.1. Hyperparameter tuning

Among the networks listed in Table 3, we highlight in yellow the one with highest recall, which is considered our main metric of interest (cf. Appendix A.4.1). In particular, the table illustrates the argument from Appendix A.4.1, namely, that the error rate has little discriminative power in our context: with the exception of networks employing a learning rate of 0.1, the variations in error rate between different networks are almost negligible.

The percentage difference between the training cost and the validation cost (%cost) is defined as

$$\%\text{cost} = \frac{|c_{\text{val}} - c_{\text{train}}|}{c_{\text{train}}},$$

where c_{val} and c_{train} are the costs of the validation and training, respectively. This metric measures the potential for overfitting and underfitting, and, despite already using cross-validation to mitigate overfitting, it provides an additional layer of control. Elevated values of %cost suggest overfitting or underfitting tendencies. For instance, in the first row of the second part of Table 3, a notable disparity in %cost coincides with notably low precision and recall values, indicative of suboptimal model performance maybe attributable to overfitting or underfitting.

Another metric in Table 3 is the variance on the recall. It is calculated by measuring the variance among the five recall values obtained from each fold. This analysis ensures that a high average recall value is not driven by an outlier, but reflects consistent performance across all folds. By examining the variance, we test whether the recall values are consistently high, indicating robust model performance, or whether they vary significantly, suggesting sensitivity to specific subsets of data. To compare these variances to the average recall values, we actually look at the standard deviation, which in the highlighted case is 0.08609. This value, compared to the average 0.8444, seems reasonable and further validates the reliability and consistency of the performance of our highlighted neural network.

From Table 3, it is evident that when the learning rate is set to 0.1, we encounter notably low precision and recall values. Examination of the cost function in this case yields Fig. 4 across all five folds for the two combinations of hidden layers and nodes. The cost's behavior appears to be nonconvergent, oscillating.

During cross-validation, we use as threshold 10^{-6} for the early-stopping criterion mentioned in Appendix A.2.3.

4.1.2. Exploring selected network architectures

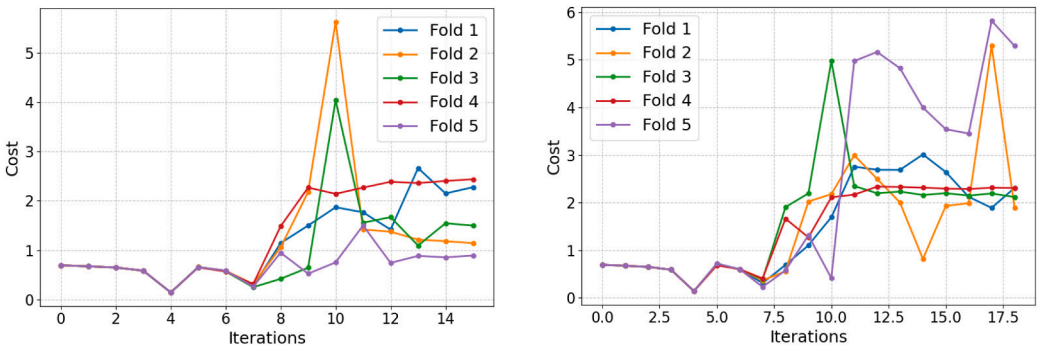
Training. Using the highlighted network in Table 3, the training is conducted on the entire training set. Fig. 5 displays the trend of the cost function during training and testing with a different test set (generated using the feature distributions in (4.1)). Notably, the expected behavior is observed, where the function steadily decreases as the number of iterations increases. Eventually, the function stabilizes, indicating convergence. The similar behavior of the cost function in both training and testing suggests that overfitting is not occurring. In this instance, a threshold of 10^{-8} is used for the early-stopping criterion.

Testing. In Table 4, the network's performance on the test set is presented. We specifically opt for a test set comprising 5 % of the total dataset, for a total of 117 samples. Notably, the recall value demonstrates a valuable performance, reaching 92 %. Conversely, the precision metric appears lower at 80 %. However, given our emphasis on recall over precision, this performance seems satisfactory.

Fig. 6a illustrates that by adjusting the classification threshold, such as to 0.75, a higher recall is obtained. Nevertheless, this adjustment inevitably entails a trade-off with precision; the decision regarding the threshold is thus left to the user. The ROC curve in Fig. 6b showcases an exemplary characteristic, summarized by the notably high AUC value of 0.99662. We recall that an AUC of

Table 3
Results of cross-validation for hyperparameter tuning for Case 1, times being obtained employing MPI with 5 processors. The columns correspond to the network structure (Table 1) and the learning rate (lr) of the network, and to the mean among the 5 folds of the following metrics and parameters: recall (rec), precision (prec), error rate (er_r), percentage difference between the training cost and the validation cost (%cost), iteration number (#iter) and computational time. The variance in recall is also provided.

net	lr	rec	prec	er_r	%cost	var	#iter	time[s]
1	0.1	0.4697	0.5502	0.05136	0.04110	0.01793	18	46.10
1	0.01	0.8444	0.8281	0.01888	0.05208	0.007412	411	987.9
1	0.0075	0.7657	0.8765	0.01950	0.02606	0.01153	349	1082
1	0.001	0.8287	0.8543	0.01733	0.03973	3.702×10^{-5}	2500	4925
2	0.1	0.3096	0.3490	0.07268	0.1750	0.01235	19	37.27
2	0.01	0.7809	0.8776	0.01858	0.03767	0.006237	416	885.0
2	0.0075	0.6956	0.8962	0.02187	0.03307	0.007823	212	398.7
2	0.001	0.8310	0.8537	0.01726	0.03972	3.385×10^{-5}	2500	5038



(a) Network $[12, 256, 512, 2500]$. (b) Network $[12, 512, 256, 2500]$.

Fig. 4. Cost during cross-validation with learning rate 0.1 for Case 1.

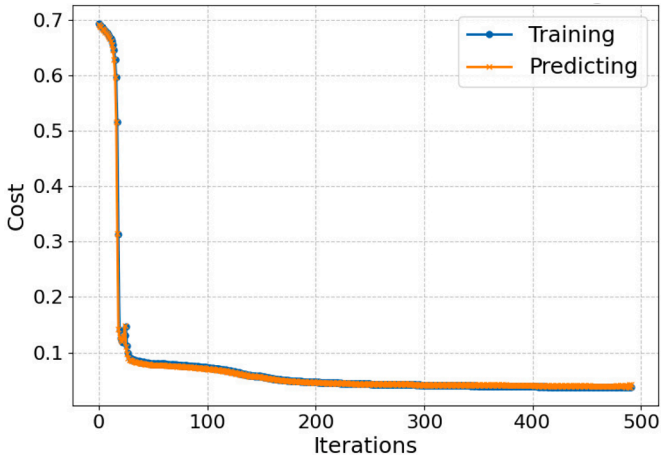


Fig. 5. Cost during training and prediction with different test set (cf.feq:diff-test-set) for Case 1.

Table 4
Metrics results for Case 1 on test set for chosen network with classification threshold 0.5. Columns: layer structure, learning rate (lr), recall, precision, error rate (er_r).

Layers and #nodes	lr	Recall	Precision	er_r
[12, 256, 512, 2500]	0.01	0.9276	0.8083	0.01676

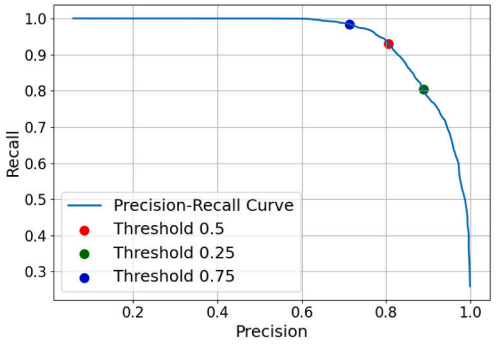
1 signifies perfect classification, wherein all positive instances supersede negative instances in score ranking across all thresholds. Additionally, Fig. 6c presents the confusion matrix at a classification threshold of 0.5. The majority of cells are correctly classified. Notably, the number of false negatives (1224) is lower than that of false positives (3720), which aligns with the desired outcome. The parity plot shown in Fig. 6d shows that the test examples closely align with the bisector within the first quadrant. This indicates a near equivalence between the predicted GF cells and the actual ones. While this does not explicitly indicate the accuracy of the predictions in terms of position in the mesh, it nonetheless provides a valuable insight.

Testing on different test set. To conduct further verification, we test the network using a dataset stemming from different parameter distributions. We use normal distributions for the influx, the Forchheimer coefficient and the Forchheimer exponent with means, standard deviations and cut-off intervals given by:

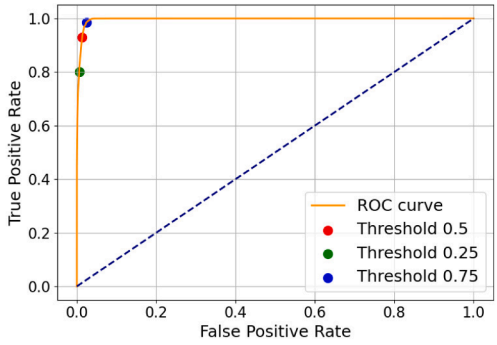
$$\begin{aligned} \mu_{u_0} = 0.02, \quad \sigma_{u_0} = 0.0035, \quad I_{u_0} = [0.00007, 0.05]; \\ \mu_{c_F} = 0.35, \quad \sigma_{c_F} = 0.45, \quad I_{c_F} = [0.05, 0.95]; \\ \mu_m = 2, \quad \sigma_m = 1.4, \quad I_m = [1, 4]. \end{aligned} \tag{4.1}$$

Table 5
Metrics results for Case 1 on different test set for network with layers and nodes [12, 256, 512, 2500], learning rate 0.01 and classification threshold 0.5. Columns: feature whose distribution changes, recall, precision and error rate (er_r).

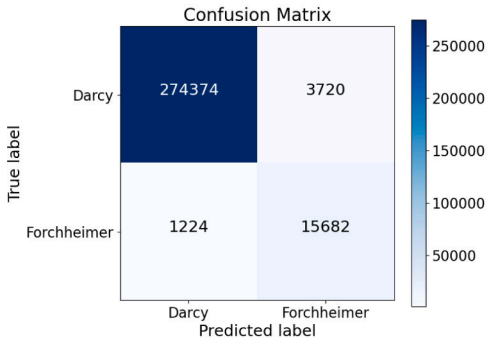
Changing feature	Recall	Precision	er_r
u_0	0.9348	0.8226	0.02541
c_F	0.9246	0.7948	0.01961
m	0.8689	0.7877	0.01827
δ	0.9260	0.8152	0.01912
all	0.9215	0.7952	0.02091



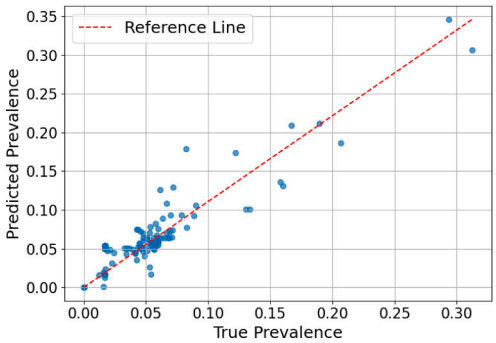
(a) Precision-recall curve.



(b) ROC curve.



(c) Confusion matrix with threshold of 0.5.



(d) Parity plot with threshold of 0.5.

Fig. 6. Performance plots on test set for Case 1.

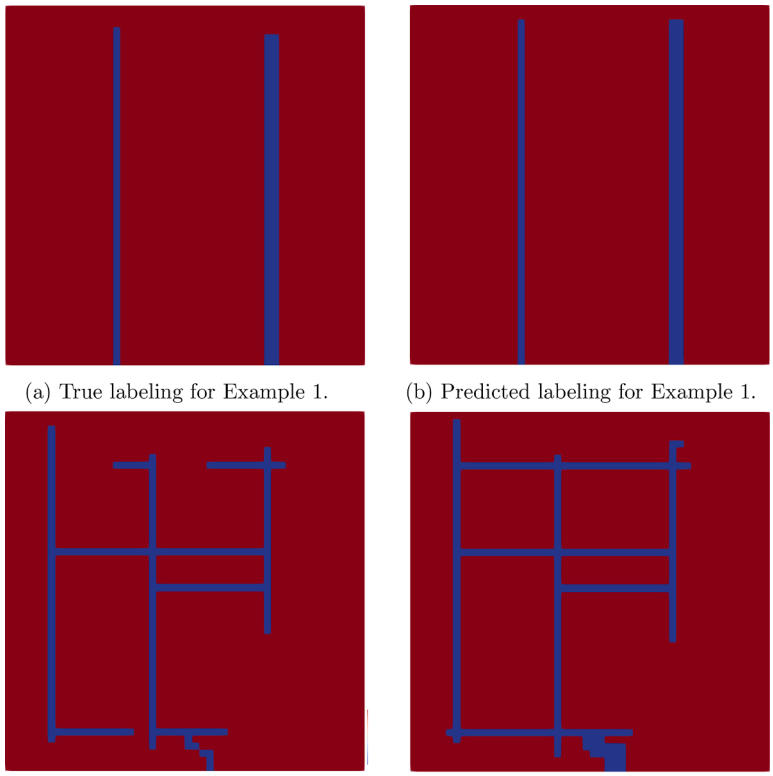
The error tolerance δ is selected from a uniform distribution within the interval $[0.008, 0.28]$. We then test the network by changing one distribution at a time and, finally, by changing all distributions simultaneously; see results in Table 5. There, we note that the metrics exhibit similar values to the test conducted previously (cf. Table 4), indicating that performance is consistent over slightly perturbed test sets, although we note a slight degradation in performance when the distribution for m is changed, which may be due to the lower variability of this feature in the training set, which in turn leads the network to have more difficulty generalizing.

Visualization. The visualization of the results is done via two test examples in Fig. 7, accompanied by the corresponding input values in Table 6. There, n_{chan} corresponds to the number of channels and ϕ_i, $i \in \{1, \dots, \}$, to the porosities in each channel. Analysis of Fig. 7a–d reveals instances where the network misclassifies, mostly predicting 'GF' instead of 'Darcy', but this is in line with our objectives since it avoids numerical errors, despite an increased computation time.

4.2. Case 2: layer 35 of SPE10 [27]

4.2.1. Hyperparameter tuning

Among the listed networks in Table 7, the one highlighted in yellow has highest recall which, as detailed in Appendix A.4.1, serves as our primary metric. This network demonstrates remarkable qualities, displaying high recall and precision, and low error



(a) True labeling for Example 1. (b) Predicted labeling for Example 1.

(c) True labeling for Example 2. (d) Predicted labeling for Example 2.

Fig. 7. Results for Case 1 for examples with input values from Table 6. GF labels are blue, and Darcy labels are red. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Table 6
Input parameters for Fig. 7.

ex.	u_0	c_F	m	δ	n_{chan}		
1	0.01224	0.8091	2.225	0.08302	2		
2	0.01583	0.4947	2.225	0.01494	7		
ex.	ϕ_1	ϕ_2	ϕ_3	ϕ_4	ϕ_5	ϕ_6	ϕ_7
1	0.9528	0.9719	–	–	–	–	–
2	0.9234	0.9485	0.9184	0.9315	0.9739	0.9766	0.9539

rate with respect to the others. However, it is essential to keep in mind the context: in this scenario, we are predict 13,200 cells for each example, with the majority being Darcy, hence we expect lower error rate compared to the first case study of Section 4.1. Another noteworthy aspect is the computational time, which is significantly higher than that of the first case (cf. Table 3), although this is understandable given the absence of GPU or parallel processing and the large volume of data to be predicted because of the mesh size.

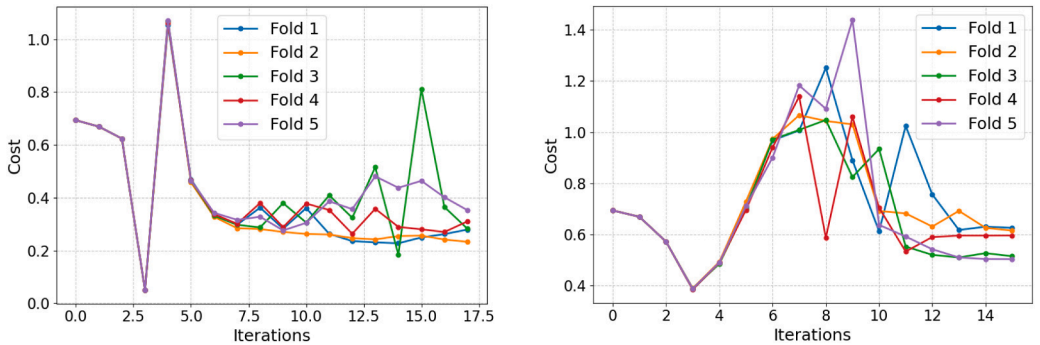
As for the previous case study, we look at the behavior of the cost function in the cases with learning rate equal to 0.1, even if from Table 7 the results for precision and recall seem quite similar to the other cases. From Fig. 8, we can observe that the values of the cost functions also fluctuate, but the fluctuations tend to be smaller than in Fig. 4, remaining consistently below 1 in amplitude in most cases. It is possible that delaying our early stopping may lead to convergence, despite the current appearance suggesting otherwise. In fact, to mitigate the computational cost without significantly compromising performance, we implement a 10^{-5} threshold for the early-stopping criterion. This adjustment aims to speed up the calculation process without overly impacting the metrics.

4.2.2. Exploring selected network architectures

Training. Using the network configuration shown in Table 7, training is performed on the entire training set. The trends of the cost function during training and testing are illustrated in Fig. 9. The expected behavior is observed with a constant decrease of the function as the number of iterations increases. Eventually, the function stabilizes, indicating convergence. The cost function in both training

Table 7
Results of cross-validation for hyperparameter tuning for Case 2, times being obtained employing MPI with 5 processors. The columns correspond to the network structure (Table 2) and the learning rate (lr) of the network, and to the mean among the 5 folds of the following metrics and parameters: recall, precision, error rate (er_r), percentage difference between the training cost and the validation cost (%cost), iteration number (#iter) and computational time. The variance in recall is also provided.

net	lr	rec	prec	er_r	%cost	var	#iter	time[s]
1	0.1	0.7808	0.8036	0.01644	0.09329	0.0008714	18	565.7
1	0.01	0.9508	0.9230	0.005406	0.03408	0.005964	390	10518
1	0.0075	0.7445	0.8490	0.01545	0.04425	$6.604\ 10^{-5}$	29	891.8
1	0.001	0.7315	0.8988	0.01397	0.01642	$7.198\ 10^{-5}$	236	6943
2	0.1	0.7398	0.8292	0.01643	0.01171	0.0005418	17	835.12
2	0.01	0.8082	0.8301	0.01426	0.1588	0.008703	116	9273
2	0.0075	0.8601	0.9399	0.00780	0.05539	0.004864	145	6060
2	0.001	0.7576	0.8323	0.01573	0.02330	$2.782\ 10^{-5}$	85	3949



(a) Network $[4, 256, 512, 13200]$. (b) Network $[4, 512, 1024, 2048, 13200]$.

Fig. 8. Cost during cross-validation with learning rate 0.1 for Case 2.

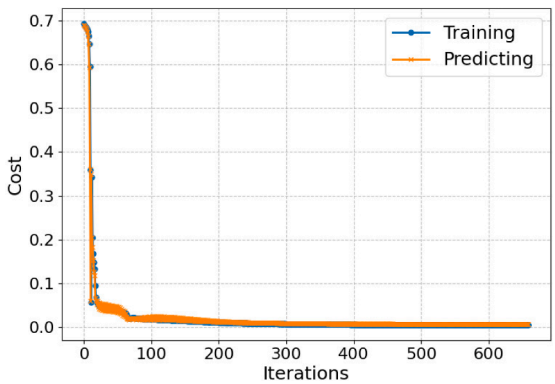


Fig. 9. Cost during training and prediction for Case 2.

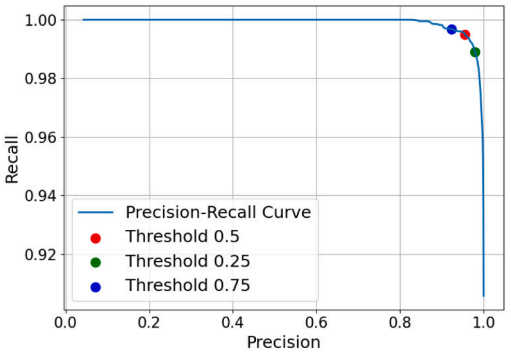
and testing behaves similarly indicating that there is no overfitting. In this scenario, an early stopping criterion with a threshold of 10^{-8} is used.

Testing. Table 8 shows the performance of the network on the test set, specially chosen to include 5 % of the entire dataset, with a total of 194 samples. The recall value obtained stands at 99 %, underlining the robustness of the network.

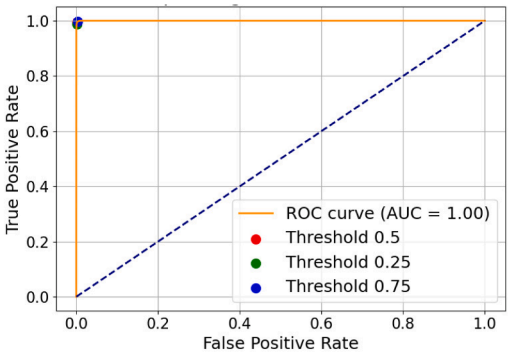
Both the precision-recall curve (Fig. 10a) and the ROC curve (Fig. 10b) present great characteristics, further confirming the results obtained. In particular, the AUC value is extremely close to 1, estimated at 0.99995. Furthermore, the confusion matrix (Fig. 10c)

Table 8
Metrics results for Case 2 on test set for chosen network with classification threshold 0.5. Columns: layer structure, learning rate (lr), recall, precision and error rate (er_r).

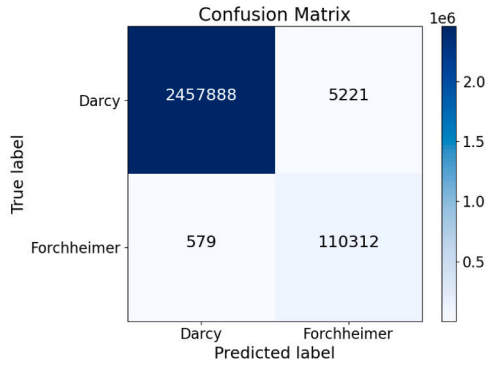
Layers and #nodes	lr	Recall	Precision	er_r
[4, 256, 512, 13200]	0.01	0.9948	0.9548	0.002253



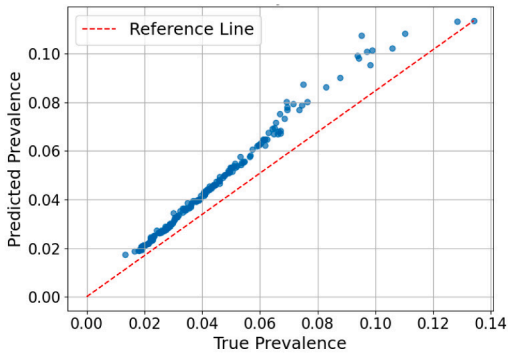
(a) Precision-recall curve.



(b) ROC curve.



(c) Confusion matrix with threshold of 0.5.



(d) Parity plot with threshold of 0.5.

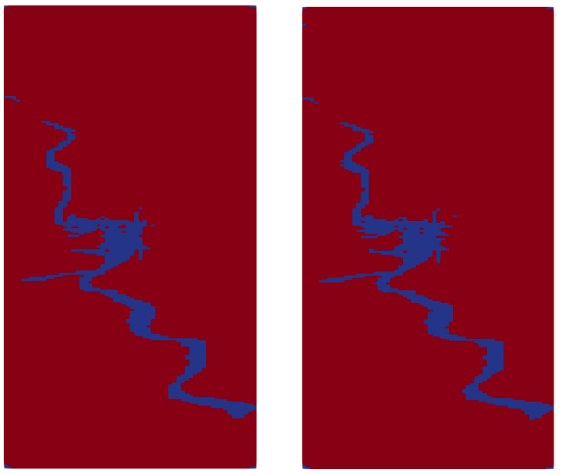
Fig. 10. Performance plots on test set for Case 2.

Table 9
Input parameters for Fig. 11.

ex.	Q	c_F	m	δ
1	92.975	0.45635	1	0.054017
2	145.69	0.49469	4	0.13594

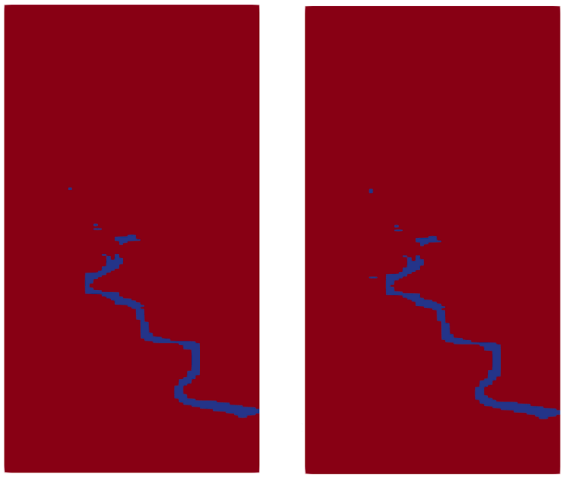
shows that the false negatives amount to only 579 instances out of the total number of cells, indicating an very efficient classification process. Also, in the parity plot illustrated in Fig. 10d, the plotted sample points closely adhere to the bisector within the first quadrant. Like in Case 1, this indicates a near equivalence between the predicted GF cells and the actual ones. However, it does not provide us with insights into their spatial distribution within the mesh.

Visualization. Fig. 11 gives the results for two test examples with the corresponding input values given in Table 9. The predictions demonstrate a high level of accuracy, beating that for the previous test case (cf. Fig. 6), which is probably attributable to a larger dataset and reduced variance; with fewer inputs resulting in less disparate outputs, the performance of the network is significantly improved.



(a) True labeling for Example 1.

(b) Predicted labeling for Example 1.



(c) True labeling for Example 2.

(d) Predicted labeling for Example 2.

Fig. 11. Results for Case 2 for examples with input values from Table 8. GF labels are blue, and Darcy labels are red. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

5. Conclusion

Given the obtained results, which help us locate the Darcy and generalized Forchheimer areas in a heterogeneous porous medium, the natural next step would be, as already mentioned in the introduction, to implement domain-decomposition methods. Overall, we would then have a complete approach partitioning a priori the domain into fast- and slow-flow regions and then solving the resulting heterogeneous model on a decomposed domain. The outcome is a method which is more precise than a global Darcy model and computationally faster than a global Forchheimer model; this, however, still remains to be studied. This comparative study would further quantify the advancements achieved through our partitioning strategy.

Turning now to the results presented in the present paper, we see a significant disparity in behavior between the SPE10 case (cf. Section 4.2) and the landfill management scenario (cf. Section 4.1), prompting further investigation of potential improvements. The datasets themselves have significant disparities. The landfill dataset, characterized by its smaller size and greater variance, comprises 12 inputs, each demonstrating considerable diversity. In fact, scenarios with only 2 channels feature 5 inputs at zero, while in scenarios with 7 channels, all porosity inputs approximate unity. As a result, these variations yield substantial disparities in the output images, particularly in the distribution of Darcy and generalized Forchheimer areas. Consequently, the networks struggle to generalize the behavior of the inputs effectively. In contrast, in the case of Layer 35 of the SPE10 scenario, with only 4 more homogeneous inputs and outputs, coupled with a larger sample size, the network adeptly generalizes the behaviors.

To resolve these disparities and improve performance, several strategies can be pursued. First, one may expect that enriching and diversifying the dataset for the landfill model should prove useful. We thus conducted cross-validation using various datasets and

Table 10
Results of cross-validation for hyperparameter tuning in our chosen network for Case 1, with layers and nodes [12, 256, 512, 2500] and learning rate 0.01. The columns correspond to the size of the entire dataset (n_{dataset}), and to the mean among the 5 folds of the following metrics and parameters: recall, precision, error rate (er_r), percentage difference between the training cost and the validation cost (%cost) and iteration number (#iter). The network highlighted in yellow is that analyzed in the Section 4.1.

$n_{dataset}$	Recall	Precision	er_r	%cost	#iter
107	0.7835	0.7979	0.02097	0.6819	248
1295	0.7388	0.9177	0.02167	0.04438	211
2352	0.8444	0.8281	0.01888	0.05208	411
2687	0.8472	0.8246	0.02162	0.05261	328
3584	0.8432	0.8142	0.01361	0.03896	776
6144	0.8038	0.7823	0.01641	0.02962	671
7776	0.8248	0.8355	0.01825	0.01699	186

sample sizes. The findings, given in Table 10, reveal a persistent trend indicating that variations in dataset size exert a negligible influence on network performance. This observation underscores the suitability of the selected dataset size for our analytical purposes, suggesting that further expansion of the dataset would not yield benefits. Notably, despite the modest dataset size, the network demonstrates robust performance and effectively captures the underlying output patterns, which shows its efficacy in learning and generalizing from limited data.

Second, training separate networks for the cases with 2 or 7 channels could be beneficial to Case 1, given the manageable computational overhead due to the smaller output size compared to Layer 35 of the SPE10 case given the coarser mesh (2500 cells vs. 13,200 cells). Additionally, integrating dropout regularization during training may strengthen the robustness of the model. While incorporating L^2 regularization into the cost function typically helps mitigate overfitting and improve generalization, Figs. 5 and 9 show that overfitting does not appear to be an issue in our case. Overall, these approaches offer promising avenues for enhancing our network performance and bridging the observed gap between the two scenarios.

Third, a more extensive hyperparametric tuning than that given here would be necessary to further validate our approach. Indeed, the presented search for optimal hyperparameters is rather narrow and searching for more hidden layers, nodes and optimizing algorithms would improve the validity of our results.

Fourth, a notable enhancement could be obtained by switching from NumPy [39] to PyTorch [40] or Tensorflow [41]. NumPy is a powerful library for numerical computations, but PyTorch and Tensorflow are prevalent libraries in the deep-learning community which offer additional benefits specifically tailored for building and training neural networks, such as GPU acceleration and automatic differentiation.

Finally, a possible direction to take to generalize our model would be to introduce a permeability tensor into the model, instead of using a scalar. This is not trivial as there is no general consensus on the exact form the generalized Forchheimer law should take in this case; possible formulations are given in Spena and Vacca [29]. This would introduce anisotropies, particularly relevant to three-dimensional scenarios, and testing our neural-network approach in this case would be a significant improvement. An additional generalization would be the inclusion of second-order derivative terms in the underlying laws, such as a Laplacian on the velocity as in Brinkman's model. This, however, relies on the theory of well-posedness for the adaptive model in this higher-order scenario, which is not yet available and particularly delicate given the additional discontinuity at the subdomain boundary on the velocity derivatives.

All in all, although there is still work to arrive at a complete understanding and validation of our approach, the results obtained so far are promising. They provide a solid foundation on which to build further refinements and investigations.

Significance and novelty of this paper

In porous media with heterogeneous permeabilities, a wide range of fluid velocities can occur, leading to significant inertial and frictional effects in high-speed regions. In such areas, the pressure gradient and velocity relationship is often described by Darcy's law, which may not adequately capture these effects. Instead, the Darcy–Forchheimer law, which introduces a nonlinear term, is more suitable. However, applying the Darcy–Forchheimer law across the entire domain is computationally expensive and should be restricted to necessary regions.

A recent study addressed the challenge of identifying subdomains where the Darcy–Forchheimer law should be applied by using an adaptive model. This model dynamically selects the appropriate law based on a velocity threshold during the solution process, flagging each mesh cell as belonging to either the Darcy or Darcy–Forchheimer subdomain upon completion. Despite its effectiveness, the model is nonlinear and computationally intensive.

To improve efficiency, this paper proposes using the adaptive model to generate partitioning data based on various input parameters, such as boundary conditions and inertial coefficients. This data is then used to train neural networks to classify each mesh cell as requiring either the Darcy or Darcy–Forchheimer law. Two test cases are examined to demonstrate the results, utilizing cost functions, parity plots, precision-recall plots, and receiver operating characteristic curves for analysis.

CRediT authorship contribution statement

Chiara Giovannini: Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Software, Resources, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Data curation, Conceptualization; **Alessio Fumagalli:** Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Software, Resources, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Data curation, Conceptualization; **Francesco Saverio Patacchini:** Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Software, Resources, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Data curation, Conceptualization.

Data availability

Data will be made available on request.

Declaration of competing interest

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

Acknowledgment

This paper was made possible thanks to the support of IFP Energies nouvelles, the research laboratory where this work was conducted. The present research is part of the activities of "Dipartimento di Eccellenza 2023–2027", Italian Minister of University and Research (MUR), grant Dipartimento di Eccellenza 2023–2027.

Appendix A. Fundamentals of our neural networks

Neural networks are particularly suited for recognizing intricate patterns and extracting essential features. Moreover, their generalizing power to new data makes them a particularly attractive tool for our purpose, which is to predict a vector representing mesh cells that exhibit common patterns [42]. We use feed-forward neural networks with dense layers implemented using the Numpy library [39] following a methodology outlined in the course [43] and using a class-based architecture to improve modularity and maintainability.

We delve below into the various components of our neural networks.

A.1. Datasets

The datasets employed in this investigation are constructed using the codebase provided by Fumagalli and Patacchini [22], after suitable modification so as to align them with the specific requirements of our study.

Each analytical scenario we test in this paper involves generating an input vector for each instance, or example, within the dataset and generating a respective binary output vector. Each slot in the output vector represents a cell within our mesh. Upon obtaining each dataset, we divide it into the following sets:

- a **training set** consisting of examples for tuning hyperparameters, such as number of hidden units and learning rate, and for training the network;
- a **test set** for measuring generalization performance.

This division is crucial for assessing how well our model generalizes to new, previously unseen data. In particular, in the cross-validation step performed to optimize the hyperparameters, the training set is further subdivided into folds, each in turn playing the role of a **validation set** (cf. Appendix A.3.1). The cross-validation is essential, as tuning hyperparameters on the entire training set is not viable due to prior exposure and may lead to overfitting, and the test set is reserved for reporting the final performance and is usable only once [38].

A fundamental pre-processing step common to both test cases is the normalization of input data with respect to the training set. This enhances the model's ability to learn patterns and generalize across diverse instances within the dataset.

A.2. Architecture

A.2.1. Dense layers

We decide to exploit classical dense layers, also known as fully connected layers. These layers exhibit simplicity and flexibility, each neuron establishing deep connections with and receiving input from every neuron in its preceding layer, facilitated through matrix-vector multiplication. Broadly speaking, a multilayer perceptron, that is, a dense feed-forward neural network, comprises $L+1$ layers (with the input considered as Layer 0) and includes $L-1$ hidden layers, with the output layer being the Lth layer. Each layer can have varying numbers of neurons, namely, n_l where l ranges from 0 to L (Fig. 12).

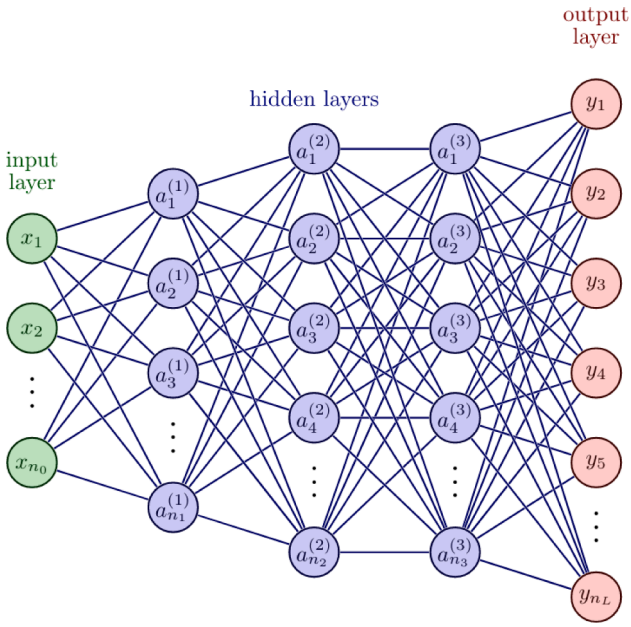


Fig. 12. General structure of a dense feed-forward neural network [44].

In our specific implementation, input data is represented by a matrix with dimensions (n_0, s), where n_0 is the number of inputs and s is the size of the dataset. The neural network's operation can be dissected into two phases: the forward propagation and the backward propagation (also known as backpropagation).

During the forward propagation, matrix-vector multiplication is applied to calculate Z, the input to the activation function, which is a mathematical operation that may introduce nonlinearity to the network to learn complex patterns in the data. The dimensions of Z are (n_l, s), where l denotes the current layer. The computation is expressed as

$$Z^{(l)} = W^{(l)} A^{(l-1)} + b^{(l)}.$$

Here, $W^{(l)}$ is the weight matrix of dimensions (n_l, n_{l-1}), $b^{(l)}$ the bias vector with dimensions $(n_l, 1)$, and $A^{(l-1)}$ the activations from the previous layer or input data, with dimensions (n_{l-1}, s). The resulting matrix $Z^{(l)}$ serves as the input to the activation function a, shaping the network's output according to $A^{(l)} = a(Z^{(l)})$.

A.2.2. Activation functions

Our neural network employs commonly used activation functions, specifically the Rectified Linear Unit (ReLU) and the sigmoid activation function. The ReLU activation function $\mathrm{ReLu} : \mathbb{R} \to \mathbb{R}$ is defined as

$$\mathrm{ReLU}(Z) = \max(0, Z),$$

and it is used in all hidden layers. One of the key advantages of ReLU is its computational efficiency compared to other activation functions like the sigmoid or the hyperbolic tangent, as it involves simple element-wise operations. Additionally, ReLU helps mitigate the vanishing gradient problem, occurring when the gradients calculated during backpropagation become extremely small as they propagate backwards through the network layers, eventually causing the gradients effectively to vanish. By producing zero outputs for negative inputs and transmitting positive inputs unaltered, ReLU ensures that it avoids saturation for positive inputs, thus enabling continuous gradient flow during backpropagation. This characteristic distinguishes ReLU from functions such as sigmoid or tanh, which are prone to gradient vanishing. Consequently, ReLU facilitates the information propagation across network layers during backpropagation, thereby promoting expedited convergence and safeguarding against neuron saturation [45]. The output layer, responsible for binary output, employs the sigmoid activation function $\sigma : \mathbb{R} \to (0, 1)$:

$$\sigma(Z) = \frac{1}{1 + e^{-Z}}.$$

The sigmoid activation function effectively squashes the output values between 0 and 1 (cf. Fig. 13). Moreover, in specific contexts such as domain classification tasks, the output of the sigmoid function assumes a significant interpretation. For instance, in classifying cells into Darcy or Forchheimer domains, the output of the sigmoid function directly corresponds to the likelihood that a given cell belongs to either category; for example, if the output is 0.7, it means that the probability that the cell belongs to the Darcy class is 70 %.

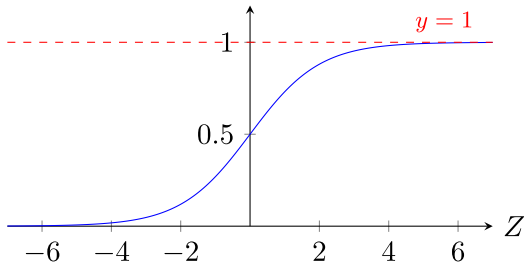


Fig. 13. Graph of sigmoid activation function, σ.

A.2.3. Cost function

In the backward propagation, the parameters W and b are trained and updated. Backpropagation involves computing the gradient of the cost, or loss, function with respect to the network's weights. In our case, where the output is a binary classification vector, we use the cross-entropy cost function, namely,

$$c(W, b) = -\frac{1}{sk} \sum_{i=0}^{k} \sum_{j=0}^{s} [y_{i,j} \log(a_{i,j}) + (1 - y_{i,j}) \log(1 - a_{i,j})].$$

Here, s is the size of the dataset, k is the size of the output vector (i.e., the size of the mesh), $y_{i,j}$ is the element in the position (i, j) in the true label vector, $a_{i,j}$ is the element (i, j) in the probability vector corresponding to the label predictions. Cross-entropy provides a measure of dissimilarity between the true labels and the predicted probabilities, lower values of cross-entropy indicating better model performance.

To optimize computational efficiency and expedite the training process, we implement an early-stopping criterion. Specifically, we halt the training procedure when 15 consecutive iterations yield no discernible improvement. This stopping criterion is compatible with the characteristics of our dataset, as we aim to strike a balance between computational resource use and achieve optimal model performance. Moreover, we introduce an additional criterion: we terminate the network training if the discrepancy in improvements falls below a certain threshold. This threshold varies for each test case and differs between cross-validation and training of the selected network. Specifically, considering that the cross-validation phase is more time-consuming, we opt for a higher threshold value, so that we stop before. Conversely, when computing on the entire training dataset, the computational time is more manageable, allowing for the adoption of a slightly smaller threshold value.

A.3. Hyperparameters

Hyperparameters are a set of fixed parameters within a neural network architecture that remain unchanged during training. These parameters play a crucial role in shaping the network's performance and behavior. Unlike trainable parameters, like weights, hyperparameters are predetermined either manually or through optimization procedures prior to training. Examples of hyperparameters include the learning rate, the architecture's depth (number of hidden layers), the width (number of neurons) of each layer.

Tuning hyperparameters is fundamental for optimizing the performance of the network, as different configurations can lead to significantly different outcomes in terms of accuracy, metrics, convergence speed, and generalization ability [46].

A.3.1. Cross-validation

To tune the hyperparameters, we employ the widely used method of cross-validation, specifically opting for the κ-fold cross-validation, where κ denotes the user-specified number of folds—in our case, 5. Cross-validation provides several advantages, including the avoidance of biased validation-set selection and more effective data use. It ensures that each example is included in the training set exactly once, promoting robust generalization to all dataset samples [38].

After initially partitioning our dataset into training and test sets according to a split of 95 % and 5 %, the training set is further divided into five folds of approximately equal sizes. Subsequently, a sequence of models is trained, each time using a different fold as the validation set while the remaining folds are used for training. This process, illustrated in Tables 14, is repeated for all five splits, resulting in the collection of five vectors of metric values (cf. Appendix A.4 for a detailed description of the metrics). Then, we compute the mean of the chosen metric over the five folds to obtain an overall evaluation of the neural-network model.

Care should also be taken when splitting the dataset to avoid potential issues related to data order. We shuffle the data to eliminate any inherent ordering in the samples based on labels.

A.3.2. Systematic methodology

As shown in Fig. 1, our methodology involves a systematic approach to model development and evaluation. Initially, we partition the dataset into training and testing subsets, employing the former for cross-validation procedures detailed in Appendix A.3.1. Leveraging this framework, we engage in hyperparameter tuning to extract optimal configurations, using them to train the model on the

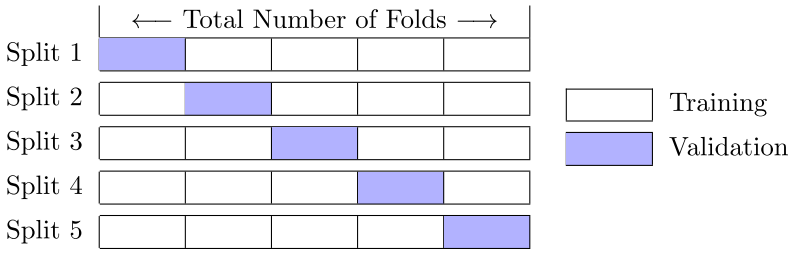


Fig. 14. Cross-validation split used in our test cases. Rows represent cross-validation iterations; columns depict folds into which the dataset is partitioned.

entirety of the training data, producing the final model. We then subject this model to evaluation using the test set and the training results.

In addition to the number of hidden layers and number of nodes, the hyperparameter we tune is the learning rate; we decide to try 4 different values, namely, 0.1, 0.01, 0.0075 and 0.001, thus spanning three different orders of magnitude and keeping a reasonable balance between precision and computation time.

In the hyperparameter tuning phase, we adopt a random-search strategy. Recognizing that our input vectors span at least two orders of magnitude, we posit the effectiveness of using hidden layers with a high number of nodes. This architectural choice facilitates the diffusion of information across a multitude of nodes, leading to convergence towards a complete vector representation. Additionally, our empirical observations indicate that networks characterized by layers with an increasing number of nodes from input to output tend to show better performance, aligning with the inherent progression from inputs to desired outputs. Accordingly, we explore various random combinations of node and hidden-layer counts (cf. Tables 1 and 2).

A.4. Metrics

We define four fundamental quantities, referred to as metrics, integral to the evaluation of neural network models predicting mesh-cell classifications. In our context, a single example yields a vector representing mesh cells as output. Specifically, we classify GF cells as *positive* and Darcy cells as *negative*, the former being the prediction we wish to miss the least, as, indeed, predicting GF when the truth is Darcy is not as detrimental precison-wise as predicting Darcy when the truth is GF. Four distinct outcomes are possible:

- **True Positive (TP)**: the neural network correctly identifies a positive output cell;
- **True Negative (TN)**: the neural network correctly identifies a negative output cell;
- **False Positive (FP)**: the neural network incorrectly labels a negative output cell as positive;
- **False Negative (FN)**: the neural network incorrectly labels a positive output cell as negative.

A.4.1. Precision and recall

One widely used metric in evaluating neural network performance is the *error rate* (Err), representing the frequency of classification errors over a given set of examples. It is defined as

$$\mathrm{Err} = \frac{\mathrm{FP} + \mathrm{FN}}{\mathrm{FP} + \mathrm{FN} + \mathrm{TP} + \mathrm{TN}},$$

where FP, FN, TP and TN are the aggregated quantities for every cell across all examples in the set. An alternative metric, the classification *accuracy* (Acc), is the frequency of correct classifications and is related to the error rate by $\mathrm{Acc} = 1 - \mathrm{Err}$.

However, error rates may not provide a comprehensive view, especially in imbalanced datasets where one class significantly outnumbers the other, which is what we expect in our case where most domain should be classified as negative, that is, as Darcy. In our case, the abundance of Darcy cells necessitates additional metrics, namely, *precision* (Pr) and *recall* (Re):

$$\mathrm{Pr} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} \quad \text{and} \quad \mathrm{Re} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}.$$

Precision quantifies the likelihood that the classifier is correct when labeling an example as positive, while recall measures the probability that a positive example is correctly recognized.

In our study, recall is more relevant than precision due to the detrimental impact of false negatives compared to false positives. Indeed, as already mentioned, misclassifying a cell as GF when it is actually Darcy merely entails the resolution of a nonlinear constitutive law (cf. (2.4)), incurring an increase in computational time but no numerical inaccuracies. Conversely, misidentifying a cell as Darcy when it is truly GF leads to a substantial numerical error. Consequently, in selecting the most suitable network for our test cases, we prioritize those exhibiting higher recall since indeed a smaller FN yields a higher value. This parallels situations in medical diagnosis, where correctly identifying a patient with condition X constitutes a true positive, while failing to diagnose a patient with X constitutes a false negative—this latter outcome, medical practitioners endeavor to minimize and thereby advocate a low FN rate, i.e., a high recall.

A.4.2. Performance plots

To evaluate further the performance of our trained neural network, we wish to study its behavior depending on the choice of the threshold in our binary classification. After the training process, the model outputs real values confined in the range $[0, 1]$; to get a binary value, 0 or 1, we need to fix a threshold separating the output values mapped to 0 from those mapped to 1. The commonly used threshold is 0.5, but it can be adjusted to achieve a desired trade-off between precision and recall. To get the desired trade-off, we examine all possible thresholds using the precision-recall curve. The proximity of the curve to the top-right corner indicates balance in the classifier performance, meaning that the points in this region represent high precision *and* high recall for the same threshold.

Another commonly used tool for analyzing classifier behavior at different thresholds is the Receiver Operating Characteristics (ROC) curve. Similar to the precision-recall curve, the ROC curve considers all possible thresholds for a given classifier but illustrates the trade-off between the False Positive Rate (FPR) and the True Positive Rate (TPR), where TPR is synonymous with recall and FPR, also called *fall-out*, is the fraction of false positives out of all negative samples:

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}}.$$

In the ROC curve, the ideal performance is reflected by a curve close to the top-left corner, indicating high recall while maintaining a low positive rate. Our ideal scenario aims for the top-left corner, where there is a zero error rate on the negatives and 100 % accuracy on the positives. The point nearest to the top left is regarded as the most balanced operating point in the sense that it represents high TPR *and* high FPR. Moreover, the ROC curve is often summarized using a single metric: the Area Under the Curve (AUC). A higher AUC signifies better overall performance, and a perfect AUC of 1 implies that all positive points have a higher score than all negative points. Notably, AUC is particularly valuable for imbalanced classification problems, as it provides a more informative metric than accuracy.

A further method to compare our neural networks is the parity plot. For each example in the set, we compare the prevalence of the predicted output against the expected prevalence. The *prevalence* for example $j \in \{1, \dots, s\}$ (Prv_j) is the ratio of the sum of positive cells in the jth example to the total number k of cells:

$$\mathrm{Prv}_j = \frac{1}{k} \sum_{\substack{i=1 \\ \text{cell } i \text{ positive}}}^{k} y_{ij},$$

where y_{ij} is the entry in the ith row and jth column of the output matrix Y. In the parity plot, the reference line, represented by the diagonal equation $y = x$, denotes perfect model performance, where, for every example, the model predicts the exact number of positive (i.e., GF) cells as present in the mesh, that is, the true and predicted prevalences match. The distance of each point in the plot, i.e., each example, from the diagonal indicates the extent to which the model deviates in the number of positive cells predicted. While this metric is not flawless, as a model could predict the exact number of positive values but in the wrong cells, it is complemented by other metrics to provide a comprehensive evaluation of the model.

Finally, one of the most comprehensive tools for representing the results of binary classifications are confusion matrices. These are two-by-two matrices in which the rows correspond to the true classes and the columns to the predicted classes. Each entry indicates the frequency with which a cell belonging to the class in the row is classified in the class in the column. Entries on the main diagonal indicate correct classifications, while entries off the diagonal indicate incorrect ones, which provides a granular view of the performance of the model [38].

References

[1] A. Fumagalli, F.S. Patacchini, Well-posedness and variational numerical scheme for an adaptive model in highly heterogeneous porous media, J. Comput. Phys. 475 (2023) 111844. https://doi.org/10.1016/j.jcp.2022.111844

[2] J. Bodin, P. Ackerer, A. Boisson, B. Bourbiaux, D. Bruel, J.R. de Dreuzy, F. Delay, G. Porel, H. Pourpak, Predictive modelling of hydraulic head responses to dipole flow experiments in a fractured/karstified limestone aquifer: insights from a comparison of five modelling approaches to real-field experiments, J. Hydrol. 454–455 (2012) 82–100. https://doi.org/10.1016/j.jhydrol.2012.05.069

[3] R. Ershadnia, C.D. Wallace, M.R. Soltanian, CO_2 geological sequestration in heterogeneous binary media: effects of geological and operational conditions, Adv. Geo-Energy Res. 4 (4) (2020) 392–405. https://doi.org/10.46690/ager.2020.04.05

[4] R. Winter, A. Valsamidou, H. Class, B. Flemisch, A study on darcy versus forchheimer models for flow through heterogeneous landfills including macropores, Water 14 (4) (2022) 546. https://doi.org/10.3390/w14040546

[5] P. Panja, J. McLennan, S. Green, Impact of permeability heterogeneity on geothermal battery energy storage, Adv. Geo-Energy Res. 5 (2) (2021) 127–138. https://doi.org/10.46690/ager.2021.02.03

[6] W. Sobieski, A. Trykozko, Darcy's and Forchheimer's laws in practice. Part 1. The experiment, Tech. Sci. 17 (4) (2014) 321–335.

[7] W. Sobieski, A. Trykozko, Darcy's and Forchheimer's laws in practice. Part 2. The numerical model, Tech. Sci. 17 (4) (2014) 337–350.

[8] T. Zhang, Y. Zhao, Q. Gan, L. Yuan, G. Zhu, Y. Cai, B. Cao, Experimental investigation of Forchheimer coefficients for non-Darcy flow in conglomerate-confined aquifer, Geofluids (2018) 1–21. https://doi.org/10.3390/w14040546

[9] E. Marušić-Paloka, A. Mikelić, The derivation of a nonlinear filtration law including the inertia effects via homogenization, Nonlinear Anal. Theory Methods Appl. 42 (1) (2000) 97–137. https://doi.org/10.1016/S0362-546X(98)00346-0

[10] S.P. Neuman, Theoretical derivation of Darcy's law, Acta Mech. 25 (3–4) (1977) 153–170. https://doi.org/10.1007/BF01376989

[11] D. Ruth, H. Ma, On the derivation of the Forchheimer equation by means of the averaging theorem, Transp. Porous Media 7 (3) (1992). https://doi.org/10.1007/BF01063962

[12] S. Whitaker, Flow in porous media I: a theoretical derivation of Darcy's law, Transp. Porous Media 1 (1) (1986) 3–25. https://doi.org/10.1007/BF01036523

[13] S. Whitaker, The Forchheimer equation: theoretical development, Transp. Porous Media 25 (1) (1996) 27–61. https://doi.org/10.1007/BF00141261

[14] P.M. Knupp, J.L. Lage, Generalization of the Forchheimer-extended Darcy flow model to the tensor premeability case via a variational principle, J. Fluid Mech. 299 (1995) 97–104. https://doi.org/10.1017/S0022112095003430

[15] M.S. Espedal, K.J. Hersvik, B.G. Ersland, Domain decomposition methods for flow in heterogeneous porous media, Contemp. Math. 218 (1998) 104–120.

[16] J.O. Skogestad, E. Keilegavlen, J.M. Nordbotten, Domain decomposition strategies for nonlinear flow problems in porous media, J. Comput. Phys. 234 (2013) 439–451. https://doi.org/10.1016/j.jcp.2012.10.001
[17] E. Ahmed, A. Fumagalli, A. Budiša, A multiscale flux basis for mortar mixed discretizations of reduced Darcy–Forchheimer fracture models, Comput. Methods Appl. Mech. Eng. 354 (2019) 16–36. https://doi.org/10.1016/j.cma.2019.05.034
[18] F. Févotte, A. Rappaport, M. Vohralík, Adaptive regularization, discretization, and linearization for nonsmooth problems based on primal–dual gap estimators, Comput. Methods Appl. Mech. Eng. 418 (2024) 116558. https://doi.org/10.1016/j.cma.2023.116558
[19] D.A. Di Pietro, M. Vohralík, S. Yousef, Adaptive regularization, linearization, and discretization and a posteriori error control for the two-phase Stefan problem, Math. Comput. 84 (2015) 153–186.
[20] A. Fumagalli, F.S. Patacchini, Model adaptation for non-linear elliptic equations in mixed form: existence of solutions and numerical strategies, ESAIM 56 (2) (2022) 565–592. https://doi.org/10.1051/m2an/2022016
[21] A. Fumagalli, F.S. Patacchini, Numerical validation of an adaptive model for the determination of nonlinear-flow regions in highly heterogeneous porous media, Math. Comput. Simul. 230 (2025) 306–322. https://doi.org/10.1016/j.matcom.2024.10.024
[22] A. Fumagalli, F.S. Patacchini, validate_adaptative, https://github.com/compgeo-mox/validate_adaptative.
[23] C. Giovannini, darcy_forch, https://github.com/chiaragiovannini/darcy_forch.
[24] E. Keilegavlen, R. Berge, A. Fumagalli, M. Starnoni, I. Stefansson, J. Varela, I. Berre, PorePy: an open-source software for simulation of multiphysics processes in fractured porus media, Comput. Geosci. 25 (2020) 243–265.
[25] porepy, https://github.com/pmgbergen/porepy.
[26] W.M. Boon, A. Fumagalli, A reduced basis method for Darcy flow systems that ensures local mass conservation by using exact discrete complexes, J. Sci. Comput. 94 (64) (2023). https://doi.org/10.1007/s10915-023-02119-3
[27] M.A. Christie, M.J. Blunt, Tenth SPE Comparative Solution Project: A Comparison of Upscaling Techniques, Society of Petroleum Engineers, Houston, Texas, 2001, p. 13. https://doi.org/10.2118/66599-MS
[28] J.D. Audu, F.A. Fairag, S.A. Messaoudi, On the well-posedness of generalized Darcy–Forchheimer equation, Bound. Value Probl. 2018 (123) (2018). https://doi.org/10.1186/s13661-018-1043-6
[29] F.R. Spena, A. Vacca, A potential formulation of non-linear models of flow through anisotropic porus media, Transp. Porous Media 45 (2001) 405–421.
[30] E. Aulisa, L. Bloshanskaya, L. Hoang, A. Ibragimov, Analysis of generalized Forchheimer flows of compressible fluids in porous media, J. Math. Phys. 50 (10) (2009) 103102. https://doi.org/10.1063/1.3204977
[31] L. Hoang, A. Ibragimov, Structural stability of generalized Forchheimer equations for compressible fluids in porous media, Nonlinearity 24 (1) (2011) 1–41. https://doi.org/10.1088/0951-7715/24/1/001
[32] V.A. Kovtunenko, Mixed variational problem for a generalized Darcy–Forchheimer model driven by hydraulic fracture, J. Vib. Test. Syst. Dyn. 7 (1) (2023) 15–21. https://doi.org/10.5890/JVTSD.2023.03.003
[33] A. Barhoi, Numerical solution of skin friction in porous medium using Brinkman Forchheimer model under fuzzy environment, Afr. J. Biomed. Res. 27 (4S) (2024). https://doi.org/10.53555/AJBR.v27i4S.3783
[34] A.P. Baitharu, S. Sahoo, G.C. Dash, Numerical approach to non-Darcy mixed convective flow of non-newtonian fluid on a vertical surface with varying surface temperature and heat source, Karbala Int. J. Mod. Sci. 6 (3) (2020).
[35] Z. Zeng, R. Grigg, A criterion for non-Darcy flow in porous media, Transp. Porous Media 63 (2006) 57–69.
[36] P. Macini, E. Mesini, R. Viola, Laboratory measurements of non-Darcy flow coefficients in natural and artificial unconsolidated porous media, J. Pet. Sci. Eng. 77 (3) (2011) 365–374. https://doi.org/10.1016/j.petrol.2011.04.016
[37] F. Camilli, A note on convergence of level sets, Z. Anal. Anwend. 18 (1) (1999) 3–12. https://ems.press/journals/zaa/articles/11030. https://doi.org/10.4171/ZAA/865
[38] A. Muller, S. Guido, Introduction to Machine Learning with Python, O'Reilly, 2017.
[39] C.R. Harris, K.J. Millman, S.J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N.J. Smith, R. Kern, M. Picus, S. Hoyer, M.H. van Kerkwijk, M. Brett, A. Haldane, J. Fernández del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, T.E. Oliphant, Array programming with NumPy, Nature 585 (7825) (2020) 357–362. https://doi.org/10.1038/s41586-020-2649-2
[40] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala, PyTorch: an imperative style, high-performance deep learning library, in: Advances in Neural Information Processing Systems 32, Curran Associates, Inc., 2019, pp. 8024–8035. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.
[41] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, 2015. Software available from tensorflow.org, https://www.tensorflow.org/.
[42] R. Rojas, Neural Networks—A Systematic Introduction, Springer, 1996.
[43] A. Ng, Y. Bensouda Mourri, K. Katanforoosh, Neural Networks and Deep Learning, https://www.coursera.org/learn/neural-networks-deep-learning.
[44] I. Neutelings, Graphics with TikZ in LaTeX: neural networks, https://tikz.net/neural_networks.
[45] X. Glorot, A. Bordes, Y. Bengio, Deep sparse rectifier neural networks, Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (2011) 315–323.
[46] F. Hutter, L. Kotthoff, J. Vanschoren, Automated Machine Learning. Methods, Systems, Challenges, Springer, 2019.