Computers and Chemical Engineering 177 (2023) 108342

Contents lists available at ScienceDirect

Computers and Chemical Engineering

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

Data augmentation driven by optimization for membrane separation process synthesis

Bernardetta Addis [a,*], Christophe Castel [b], Amalia Macali [a,b], Ruth Misener [c], Veronica Piccialli [d]

[a] University of Lorraine - CNRS - LORIA, Nancy Cedex, France
[b] University of Lorraine - CNRS - LRGP, Nancy Cedex, France
[c] Imperial College,Department of Computing, London, United Kingdom
[d] Sapienza University, DIAG, Rome, Italy

ARTICLE INFO

MSC:
62M45NN
90C26
90C90

Keywords:
Neural network
Data augmentation
Global optimization
Membrane gas separation

ABSTRACT

This paper proposes a new hybrid strategy to optimally design membrane separation problems. We formulate the problem as a Non-Linear Programming (NLP) model. A common approach to represent the physical behavior of the membrane is to discretize the system of differential equations that govern the separation process. Instead, we represent the input/output behavior of the single membrane by an artificial neural network (ANN) predictor. The ANN is trained on a dataset obtained through the MEMSIC simulator. The equation form of the trained predictor (shape and weights) is then inserted in the NLP model at the place of the discretized system of differential equations.

To improve the ANN accuracy without an excessive computational burden, we propose data augmentation strategies to target the regions where densify the dataset. We compare a data augmentation strategy from the literature with a novel one that densifies the dataset around the stationary points visited by a global optimization algorithm.

Our approach was validated using a relevant industrial case study: hydrogen purification. Validation by simulation is performed on the obtained solutions. The computational results show that a data augmentation smartly coupled with optimization can produce a robust and reliable design tool.

1. Introduction

Recent research represents membranes using machine learning models such as artificial neural networks (ANN) (Wang and Lin, 2021; Piron et al., 1997; Wessling et al., 1994; Richard Bowen et al., 2000; Himmelblau, 2000; Hamachi et al., 1999; Dornier et al., 1995a; Niu et al., 2022). One reason to represent membrane behavior with an ANN surrogate is that we may wish to integrate the ANN into a larger decision-making problem such as identifying manufacturing conditions. But ANNs may have difficulty extrapolating to new regimes (Schweidtmann et al., 2021). The challenge is that optimal decision-making problems (Kim and Boukouvala, 2020), for instance choosing process conditions holistically, may select regions where the trained ANN is inaccurate. This paper investigates data augmentation strategies, illustrated in Fig. 1 that, together with our optimization approach, give confidence in the result of the optimal decision making problem.

1.1. Membrane separation

Membrane gas separation by means of synthetic polymeric membranes is a well-established technology for several industrial applications such as production of nitrogen from air, hydrogen recovery from ammonia plants and refineries, natural and biogas treatment, and vapor recovery from vent and process gas streams (Baker, 2012). Among the numerous advantages brought by membrane processes, the most reported are a small environmental footprint due to a solvent free technology, a facilitated operability, an easy up and down scaling. Membrane separation processes are based on two pillars, membrane separation performance, derived from intrinsic material separation performance (selectivity and permeability) and process design, namely the choice of operating conditions and interconnections between the selected equipments. Membranes for gas separation are constituted by selective materials that have a selective permeation capacity with respect to different gases. Membrane processes are, for most of them, pressure driven separation processes. The fastest components (highest

* Corresponding author.
E-mail address: bernardetta.addis@loria.fr (B. Addis).

https://doi.org/10.1016/j.compchemeng.2023.108342
Received 25 January 2023; Received in revised form 17 May 2023; Accepted 26 June 2023
Available online 5 July 2023

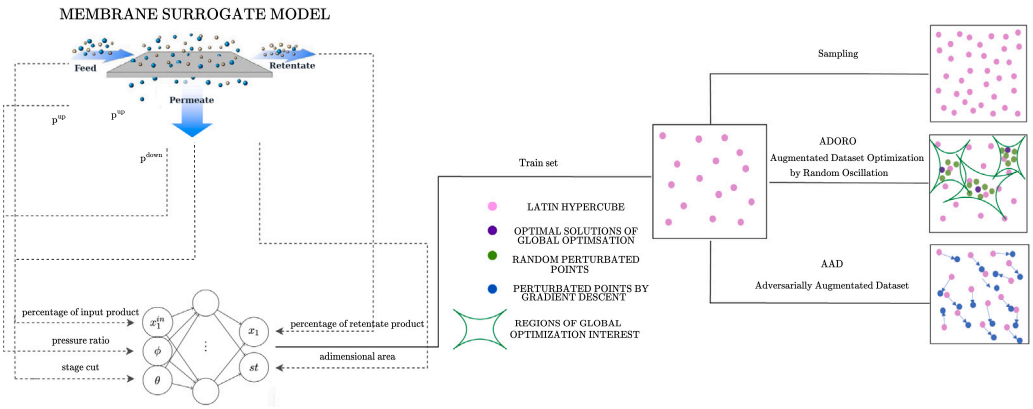


Fig. 1. This figure summarizes our approach: an ANN is used to model the membrane behavior trained on a small dataset. Then a smart data augmentation is used to improve the model in the appropriate regions.

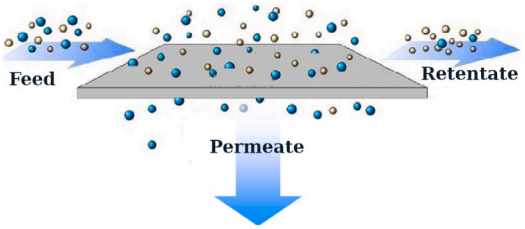


Fig. 2. Membrane schematic functioning.

permeabilities) are preferentially recovered on low pressure side (permeate) while the slowest ones are collected on the high pressure side (retentate) (Sridhar et al., 2014) (see Fig. 2).

The CO_2/H_2 separation chosen as a case study to illustrate the proposed methodology is of very high industrial interest. Numerous studies focus on membrane technology applied to H_2 purification and/or CO_2 capture from syngas produced from Integrated Gasification Combined Cycle or Blast furnace Gas in the steel industry. Both CO_2 and H_2 selective materials commercially exist that exhibit quite interesting selectivities (around 10–20) for both H_2 and CO_2 selective membranes (Merkel et al., 2012; Lin et al., 2014b; Arias et al., 2016; Lin et al., 2014a, 2015; Ritter and Ebner, 2007).

1.2. Optimal process configuration for multi-stage membrane separation

Separation performances, mainly purity and recovery rate for the targeted component, depend on different design choices including: material properties, membrane area, applied high and low pressures for each module. Considering the entire process architecture, several separation stages (membranes) may be necessary to increase the requested *purity*, recycling between stages (from permeate or retentate sides) can be useful and necessary to increase recovery rate. Multiple process configurations are possible for such a multi-stage membrane separation, i.e. number of membrane stages, type of membrane material per stage, membrane surface of each stage, upstream and downstream pressure of each stage and stream's connections between stages including partial or total recycling. The entire set of combinations can obviously not be assessed.

Optimal process design aims to explore the full set of meaningful configurations (Qi and Henson, 2000; Ohs et al., 2016; Uppaluri et al., 2004; Gabrielli et al., 2017; Arias et al., 2016; Scholz et al., 2015).

Ramírez-Santos et al. (2018) develop a nonlinear optimization formulation, and use a multistart optimization algorithm based on a combination of smart random generation and local search, an adaptation of Monotonic Basin Hopping (Wales and Doye, 1997; Locatelli and Schoen, 2013), to address CO_2/CH_4 separation. The Ramírez-Santos et al. (2018) optimization formulation is the starting point for our CO_2/H_2 separation model, which we solve with a multistart approach. However, we replace the membrane functioning equations used in Ramírez-Santos et al. (2018) with an ANN representing membrane behavior and thereby overcome the numerical difficulties arising from discretizing the differential equation.

1.3. Machine learning for representing membrane behavior

Machine learning has been widely used to predict different chemical processes and membrane gas separation (McBride1 and Sundmacher, 2018; Kajero et al., 2017; Henao and Maravelias, 2011; Fahmi and Cremaschi, 2012; Dornier et al., 1995b; Alam et al., 2022; Granacher et al., 2021; Khayet and Cojocaru, 2012; Liu et al., 2009; Richard Bowen et al., 2000; Schmitt et al., 2018; Yangali-Quintanilla et al., 2009; Di Pretoro et al., 2022). Prior work has represented physical phenomena with ANNs: these ANNs replace complex differential equations that may lead to numerical problems (Bhosekar and Ierapetritou, 2018; Cozad et al., 2014; McBride1 and Sundmacher, 2018; Rall et al., 2020b; Henao and Maravelias, 2011; Eason and Cremaschi, 2014; Mencarelli et al., 2020).

These papers typically choose bounds for the ANN input variables and generate sample points within these bounds (Ibrahim et al., 2018; Fahmi and Cremaschi, 2012; Henao and Maravelias, 2011; Rall et al., 2019, 2020a; Hu et al., 2021; Asghari et al., 2020), e.g. using a Latin Hypercube (Nuchitprasittichai and Cremaschi, 2012; Cozad et al., 2014; Eason and Cremaschi, 2014; Rall et al., 2020b; Bhosekar and Ierapetritou, 2018). Our paper uses the MEMSIC module (Bounaceur et al., 2017) to simulate a gas permeation membrane embedded into process simulation software. The sampled inputs and the simulator outputs create the dataset to train the ANN.

Obviously, the size of the training dataset is important (Satyanarayana and Davidson, 2005). Increasing the number of sampling points may increase the ANN accuracy, but it has many drawbacks: generating the data points via the simulator is computationally expensive, the network training becomes more time consuming, and the model itself becomes more complex.

1.4. Adversarial data augmentation

Data augmentation has received significant attention in the recent machine learning literature due to the increasing need of producing accurate datasets in reasonable time (Brendel et al., 2017; Papernot et al., 2017; Goodfellow et al., 2014; Antoniou et al., 2017; Volpi et al., 2018; Gao et al., 2019; Sinha et al., 2017; Gao et al., 2020; Magar et al., 2022; Zhou et al., 2020; Devabhaktuni and Zhang, 2000). In

chemical and process systems engineering, prior research (i) starts from a small dataset, (ii) integrates new points into areas of interest, and (iii) decides a set of stopping criteria (Eason and Cremaschi, 2014; Asghari et al., 2020; Garud et al., 2017; Tran et al., 2017). Several authors propose sampling-based approaches to automatically augment the data-driven ANN with additional input/output pairs in a region of interest (Eason and Cremaschi, 2014; Cozad et al., 2014; Nuchitprasittichai and Cremaschi, 2012; Garud et al., 2017; Tsymbalov et al., 2018). Adding points to the dataset is important because the membrane behavior is represented by the dataset used for training rather than by mathematical equations (Niu et al., 2022).

1.5. Paper contribution

In this paper, we use a machine learning model to represent the input/output membrane separation behavior. The resulting surrogate model is inserted into the mathematical programming model allowing us to explore the space of optimal process designs for our case study. The critical point is that the quality of the machine learning model influences the quality of the solutions. Indeed, any solution produced by the mathematical model needs to be simulated by MEMSIC to get the "real" performance of that design. The solution of the mathematical model has a predicted purity that always meets the requirements (or the problem would be infeasible), but if the purity estimate is wrong, then the simulation may produce a useless design. If the model was overestimating the purity, the purity requirements will not be satisfied in practice, and the solution must be discarded; if it was underestimating the purity, there may be a large error in the flows or in the areas which may result in too expensive designs.

As a first step, we build a large training dataset to get an accurate representation of the membrane behavior on the entire span of the membrane functioning. The resulting model can be used successfully in the optimization process, but the computational cost to obtain it is too high. Therefore, the entire process is too expensive to be extensively used in other case studies. To address this issue, we develop a data augmentation algorithm, *ADORO* (Augmented Dataset driven by Optimization by Random Oscillation), that guides the dataset generation in the regions around the stationary points found by our multistart optimization algorithm for different performances requests (purity and recovery). The core idea is that the model accuracy must be high not everywhere, but only around some regions of interest, namely, the regions around the local optimum points. Unfortunately, these regions are unknown beforehand, not even approximately. Indeed, in multistage systems, even the composition of the single stage cannot be determined apriori.

Prior work perturbs points close to a global solution, so our focus on multistart is effectively an acknowledgment that, in process operation, there may be limited variation around a few data points (Qin and Chiang, 2019; Thebelt et al., 2022). In our case, only perturbing around the global solution leads to generating too few additional data points and only focusing on solutions that, after scrutiny, turn out to be non-robust. Considering different operating conditions improves both the quality and the robustness of the machine learning model.

We compare the new algorithm ADORO with more traditional adversarial augmented data (AAD) approaches from the machine learning literature (Papernot et al., 2017; Volpi et al., 2018). These adversarial data augmentation techniques are mainly used in image recognition, natural language processing, and computer vision. The AugLiChem (Magar et al., 2022) library has been developed for chemical structures: AugLiChem perturbs crystals through random perturbation, axis change, random rotation, etc.

The rest of the paper is organized as follows: in Section 2 we detail the machine learning model built starting from the large reference dataset. In Section 3, we describe two data augmentation techniques, one from the literature and our proposal (ADORO), and in Section 4 we discuss the obtained results. Finally, we draw conclusions in Section 5.

2. Machine learning model

2.1. Neural network

Our machine learning model mimics the stand-alone Fortran version of the MEMSIC simulator described in Fig. 3(a) and Appendix A.

We capture the nonlinear input–output relation using a shallow, multi-layer perceptron network, with one layer of neurons and sigmoid activation. This simple network has useful theoretical approximation properties (Pinkus, 1996). Additionally, because the trained network is then inserted into an optimization problem as equality constraints, the single layer keeps these constraints simple and limits the nonlinearities. Thanks to the simple architecture (see Fig. 3(b)), the only hyperparameters are the number of neurons in the hidden layer, and the batch size.

2.2. Reference dataset generation

We first build a reference model using a large number of points, neglecting the computational cost, aiming only at a high accuracy on the entire input space. This step has two aims: showing that the chosen architecture allows to accurately represent the membrane behavior, and setting a target accuracy to reach with a smaller dataset.

Our model has three inputs: (i) ϕ pressure ratio $\left(\frac{P^{down}}{P^{up}}\right)$(ii) x_{in}^1 feed molar fraction of gas 1 (iii) θ stage cut ratio between the permeate and the feed flowrates, see Eq. (A.2). We recall that the overall design may include several membranes, therefore we cannot know in advance the operating conditions of each membrane. Thus we need that a high model accuracy in the entire input space. Drawing inspiration from the literature (Eason and Cremaschi, 2014; Bhosekar and Ierapetritou, 2018), we considered three different dataset generations: grid generation, Latin hypercube and uniform random generation. The following input variable bounds allow us to cover the region of interest for the case-study:

$$\begin{aligned} 0.1 &\le x_1^{in} \le 0.98 \\ 0.002 &\le \phi \le 0.65 \\ 0.1 &\le \theta \le 0.9. \end{aligned}$$

To choose the dataset size, we started with grid generation and sampled each interval's variable with 70 points, resulting in a dataset of $70^3 = 343{,}000$ points. Then, we generated the same number of points with the Latin hypercube technique and the uniform random distribution. All the input and output variables range between 0 and 1 except the output st which ranges, for our dataset, between 0.1 and 26.4. We normalize st by dividing its values by 10.

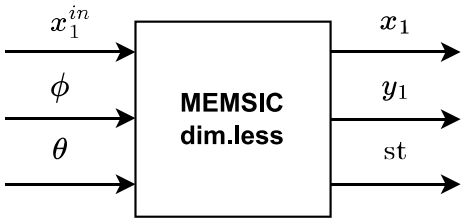
2.2.1. Data cleaning by an SVM classifier

No matter the technique used to generate the dataset, there are input points where the simulator cannot converge, particularly at the boundary values of the input and output variables. For example, there exist input concentration (x_1^{in}) and pressure ratios (ϕ) where it is not possible to obtain a given stage-cut (st). Fig. 4 illustrates in **dark red** the points in the 2-dimensional grid of ϕ and θ where the simulator fails for input gas concentration $x_1^{in} = 0.049$ and $x_1^{in} = 0.6$.

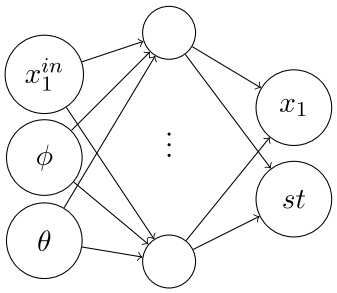
These points must be removed from the dataset.

To identify the problematic area, we built a binary classification problem, where we assign label equal to 1 to all the grid points where the simulation fails, and zero to the others. Then we build a separation hyperplane by training a linear support vector machine (SVM) (Piccialli and Sciandrone, 2022) for separating the two classes, getting an F1 score equal to 92.4% on the grid dataset. Not only do we want to avoid the points where the simulator will not converge, but also the neighborhood of these points where the simulation may not be reliable due to accuracy issues.

To avoid these physically meaningless domains, we require the points in all the generated dataset to be below the hyperplane with

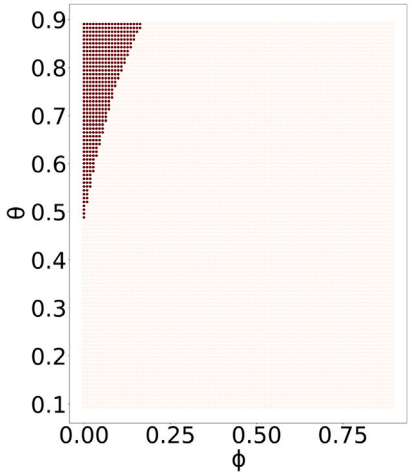


(a) MEMSIC standardized simulator in/out scheme

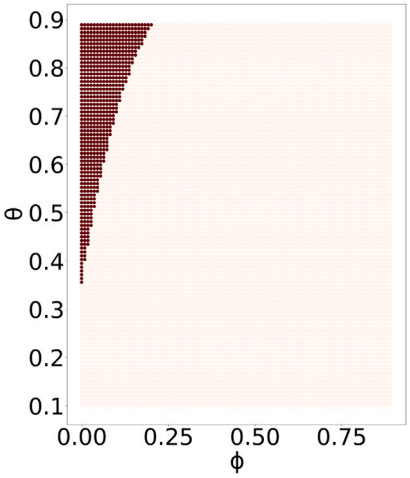


(b) ANN structure: three inputs, 1 hidden layer and two outputs

Fig. 3. The ANN has the same inputs of the standardized dimensionless simulator. Inputs: (i) ϕ pressure ratio $\left(\frac{p^{down}}{p^{up}}\right)$ (ii) x_{in}^1 feed molar fraction of gas 1 (iii) θ stage cut ratio between the permeate and the feed flowrates, see Eq. (A.2). The outputs of the ANN are (i) x_1 concentration of gas 1 in the retentate (ii) st dimensionless area, whereas y_1 is derived by the mass conservation equation (A.10).



(a) Projection on $x_1^{in} = 0.049$.



(b) Projection on $x_1^{in} = 0.60$.

Fig. 4. The points highlighted in **dark red** are where the simulator cannot converge. Note that ϕ is the pressure ratio and θ the stage-cut.

a certain tolerance, to be sure that we stay in a region where the simulation is reliable. We also use the SVM hyperplane as a constraint in our optimization model to be sure that our feasible region does not include meaningless points.

2.2.2. Datasets comparison

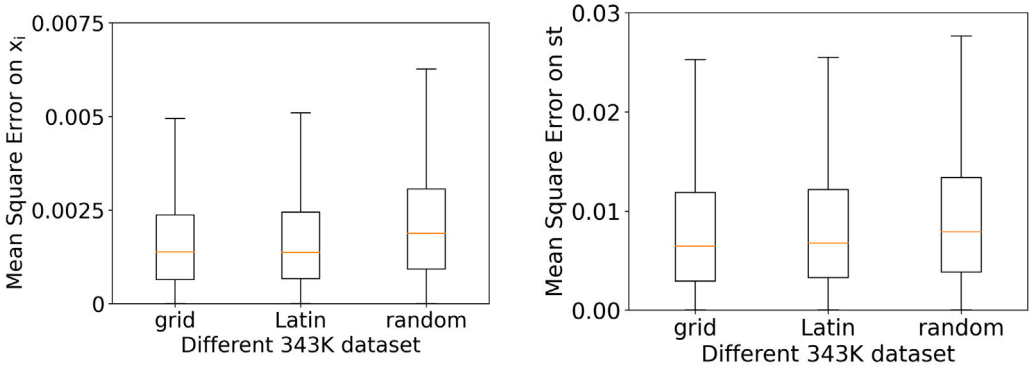
We train the neural network for all three datasets obtained after the SVM procedure and evaluate the resulting model. We use ADAM as an optimizer with the default setting, set a maximum number of epochs equal to 20 000, without early stopping, and use a random weights initialization.

For each dataset, we perform a 5-fold validation grid search on the following values of the hyperparameters batch size (BS) and the number of neurons (NN): $BS \in [256, 512]$ and $NN \in [32, 64, 128, 256, 512]$. We use as loss of the mean square error (MSE), and the results on the average MSE were comparable on all three datasets. For all the datasets, the best configuration was with $BS = 256$ and $NN = 64$ with similar values of MSE. To get a better measure of the model quality, we look into the absolute error distribution of the output variables on the training set, since we aim to perform well on the whole input space. Fig. 5 shows that the Latin hypercube dataset performed best, having the lowest absolute error quartiles on the output variables. This is the most important error measure for the case study because we need high accuracy on the concentration x_1 and the area st for every feasible input. Note that the model built for a single membrane is then embedded in a multistage design. This implies that the operating conditions of each stage that allow reaching the required performance in the multi-stage system depend on the entire design and hence cannot be known in advance. Therefore, to test the behavior of the model in the interesting areas, we need to evaluate the accuracy when embedded in the optimization model with more than one stage.

The variables and constraints in the model represent not only the operating conditions of each stage, but also the interconnections between the stages, according to a superstructure representing all the allowed possible configurations (Ramírez-Santos et al., 2018). Appendix B describes the complete optimization model, i.e. the flow conservation constraints, both at the system level and the stage level, membrane functioning constraints, performance requirements, and the objective function. Clearly, we are interested in an accurate model not only in terms of performance but also in terms of amount of flows. The flows can be either at the system level (input flow, retentated, and permeated flow in output) or internal: internal flows are the ones going from one membrane to another, or in self-loop to the membrane itself.

To have a deeper understanding of the performances of the models generated by means of the three datasets, we run a multistart algorithm on the resulting optimization models with 150 iterations, i.e. we run 150 local searches starting from randomly generated initial points. We use Knitro (Byrd et al., 2006) as local solver, and generate randomly in the original variables the starting points following Ramírez-Santos et al. (2018).

It is known that the differential Eqs. (A.1) are numerically difficult for high x_1 purity (Neveux et al., 2022), so, we want to verify that the neural network has good accuracy for this case. Our goal is to produce an output satisfying some purity and recovery constraints with



(a) Box plot x_i variable

(b) Box plot *st* variable

Fig. 5. Box plots of the absolute errors on the two output variables with $BS = 256$ and $NN = 64$ on the 343k datasets obtained with regular grid, Latin hypercube and uniform random generation.

a multi-stage system of CO_2 selective membranes. We consider three levels $(90\%, 95\%, 99\%)$ of purity with recovery as high as possible for a two stage system. The requested recovery is 99% for all purity cases, with the exception of the 99% purity. Indeed, when a 99% purity is requested, a two-stage system allows recovery of at most 95%.

For each combination of performance requirements, we perform a multistart optimization procedure, and simulate using MEMSIC the best solution found, i.e. take the design obtained corresponding to the best solution (membranes area, applied pressures, split flows percentages) and use a MEMSIC simulation to re-calculate flows and gas concentrations. The following analyzes the difference between the best solution found and its simulated counterpart. Table 1 reports:

simulated purity the purity of the solution simulated by MEMSIC fixing the best design obtained from optimization. This allows us to verify whether the purity guessed by the machine learning model is actually achieved in simulation. We stress that since the optimal solution is feasible for the optimization model, the purity requirement is always satisfied according to the surrogate model. However, when the solution is simulated it can happen that the purity was overestimated or underestimated. If the purity is overestimated then in practice the actual solution does not satisfy the purity requirement and must be discarded.

H_2 absolute error the absolute error in the retentate and permeate outputs for the hydrogen concentration

Flows (% error) relative errors on retentate and permeate output flows.[1]

The neural network trained by using the dataset generated by the Latin hypercube strategy is the only one that actually achieves the required purity in the three scenarios. Furthermore, the percentage error on the flows is reasonably low. The grid generated dataset never reaches the required purity and produces a higher error on the flows. No matter what machine model is used in the optimization process, there is a huge improvement in terms of efficiency and numerical reliability with respect to the discretized model of Ramírez-Santos et al. (2018). The local optimization by Knitro is at least 10 times faster, and the number of infeasible runs becomes negligible.

But Table 2 shows that, despite the final model being accurate and reliable, the time needed to obtain the model (cross-validation and training) is very high (>5 days). For this reason, we wish to reduce the dataset size without reducing the quality of the model. As the Latin hypercube generation is the more performing strategy, in terms of quality and computational time, we retain this dataset generation strategy for all the following experiments.

The model built by using the 343k dataset generated by Latin Hypercube will constitute our target in terms of accuracy, aiming at reaching a similar performance with a much smaller training dataset.

Table 1
For each combination of purity and recovery, we report the optimization performances obtained using the three different surrogate models (using grid, Latin, and random generation). The quality is assessed by comparing the discrepancy between the best solution's system outputs obtained by the surrogate model and the ones obtained by the MEMSIC simulator. We report the simulated purity and the percentage errors on the (permeated and retentated) flows. We highlight in bold the best performances: smallest discrepancy for outputs, achieved purity target. We note that the Latin surrogate is the only one that allows to meet the purity requirements after the simulation in all cases.

343k dataset				
		Grid	Latin	Random
Optimization constraints: 90% purity, 99% recovery				
	Simulated purity	0.884	**0.903**	0.896
H_2 (abs error)	ret	0.016	**0.003**	0.004
	perm	**0.009**	0.012	0.018
Flows (% error)	ret	1.27	1.00	**0.65**
	perm	2.56	1.89	**1.23**
Optimization constraints: 95% purity, 99% recovery				
	Simulated purity	0.942	**0.958**	**0.951**
H_2 (abs error)	ret	0.008	0.008	**0.001**
	perm	**0.006**	0.012	0.017
Flows (% error)	ret	**0.48**	1.64	1.29
	perm	**0.79**	2.62	2.07
Optimization constraints: 99% purity, 95% recovery				
	Simulated purity	0.989	**0.990**	0.989
H_2 (abs error)	ret	0.001	**0.000**	0.001
	perm	0.004	**0.003**	0.004
Flows (% error)	ret	0.45	**0.21**	0.47
	perm	0.62	**0.28**	0.65

Table 2
For the dataset of size 343k generated by three different techniques we report: (i) Dataset size after SVM filtering and (ii) overall computational time (cross-validation + training). The data generation time is around 11 h, that have to be added to the overall computational time.

	Grid	Latin	Random
Dataset size	340 912	341 042	340 990
Time (h)	173.6	123.7	196.7

[1] For permeate flow G with G_{sim} obtained in the simulation and G_{opt} obtained by the optimization, define: $|G_{sim} - G_{opt}|$ as absolute error and $\frac{|G_{sim} - G_{opt}|}{1+G_{sim}} * 100$ as relative error.

Fig. 6. To define a good solution our KPIs are : (i) the objective function (minimize) (ii) dimension of the dataset (minimize) (iii) achieve 99% purity on the product (in our case 99% per cent H2 in the retentate) (iv) time of the procedure (minimize) (v) error on each flow (e.g. the flow passing from one membrane to another etc.) (minimize).

3. Data augmentation

Surrogate models have the advantage of reducing computational time in the optimization phase, as they lead to simplified models. Nevertheless, to be effectively used in an optimization strategy, a surrogate model needs two key elements: a high quality in terms of accuracy and a reasonable overall "creation" time. High accuracy is necessary for a large span of the variables to obtain a correct description of the optimization problem (feasible region and objective function shape). It is not possible to predict which search space points will be sampled by the optimization method (directly, or along the standard local search procedures), and a wrong evaluation of the constraints (and/or objective function) on such points can lead the optimization to the "wrong" search space portion, preventing the global optimization method to find the best local optimal solutions. The computational time needed to produce the surrogate model must be comparable (or smaller) than the time gained moving from an equation based model to the surrogate one.

The results presented in the previous section empirically demonstrate that an accurate surrogate model can be produced employing an ANN and data obtained through simulation. The goal of the rest of the paper is to investigate different data augmentation techniques to maintain (or improve) the accuracy, reducing drastically the overall time needed to calibrate the ANN. Indeed, data augmentation techniques allow densifying the dataset only in some regions, thus obtaining high quality models with a reduced dataset. To assess the quality of the process and the produced models, we will focus on several quality indicators, as shown in Fig. 6.

In the first step, we will focus on finding a model that is accurate enough to obtain using a Multistart-like global optimization method solutions that lead to feasible solutions in the simulation validation procedure.

We know that with 343k points we are able to achieve the target accuracy, but we want to keep the computational time low. For this reason, we aim to build a dataset of at most 125k points allowing us to reach a comparable model quality. To this aim, we have different possibilities: try datasets of different sizes up to 125k, or start with a smaller dataset, and use data augmentation techniques to improve the obtained model. In the latter case, we need to decide the size of the initial dataset and/or the number of iterations of data augmentation. By using Latin hypercube, we generate datasets of different sizes that we call data_{ini}. Our idea is to exploit the quality of the current model, to add points in the area where the model is not good enough.

We propose two different strategies, the first one based on the literature on adversarial data augmentation, the second one is new, and exploits our optimization procedure, see Fig. 1:

- **AAD** (Adversarially Augmentated Dataset)
  We add to the training set a perturbation of each point of data_{ini} in the direction of the gradient descent of the loss function.

Table 3
Features of the baseline (343k) and initial datasets data_{ini}. We report the size for (i) the initial dataset; (ii) the final dataset, i.e. after the SVM-filtering procedure; the time in hours for (i) the generation of the dataset; (ii) for the cross-validation (iii) training of the best model found with the cross-validation.

	10k	20k	50k	125k	343k
Initial size	10 000	20 000	50 000	125 000	343 000
Final size	9957	19 886	49 735	124 282	341 042
Computational times (h)					
Generation	0.33	0.66	1.64	4.08	11.2
Cross-val.	3.5	7	16	45	123.7
Training	0.15	0.19	0.34	1	3

- **ADORO** (Augmentated Dataset driven-by Optimization by Random Oscillation)
  We add to the training set a random perturbation of the stationary points produced by the multistart optimization procedure in different interesting scenarios.

We need to decide the size of the initial dataset, generated by means of Latin hypercube. To this aim, we generate datasets of different size, whose characteristics are reported in Table 3 and we label them according to their initial size. The first two rows report information related to the size of the datasets: the size the original random generation and the number of residual samples after filtering the points using the SVM-generated hyperplane (to remove the numerical instability points of the simulator). The last three rows report the computational times (in hours) of the different phases of the model generation: dataset generation by means of simulation, cross-validation and final training.

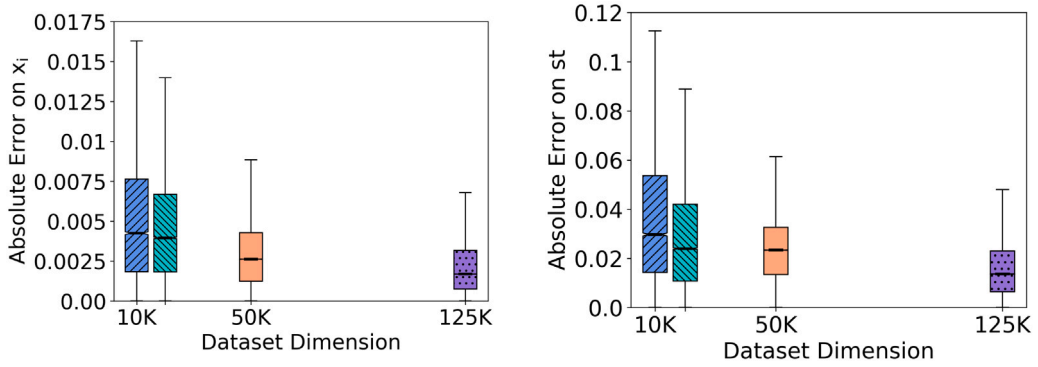
We retain the same procedure used for the baseline dataset for the cross-validation. The best choice turns out to be 64 neurons and BS = 256 for each dataset.

Not surprisingly, given a number of neurons and a batch size, the largest dataset present a smaller training error, but still significantly higher than the one achieved with 343k points. However, if we look at the computational time, training the model with 125k is still too expensive, see Table 3.

In Fig. 7, we visualize the absolute error of the two output variables (x_i and st) in the form of box plots. The error slightly reduce for both variables passing from 10k to 20k. Not surprisingly, a larger improvement is observed for the case of 50k and 125k. It is worth to notice that the reduction passing from 50k to 125k is less significant than the one obtained from 20k to 50k, if compared with the increase in size of the dataset.

Finally, we evaluate the solutions' quality as in Section 2.2.2 : we use the obtained surrogate models in the optimization procedure and we compare the solutions in term of accuracy with respect to the simulation in Table 4.

We can observe that the models obtained using 10k and 20k datasets are not accurate enough to obtain an optimal solution that satisfies the



(a) Box plot x_i variable

(b) Box plot *st* variable

Fig. 7. Box plots of the absolute errors on the two output variables with $BS = 256$ and $NN = 64$ on the sampling datasets with 10k, 20k, 50k and 125k.

Table 4
We compare the best solution found by the optimization with its simulated counterpart in terms of system outputs. When we use the surrogate models built with less than 50k points (10k and 20k), the simulated counterpart never reaches the required purity, whereas when we use the 50k dataset, the simulated counterpart is always feasible, but the purity is underestimated for performance requirements equal to 95% and 99%. On the other hand, using the 125k data set allows us to obtain a simulated feasible solution only for purity equal to 99%.

Evaluating via simulation the accuracy of the model on the best solution found by optimization. Different size datasets (Latin hypercube sampling)					
		10k	20k	50k	125k
90% purity, 99% recovery					
	Simulated purity	0.897	0.891	**0.900**	0.890
H_2 (abs error)	ret	0.003	0.009	**0.000**	0.010
	perm	0.022	0.015	0.019	**0.003**
Flows (% error)	ret	0.96	**0.08**	1.12	6.32
	perm	1.80	**0.15**	2.10	6.32
95% purity, 99% recovery					
	Simulated purity	0.949	0.944	**0.953**	0.948
H_2 (abs error)	ret	**0.001**	0.006	0.003	**0.001**
	perm	0.026	0.016	0.019	**0.003**
Flows (% error)	ret	1.63	0.38	1.64	**0.33**
	perm	2.61	0.62	2.62	**0.34**
99% purity, 95% purity					
	Simulated purity	0.988	0.989	**0.991**	**0.993**
H_2 (abs error)	ret	0.002	**0.001**	**0.001**	0.002
	perm	**0.003**	0.012	0.005	0.011
Flows (% error)	ret	0.49	1.06	**0.29**	0.59
	perm	0.68	1.48	**0.40**	0.81

requested purity constraint. It is worth noticing that, in terms of flows, the smaller dataset can perform better than the larger ones.

These results seem in contradiction with the quality presented in the box plots, but it is not the case. Indeed, the overall quality of the model is not enough to guarantee good performance of the overall optimization strategy, in particular in terms of the feasibility of the obtained solution. Small errors in the prediction of two output variables can impact strongly the shape of the feasible region, shifting the "real/original" local optima (full equation/simulation) or even removing them. The prediction error on variable *st* (the standardized area) results in an amplified error on the membrane area, which is computed by Eq. (A.4). The area impacts directly the objective function, and in our case study, the order of magnitude of the overall conversion factor is 10^4, so that a small error in *st* translates into a significant error in the predicted area. Furthermore, the combination of 2 (or more) membranes leads to an error propagation that is difficult to anticipate (as the overall design is decided by the optimization procedure). These considerations lead us to the study of methods that instead of working with a larger starting dataset, increase it selectively after evaluating its quality.

The computational time of 125k is already too high: around 45 h for the cross-validation of the 125k dataset. Furthermore, this time is not compensated by better results in the optimization phase. Thus, we selected as starting dataset for data augmentation only the 10k, 20k, and 50k. The two methods that we investigate are a "classical" AAD and a new strategy (ADORO) that aims at combining the information obtained by the optimization into the dataset augmentation procedure. In what follows we describe the two techniques and report the results we obtained with these two procedures starting from different data_{ini} and with a different size of the augmentation step.

3.1. Adversarial data augmentation

A quite common technique to improve the accuracy of the network (i.e. the training error) is adversarial data augmentation. The main idea is to enrich the dataset with a perturbed version of the input. The perturbation is aimed at introducing samples where the initial model produces a significant prediction error. The procedure for the AAD algorithm is shown in Fig. 8. We perturb each point of data_{ini} by using the gradient of the loss multiplied by a scalar factor α. The new points are calculated as follows:

$$z_i^{new} = z_i + \alpha \nabla \mathcal{L}(u_i, z_i) \quad i \in \text{data}_{\text{ini}} \tag{1}$$

where z_i is the ith vector of input values in data_{ini}, u_i is the corresponding output, and $\mathcal{L}$ is the loss function.

The parameter α influences the performance of AAD. We performed a grid-search on parameter α in the interval $[0.01, 0.5]$ and we retained the value that produced the best trade-off between training error and training time: $\alpha = 0.4$.

The AAD procedure is an iterative one: the initial dataset is used to obtain a starting model, then the dataset is augmented by the new points (1), the new model is trained, and the augmentation procedure is repeated until some stopping condition is met. The AAD procedure doubles the number of points of the dataset at each iteration. We have two stopping criteria: training_error $\leq 1.0e^{-6}$ or len(data_ini) $\geq$ 100K. The stopping criteria are verified at each training step performed after the data augmentation. For all the datasets, the procedure was stopped by the limit in the size of the final dataset. We refer to the final datasets as 10k+, 20k+, and 50k+.

Fig. 9 shows the box plot for the prediction errors of the ANN on the two output variables, x_1 and *st* for the 3 new datasets generated by the AAD algorithm (starting from 10k, 20k, and 50k respectively). It is worth noticing that the 50k even with a larger number of samples,

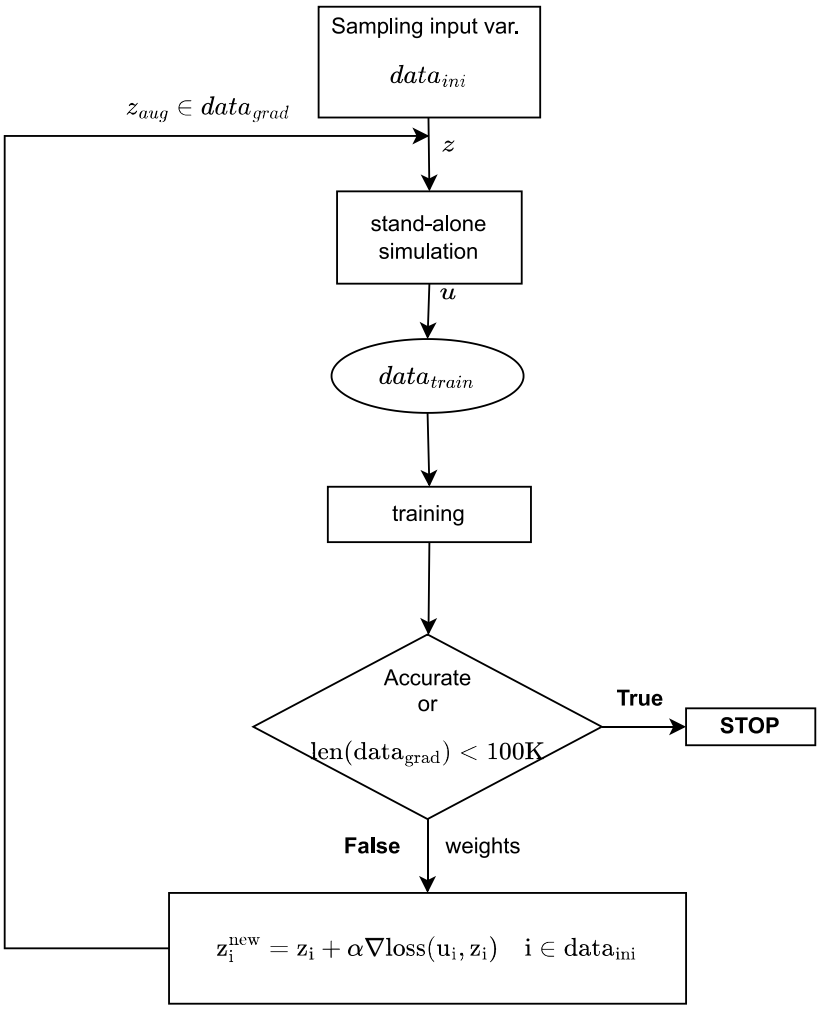
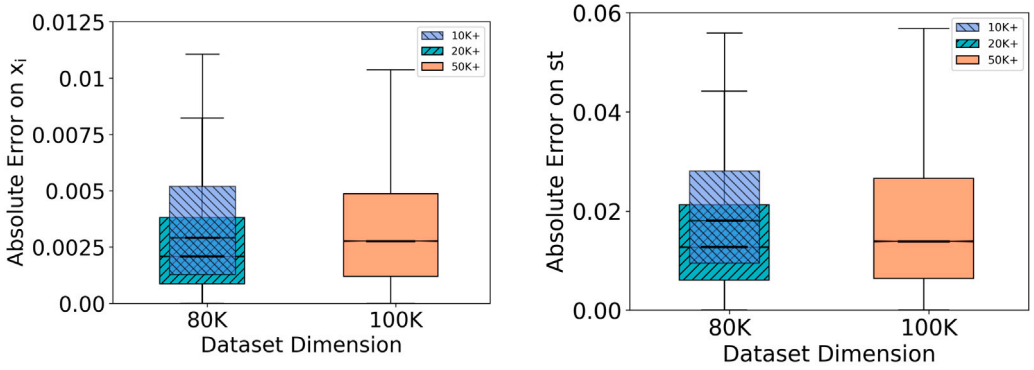


Fig. 8. Schematic view of the AAD algorithm. The size of the dataset is doubled at each iteration by adding a perturbation along the loss gradient of every point in the dataset.



(a) Box plot x_i variable

(b) Box plot *st* variable

Fig. 9. Box plots of the absolute errors on the two output variables with $BS = 256$ and $NN = 64$ with 10k+,20k+ and 50k+.

cannot reach the performance of the 20k dataset. The dataset 20k shows the best performances for both variables.

In Fig. 10 we report the details of the computational time for each iteration, including the initial phase:

- **gen**: time needed to generate data_{ini} (10k,20k and 50k)
- **cv**: cross-validation to define the best hyperparameters for the ANN
- **train**: training of the best ANN on data_{ini}.

For each loop we distinguish:

- **gen grad**: time needed to generate the new points, compute the loss and simulate the new points
- **retrain**: the training time for the augmented dataset $\text{data}_{\text{grad}}$, starting from the ANN weights obtained in the previous **train** phase.

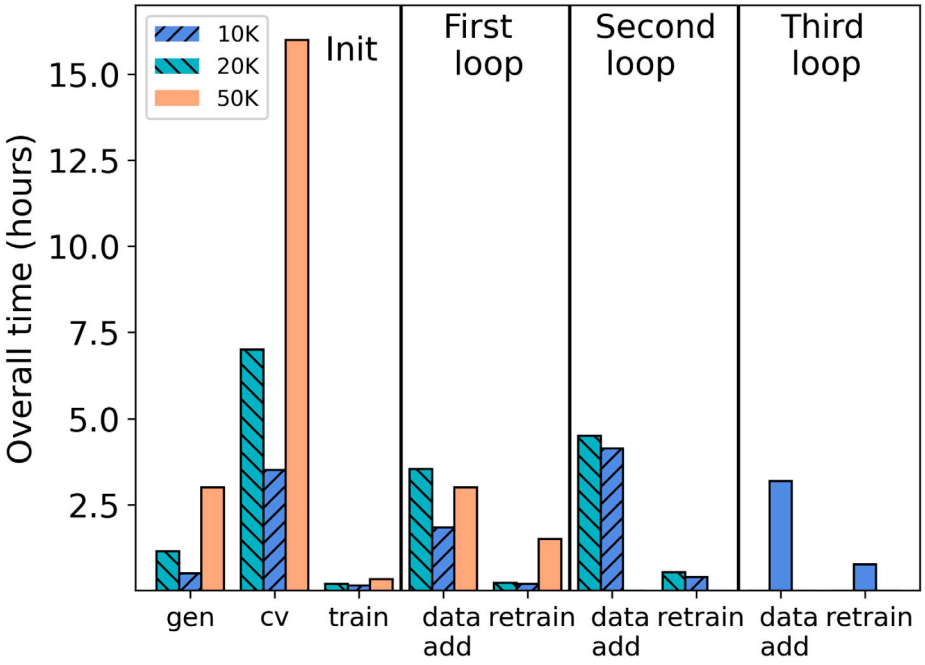


Fig. 10. All the steps involved in AAD procedure for each of the three $data_{grad}$. They represent respectively the new dataset 20k+,10k+ and 50k+. The one where we spend the smallest amount of time is the 10k dataset, because although it is the one for which we have the larger number of loops, it is the one where the initial step is cheaper.

The number of iterations depends on the size of the initial dataset. The overall computational time of the procedure is comparable for the 3 cases, the cross validation being the most expensive phase. The AAD phase takes almost the same time as the building phase for 10k and 20k. The re-training phase is a cheap one thanks to the warm start from the previous ANN weights.

We proceed to evaluate the accuracy of the method when embedded in our optimization procedure. We report here the case for the higher requests in Table 5.

In all the cases the optimization underestimates the purity, leading to solutions that are feasible (but we will see in the final comparison that are very expensive). In particular, the underestimation error of the 50k+ is 3 times larger than the one obtained by the 10k+ and the 20k+. The 10k+ performs better in terms of flows than the 20k+.

3.2. ADORO

The AAD procedure aims to improve the model everywhere considering the relative error obtained on each sample.

The main idea of ADORO is to enlarge the dataset on the basis not only of the information on accuracy but also by exploiting the information provided by the optimization procedure. Indeed, we are interested in having an accurate model in the "interesting" areas: the regions explored by the optimization algorithm.

The main idea is to add samples only in the regions explored by the optimization procedure but only where the surrogate model is not accurate enough with respect to the results obtained by simulation (see Fig. 11).

Given the initial model, ADORO performs the following steps:

S1 the multistart optimization procedure is run for different combinations of purity and recovery to span areas of interest in relevant scenarios,

S2 all the stationary points found by the procedure are then simulated, and they are kept only if there exists a significant discrepancy between the output obtained by the surrogate model and the one obtained by the simulation, (meaning that an improvement of the model is needed in that area)

Table 5
The simulated counterpart of the solution produced by AAD data augmentation meet the purity requirements when the starting size of the dataset is 50k, apart from the 90% case. However, in general the purity is underestimated by the models, resulting in simulated solutions with purity higher than requested. Overall, the AAD technique does not provide a better model than just sampling 50k points with a Latin hypercube strategy.

Evaluating via simulation the accuracy of the model on the best solution found by optimization AAD for different sizes of the starting dataset				
		10k+	20k+	50k+
90% purity, 99% recovery				
	Simulated purity	0.898	0.894	0.899
H_2 (abs error)	ret	0.002	0.006	**0.001**
	perm	0.004	0.013	**0.001**
Flows (% error)	ret	**0.03**	0.07	0.25
	perm	**0.05**	0.13	0.49
95% purity, 99% recovery				
	Simulated purity	**0.963**	**0.953**	**0.955**
H_2 (abs error)	ret	0.013	**0.003**	0.005
	perm	**0.003**	0.011	0.006
Flows (% error)	ret	1.54	1.02	**0.92**
	perm	2.47	1.66	**1.49**
99% purity, 95% recovery				
	Simulated purity	**0.991**	**0.991**	**0.993**
H_2 (abs error)	ret	**0.001**	**0.001**	0.003
	perm	0.005	0.011	**0.002**
Flows (% error)	ret	0.53	1.03	**0.48**
	perm	0.71	1.36	**0.64**

S3 a given number of points is obtained by perturbing each retained stationary point and added to the dataset

S4 the model is retrained on the augmented dataset.

We stress some interesting aspects:

- we consider multi-stage systems, thus each stage (membrane) can operate in different conditions. Therefore for each stationary

Table 6
This table shows for each PURITY-RECOVERY combination the number of locally optimal solutions found via multistart. The more stringent the performance constraints are, the more difficult it is to find optimal solutions.

	10k	20k	50k
90%–90%	16	23	36
90%–95%	2	19	11
90%–99%	4	2	10
95%–90%	11	4	9
95%–95%	4	2	3
95%–99%	2	1	5
99%–90%	4	5	10
99%–95%	5	2	2

point, we simulate and check the accuracy of each membrane composing the system, which translates into twice the number of stationary points since we use two stages in the experiments;

- The pool of stationary points produced by the multistart algorithm does not contain enough points to influence the ANN training, see Table 6, where we report the number of distinct local optima found for each purity/recovery combination and for each starting dataset (10k, 20k, 50k). In order to produce a robust augmentation procedure, we consider a large number of scenarios in terms of purity and recovery constraints. This should differentiate the points used as starting points for perturbation.

More formally, our augmentation procedure works as follows. Let Q be the set of stationary points obtained by the multistart procedure. Consider a point $q \in Q$, and define u_q^{opt} and u_q^{sim} as the vectors of the output variables obtained respectively by the optimization and the simulation for the input values corresponding to q. For each point q, we calculate the discrepancy between the optimization and the simulation:

$$d_q = \sum_{k=1}^{2} |u_{q,k}^{sim} - u_{q,k}^{opt}| = |(x_q^1)^{sim} - (x_q^1)^{opt}| + |st_q^{sim} - st_q^{opt}| \quad (2)$$

If $d_q > \varepsilon$ (in our experiments $\varepsilon = 1.0e^{-6}$), we add to the augmented dataset point q and a pool of randomly generated points around it.

The augmentation procedure may be repeated several times until a termination condition is met. However, in our experiments, a single iteration was enough to reach a good-quality model.

A parameter of the procedure is the number of points added at each iteration. We performed tests with 5k, 10k, and 20k points for all three initial datasets. The dimension of the augmented datasets is reported in Fig. 12(a). We name the datasets by a combination of the size of the initial dataset and the size of the added samples, for example, the 20 + 10k dataset uses the 20k data_{ini} and adds 10k samples in the augmentation phase. In all the figures and tables, we report the datasets in order of magnitude, from the smallest to the largest.

It turns out from the experiments, that the data_{ini} equal to 50k leads to disappointing results since the model obtained by ADORO has similar performance to the initial dataset and underestimates constantly the actual purity of the solution leading to unnecessarily complicated configurations. Furthermore, it was already less interesting given the larger size of the dataset. For this reason, we omit the results on data_{ini} equal to 50k.

In Fig. 13, we report the box plot for the training error for the six datasets produced by ADORO. For the output variable x_1, only the datasets starting from 10k show a clear improvement thanks to the addition of points. The 20k case is oscillating (reduces for +10k and increases for +20k). For the output variables st, both the 10k and 20k seem to benefit from the data augmentation.

In Fig. 12(b) we report the training time. It is worth noticing that the time is not monotonously increasing in the number of samples. This phenomenon is probably amplified by the warm start procedure used in ADORO. The 10k dataset requires always less training time than the other cases, independently of the number of added samples.

In Fig. 14, we report the times of the different phases of ADORO:

Table 7
The columns are in ascending order with respect to the size of the final dataset. It turns out that the simulated counterparts of the ADORO solutions starting from 10k and 20k have better performance than the ones starting from 50k points. The simulated counterparts obtained by ADORO meet purity requirements. In the end, ADORO produces high quality solutions with a dataset of size at most 50k.

Evaluating via simulation the accuracy of the model on the best solution found by optimization ADORO: different sizes of the starting dataset and the increment step		10+5k	10+10k	20+5k	20+10k	10+20k	20+20k
90% purity, 99% recovery.							
Simulated purity		**0.901**	**0.908**	**0.904**	0.899	**0.904**	0.894
H_2 (abs error)	ret	**0.001**	0.008	0.004	**0.001**	0.004	0.005
	perm	0.023	0.018	**0.006**	0.018	0.021	0.023
Flows (% error)	ret	1.47	2.05	1.59	0.99	1.68	**0.84**
	perm	2.73	3.75	2.95	1.86	3.10	**1.59**
95% purity, 99% recovery.							
Simulated purity		**0.951**	**0.957**	**0.955**	0.949	**0.953**	0.946
H_2 (abs error)	ret	**0.001**	0.007	0.005	**0.001**	0.003	0.003
	perm	0.025	**0.019**	0.020	0.020	0.022	0.020
Flows (% error)	ret	1.74	2.12	1.85	1.18	1.78	**1.22**
	perm	2.77	3.34	2.93	**1.91**	2.83	1.97
99% purity, 95% recovery.							
Simulated purity		0.988	**0.991**	**0.993**	**0.991**	**0.990**	0.987
H_2 (abs error)	ret	0.002	0.001	0.003	0.001	**0.000**	0.002
	perm	0.003	0.002	0.006	0.006	0.005	**0.001**
Flows (% error)	ret	**0.01**	0.02	0.14	0.41	0.38	0.14
	perm	**0.01**	0.03	0.19	0.55	0.51	0.19

1. **gen**: time for the generation of the data_{ini}
2. **opt+gen**: optimization phase (1000 Multistart iterations)
3. **per+gen**: generation of the new points (perturbation of the selected local optima)
4. **retrain**: retraining of the neural network by using the augmented dataset, starting from the previous weights' values.

We recall that the initial data generation, the cross-validation, and the first train are common to all procedures (pure sampling, AAD, and ADORO).

For the datasets from 20k, only the case with +20k needs a time larger than around double the generation (1.98 h). We recall, thou, that this time is still significantly smaller than the cross-validation time (7 h) and thus of the overall initialization phase (around 8 h). For the case of 10k, the case with +20k needs more than three times the initial generation (1.6 h) but is still much smaller than the initialization phase (around 4 h).

We can now evaluate the quality of the solutions of the optimization procedure in the different scenarios, see Table 7.

The ADORO strategy starting with the 10k dataset always produces feasible solutions apart from the model 10 + 5k that fails for the purity 99% (error of 0.002). When starting with the 20k dataset the performances slightly deteriorate, mainly when adding 20k points.

In some cases, the purity is underestimated, resulting in solutions with higher purity than requested. The model showing the best performance is 10 + 20k since it always produces a feasible solution without underestimating too much. The model 20 + 10k on the other hand is very close to feasibility, and very accurate where it reaches feasibility, being an interesting alternative.

4. Discussion

We now summarize the experimental results using some global metrics to compare the two proposed augmentation strategies. We chose three indicators to evaluate the effectiveness of the method by means of the quality of the solutions: the overall cost and the simulated purity

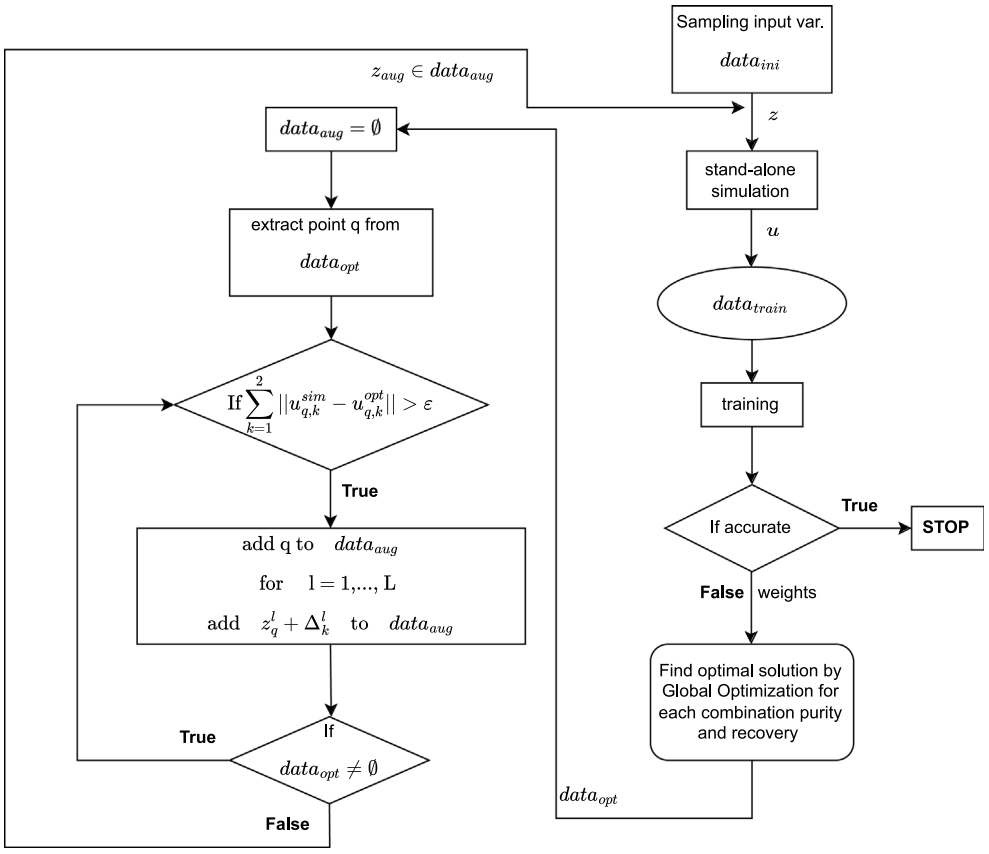
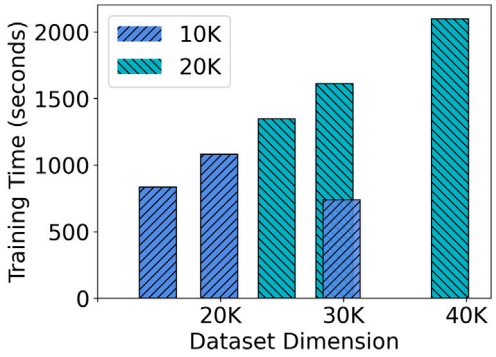


Fig. 11. Schematic view of the ADORO algorithm. New data are added perturbing stationary points that have a prediction error greater than a given threshold ε.

overall dataset size at the end of ADORO

	$data_{ini}$ size	
added points	10K	20K
+5K	14911	24576
+10K	19893	29275
+20K	29853	38673

(a) For each $data_{ini}$, we report how many points are retained in the dataset by adding 5K, 10K, and 20K and filtering with the SVM for numerical instabilities.



(b) Training times in seconds. The datasets are ordered by the size of the dataset.

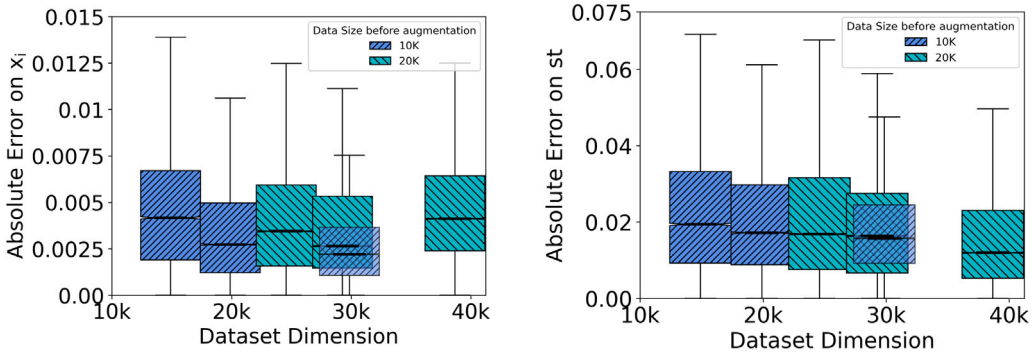
Fig. 12. ADORO size and training time of the different data-sets depending on the different $data_{ini}$.

(see Fig. 15) and the design (Fig. 16). The design and the cost allow us to evaluate the solution for the industrial application, the simulated purity translates into the practical feasibility of the solutions. Furthermore, we consider the overall computational effort (optimization and model fitting) to evaluate the efficiency of the approach (Section 4.2).

For enhancing readability, the design cost are normalized by the cost obtained with the 343k model for 90% purity and 99% recovery case-study (544 euros/ton). We include as a reference the pure sampling datasets (10k, 20k, and 343k).

4.1. Design cost and purity

Looking at the objective function allows us to evaluate also the cost of underestimating the purity, which could lead to unnecessarily complicated configurations or too expensive designs, which is what happens for the solution found by the model built by using AAD. In Fig. 15, we have selected the best solution in terms of the trade-off between feasibility and size of the dataset both for ADORO and for AAD. Histograms represent the cost of the design (normalized with respect to the cost of the reference solution found by the 343k dataset for the case with 90% purity). The simulated purity is reported as a dot and a line connects the purity achieved by a single approach for different dataset sizes. As an example in Fig. 15(c) ADORO achieves a purity of 99.1% with the dataset 10 + 10k and a purity of 99% with the 10 + 20k (second and third columns) and the two values are connected by the dotted orange line. The horizontal line represents the target purity.



(a) Box plot x_i variable

(b) Box plot *st* variable

Fig. 13. Box plots of the absolute errors on the two output variables with $BS = 256$ and $NN = 64$ on the ADORO datasets The order in which we report the datasets is: 10 + 5k, 10 + 10k, 20 + 5k, 20 + 10k, 10 + 20k, 20 + 20k.

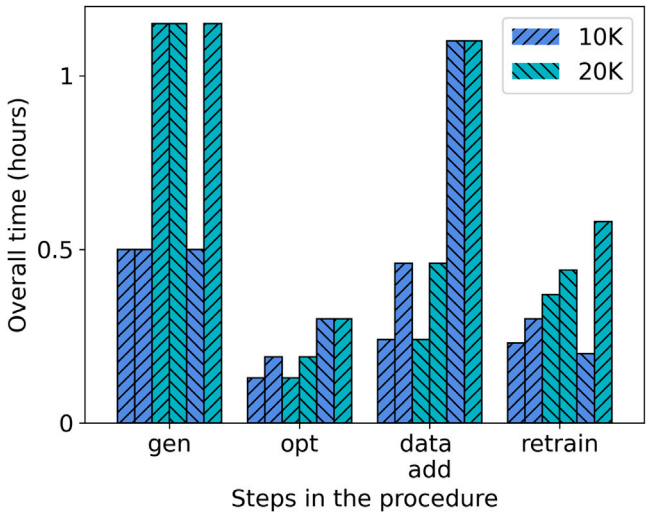


Fig. 14. ADORO. Computational times for the different phases of the procedure. The "gen" refers to the $data_{ini}$ (baseline).

It turns out that ADORO produces the best results since the cost is similar (and sometimes even better) to the one found with the 343k dataset. Indeed, the histograms of ADORO are most of the time lower than the others. AAD pays the price of underestimating the purity, obtaining much higher objective function values (and hence higher histograms).

Moreover, the solution produced by ADORO always satisfy the purity requirements without overestimating significantly.

In Fig. 16 (and Table 8), we report the design corresponding to the best solution found for the case-study with purity 99% and recovery 95%. We consider only the designs obtained with the 343k dataset and ADORO 10k + 20k, since they have the best objective function values. The process designs obtained with AAD are much more complex, using partial and self-recycling streams, which strongly decrease the robustness and operability of the process. The designs obtained with ADORO and the 343k dataset have the same global structure with the same number of stages and connections. Still, the operating conditions, the membrane's surfaces, and the down pressures are different. In this case and more generally for binary systems targeting these levels of purity and recovery, two (or three) stages processes are found similar to the ones described in the scientific literature, and compliant to industrial practices (air separation, biogas). It is worth noticing, that the cost of the solution obtained using ADORO (10k + 20k) is smaller than the one obtained with the pure sampling strategy (see Fig. 15(c)). We can explain this result considering that ADORO refines the prediction model only around the interesting points (local optima). For this reason, it needs fewer points than a pure sampling strategy in order to obtain a model of equal or better quality in the area of interest.

Table 8
Parameter of the designs obtained for case study 99% purity and 95% recovery. See Fig. 16 for the full design.

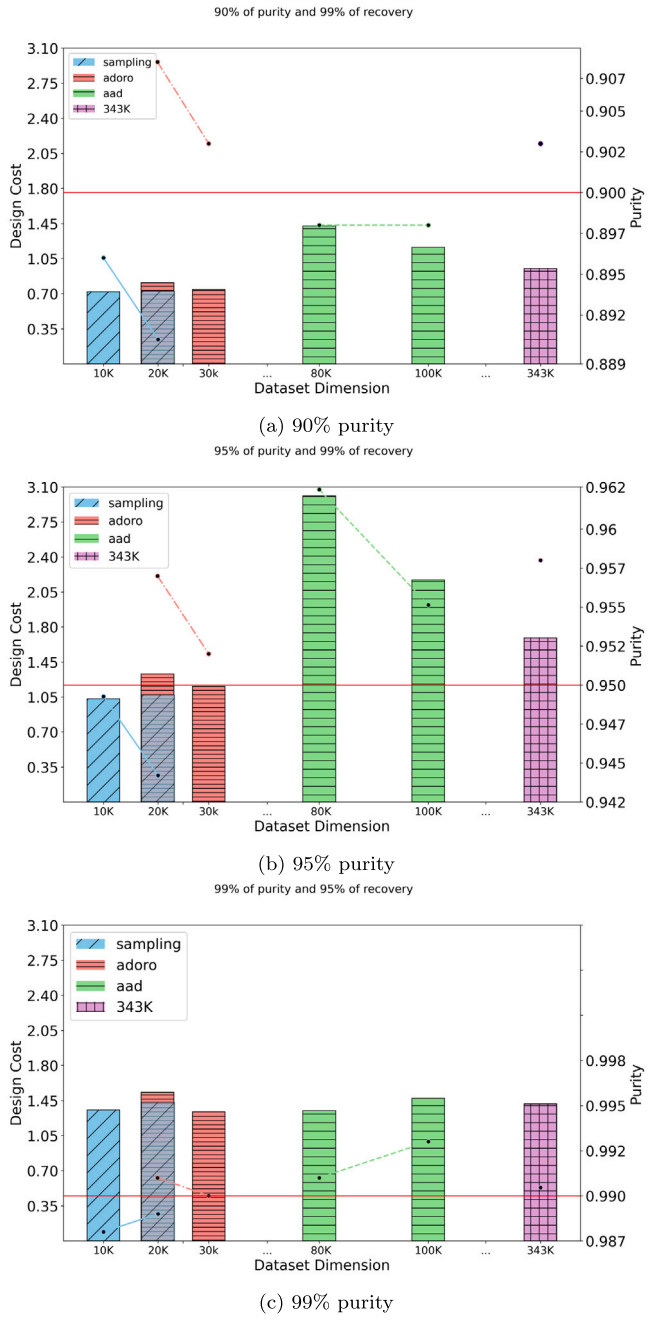
	343k		ADORO 10k+20k	
	I stage	II stage	I stage	II stage
Area	4082.95	445.35	4681.06	472.87
P^{up}	50		50	
P^{down}	3.3	1	1	4.47
θ	0.685	0.338	0.309	0.704
H_2 ret	0.989	0.549	0.990	0.457
H_2 perm	0.387	0.070	0.409	0.071

4.2. Computational effort

In Table 9, we summarize the time needed for building and using a machine learning model instead of the original model based on the discretization of differential equations. Performing 1000 iterations of Multi-start with the original model can take up to 10 h when the purity required is high. Furthermore, the optimization suffers from high numerical instability leading to many infeasible local searches. For all ANN models, the optimization time is significantly reduced: two up to 3 orders of magnitude. When we use an ML model built with ADORO, also the time needed to build the model is competitive. Indeed, if we use the ADORO model to study different combinations of purity and recovery the overall time of building the model and the optimization process is comparable to or even smaller than the one necessary for the model based on the discretization of the differential equation. Furthermore, the numerical issues are highly reduced, allowing the optimization strategy to better explore the search space.

5. Conclusion

It is already well-known that data-driven, machine learning models are vulnerable to inaccuracies outside the domain of their training data (Schweidtmann et al., 2021). It is also well-known that adversarial data augmentation is a useful tool for increasing the trustworthiness of machine learning models. This paper explores the possibility of augmenting the training data with points selected by multistart optimization procedure. This data augmentation strategy implicitly uses the way optimization pushes decision variables to extremes to generate points where the neural network approximation and MEMSIC



(a) 90% purity

(b) 95% purity

(c) 99% purity

Fig. 15. We compare the best results obtained by the optimization with the two augmentation strategies (AAD 10k and 50k, and ADORO with 10 + 10k and 10 + 20k datasets) and the pure sampling (10k, 20k and 343k). Histograms represent the cost of the design. The simulated purity is reported as a dot and a line connects the purity achieved by a single approach for different dataset sizes. The horizontal line represents the target purity.

simulator disagree. The idea is complementary to batch Bayesian optimization (González et al., 2016; Folch et al., 2022; Paulson and Lu, 2022) in the sense that the multistart strategy allows us to develop adversarial examples around several data points. The idea is also complementary to multi-fidelity Bayesian optimization (Song et al., 2019; Folch et al., 2022) in the sense that the new data augmentation strategy used on one purity level can then be used as a base model for another purity level, i.e. the data generated by the adversarial strategy can be reused in similar optimization strategies. One interesting item of note here is that focusing only on the *global minimum point* over the entire decision-making problem is therefore not enough to us: the new data augmentation strategy works best when we can use all the different stationary points. This can be achieved for instance by using multistart optimization.

Table 9
Comparison between the computational time needed by the overall procedure with and without an ML model. We consider ML models built with datasets of different sizes, including both the time for building the model and the (negligible) time for performing 1000 multistart iterations. Then we add as a baseline the time needed to perform 1000 multistart iterations when modeling the membrane behavior by the discretization of the differential equation.

Model	# samples	Building model time	Optimization time
Discretization-based	–	–	1–10 h
Pure sampling	~150k/300k	>50 h	minutes
AAD	~100k	15 h	minutes
ADORO	~50k	7 h	minutes

CRediT authorship contribution statement

Bernardetta Addis: Wrote (and revised) the methods and computational results sections, Formulated the idea of using machine learning in the optimization model and supervised the development of the ADORO data-augmentation strategy. Supervised all the computational experiments and the analysis of the results. **Christophe Castel:** Proposed the case study and provided the simulation tools and the physical knowledge of the problem, Analysis of the results, Provided feedback on the industrial point of view, and revised the paper. **Amalia Macali:** Wrote the first draft of the text, Formulated the idea of using machine learning in the optimization model and the novel ADORO data-augmentation strategy, Produced all the codes, performed the computational tests, and participated in the analysis of the results. **Ruth Misener:** Wrote the introduction, Literature review, Supervised the development of the ADORO data-augmentation strategy. **Veronica Piccialli:** Wrote (and revised) the methods and computational results sections, Formulated the idea of using machine learning in the optimization model and supervised the development of the ADORO data-augmentation strategy. Supervised all the computational experiments and the analysis of the results.

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.

Data availability

The authors do not have permission to share data.

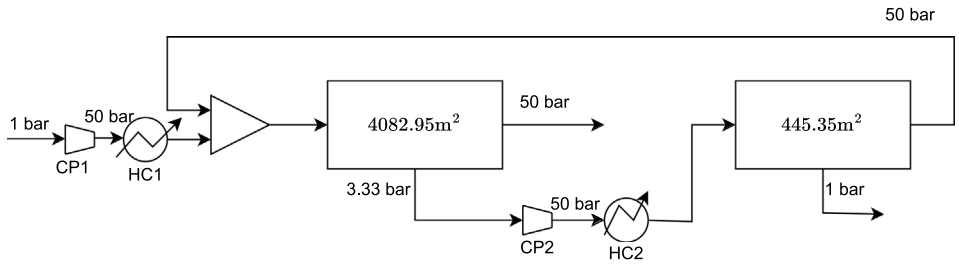
Acknowledgments

Amalia Macali gratefully acknowledges the Dreams funding granted by the University of Lorraine, France.

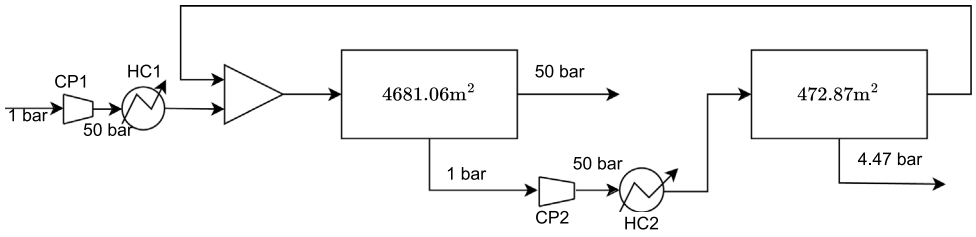
Ruth Misener gratefully acknowledges an Engineering & Physical Sciences Research Council Fellowship, UK (grant no. EP/P016871/1) and a BASF/Royal Academy of Engineering Research Chair in Data-Driven Optimization.

Veronica Piccialli gratefully acknowledges the SEED-PNR H2OptLearn funding granted by Sapienza University of Rome, Italy.

Part of the experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see https://www.grid5000.fr).



(a) Pure sampling with the 343K dataset.



(b) ADORO with the 10K+20K dataset.

Fig. 16. Design for case study 99% purity and 95% recovery. Input feed: 1000 mol/s, 40% of CO_2 and 60% of H_2, $P^{down} \in [1, 30]$ bar (no vacuum pump), $P^{up} \in [50, 100]$ bar uniform (i.e. it is the same for all stages).

Appendix A. Mathematical model and simulation of a single membrane behavior

A chain of differential equations describes the physical behavior of a membrane module. The equation links each component's input flow to that component's permeated flow. The membrane's permeability with respect to each component, the membrane's area, and the applied pressures influence the output composition and flow rate. The material balance for component j over the differential area dA is described by the following differential equation:

$$-d(x_j F) = d(y_j G) = dA\ \pi_j \left(P^{up} x_j - P^{down} y_j\right) \quad \forall j \in C \tag{A.1}$$

where:

1. C is the set of gas components, for our example $C = \{H_2, CO_2\}$
2. F and G are the flow rates on the feed (high-pressure) stream and on the permeated (low-pressure) stream respectively;
3. P^{up} and P^{down}, are the pressures on the feed side and the permeate side, respectively;
4. π_j is the permeability of component j
5. x_j and y_j, are the mole fractions of component j on the feed side and the permeate side, respectively.

A.0.1. Standardized form

Eq. (B.19) can be reduced to a standardized dimensionless form, that allows solving the problem for the largest span of input values. The standardized equation can be obtained by dividing equation (A.1) by the feed flow F, the maximal permeance π_j and pressure up P^{up}. New process variables are then introduced:

- the stage-cut θ: the ratio between the permeated flow and the feed flow

$$\theta = \frac{G}{F} \tag{A.2}$$

- the pressure ratio ϕ

$$\phi = \frac{P^{down}}{P^{up}} \tag{A.3}$$

- *standardized* area st_j

$$st = A \frac{P^{up} \pi_{ref}}{F} \tag{A.4}$$

st is often reported in the literature as a dimensionless area. Its formulation comes from the normalization of the regular equations (B.19). This variable is defined relatively to a reference component ref through its permeability π_{ref}.

Finally, we obtain:

$$d(y_j \theta) = dst_j \left(x_j - \phi y_j\right) \quad \forall j \in C \tag{A.5}$$

A.1. Single membrane's simulators

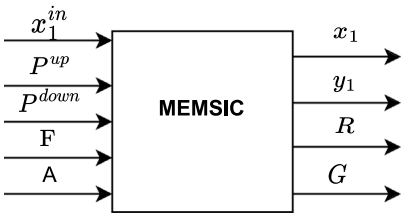
A single module behavior can be efficiently simulated by means of different simulation tools based on Eq. (A.1) or (A.5). We use two simulation tools (Bounaceur et al., 2017):

MEMSIC: a CAPE-OPEN module to simulate the behavior of a gas permeation membrane (Bounaceur et al., 2017). In conjunction with process simulation software (such as Aspen, Hysys, Pro/II, PROSIM), it allows to simulate multi-stage separation. Each simulation requires the use of a graphical interface, where the input parameters are inserted by hand, see Fig. A.17(a).

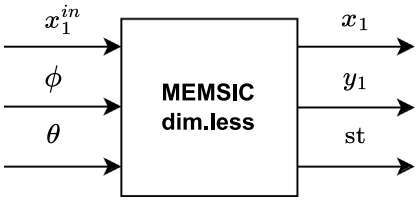
A stand-alone FORTRAN code implementing the core of MEMSIC, that allows to simulate a single membrane's behavior by using the normalized equation (A.5), see Fig. A.17(b). This code has a textual interface.

We used each tool for a different task. In order to generate our training set, we use the FORTRAN code (allowing batch calculations). To test the validity of the approach, we performed tests also on multi-stage systems, thus, we used the CAPE-OPEN MEMSIC module.

In Fig. A.17 and Table A.10, we describe the inputs and outputs of the two simulation tools. For sake of clarity, we consider the case of a gas with two components, and without loss of generality we denote with "1" the most permeable gas. In our example, the gas is composed of H_2 and CO_2, and we consider a CO_2 selective membrane, thus the most permeable gas is the CO_2 which is denoted by index 1. Since we



(a) The MEMSIC simulator has in input the input concentration of gas 1 x_1^{in}, the pressures on the feed side P^{up} and on the permeate side P^{down}, the feed F, the membrane area A. In output it produces the retentated and permeated flows R and G, the gas concentration x_1 of gas 1 in the retentated and the concentration y_1 of gas 1 in the permeated.



(b) The stand-alone Fortran simulator takes in input the pressure ratio $\phi = P^{down}/P^{up}$, the input concentration of gas 1 x_1^{in} and the stage-cut $\theta = G/F$, producing in output the concentration x_1 of gas 1 in the retentated, the concentration y_1 of gas 1 in the permeated and the standardized area $st = A\frac{P^{up}P_1}{F}$.

Fig. A.17. Comparison between the MEMSIC module and stand-alone Fortran simulator.

are in the binary case, we can derive the concentration of the H_2 by exploiting that they have to sum up to 1. Similarly, we denote by st the area standardized relative to the CO_2 permeance, i.e. $st = A\frac{P^{up}\pi_{CO_2}}{F}$ (see Eq. (A.4))

We stress that we consider the membrane permeability as an input parameter, and we build the machine learning model representing a specific membrane with a given permeability for each gas component.

The Fortran simulator solves the Eqs. (A.6)–(A.8) where $\alpha_i = \frac{\pi_i}{\pi_1}$ is the ideal selectivity of component i versus component 1 of the membrane.

$$\frac{\mathrm{d}st}{\mathrm{d}\theta} = \left(\sum_{i=1}^{2}\left(\frac{x_i - \phi y_i}{\alpha_i}\right)\right)^{-1} \tag{A.6}$$

$$\frac{\mathrm{d}x_1}{\mathrm{d}\theta} = \frac{1}{1-\theta}\left(x_1\sum_{i=1}^{2}\left(\frac{x_i - \phi y_i}{\alpha_i}\right) - (x_1 - \phi y_1)\right)\left(\sum_{i=1}^{2}\left(\frac{x_i - \phi y_i}{\alpha_i}\right)\right)^{-1} \tag{A.7}$$

$$y_1 = (x_1 - \phi y_1)\left(\sum_{i=1}^{2}\left(\frac{x_i - \phi y_i}{\alpha_i}\right)\right)^{-1} \tag{A.8}$$

We can observe that Eq. (A.8) is an algebraic expression and not a differential equation. Indeed, y_1 can be derived from the flow balance constraint of the single component:

$$x_1^{in}F = x_1(F - G) + y_1 G \tag{A.9}$$

where $F-G$ corresponds to the retentate flow (due to flow conservation constraints). Therefore, y_1 is calculated as follows:

$$y_1 = \frac{x_1^{in}}{\theta} - x_1\frac{(1-\theta)}{\theta} \tag{A.10}$$

A.2. Relationship between the simulator and the ML model

In Table A.10, we distinguish input and output variables for both the simulation tools and the machine learning model. The machine learning model represents the Fortran tool, so it has the same input and output. The only difference lies in how the variable y_1 is treated.

Having the possibility to compute y_1 by Eq. (A.8), allows us to choose the number of outputs of the ANN, which can be either 2 (x_1, and st) or 3 (x_1, st, and y_1). Indeed, we did several tests trying to train our ANN with 2 or 3 outputs. The difference in the prediction error turned out to be not too significant. So we preferred to have only 2 outputs. Furthermore, we chose to have the retentate as the output because in this case-study it represents the purity of H_2 that we are trying to achieve as accurately as possible. So we define x_1 as the concentration of gas 1 in the retentated considering the single membrane, and we define the purity as the percentage of product that goes out of the system, that is, in our case-study, the percentage of H_2 in the retentate coming out of the system.

Table A.10
Input(IN) and output(OUT) for the two simulation tools and the Neural Network Model. M = MEMSIC, F = Fortran stand-alone, ANN = Neural Network Surrogate.

Name	Description	M	F	ANN
π_j	Membrane gas permeability for each comp. $j = 1, 2$	IN	IN	FIX
x_1^{in}	Input concentration of gas 1	IN	IN	IN
P^{up}	The pressures on the feed side	IN	–	–
P^{down}	The pressure on the permeate side	IN	–	–
ϕ	Pressure ratio P^{down}/P^{up}	–	IN	IN
F	Flow in input to the membrane (feed)	IN	–	–
R	Retentated flow	OUT	–	–
G	Permeated flow	OUT	–	–
θ	Stage-cut Ratio between G and F	–	IN	IN
A	Membrane area	IN	–	–
st	Standardized area $st = A\frac{P^{up}\pi_1}{F}$	–	OUT	OUT
x_1	Concentration of gas 1 in the retentated	OUT	OUT	OUT
y_1	Concentration of gas 1 in the permeated	OUT	OUT	–

It is worth to notice that, even if the normalized equations are sufficient to determine the retentated and permeated compositions, the optimal design of a system asks for the determination of membrane surfaces and applied pressures. Indeed, the overall cost of a system depends on flows, applied pressures (for compressors cost estimation) and membrane surfaces (for the membrane modules cost estimation). These parameters can be derived using the following equations:

$$G = \theta F \tag{A.11}$$

$$A = st\frac{F}{P^{up}\cdot\pi_1} \tag{A.12}$$

Appendix B. Optimization model

The mathematical optimization model is based on some quite common assumptions used in the literature (the interested reader can refer to Ramírez-Santos et al. (2018) for details). We consider a generic number of stages, i.e. number of membranes in the overall system, and we represent them with the set S.

The general model considers an arbitrary number of gas components. We restrict to the case of two gases to simplify the notation. We observe that gas composition are represented by molar fractions, thus they must sum up to 1. Without loss of generality, we can consider only the concentration of one gas (namely gas 1) and calculate the concentration of the other by complementing it.

We assume that all the membranes have the same pressure on the feed (and retentate) side (uniform P_{up}), and we assume that the pressure

Table B.11
Optimization model variables (bi-gas case).

System variables	
F	Feed flowrate
G	Permeate flowrate
R	Retentate flowrate
y_1	Fraction of component 1 in the permeate
x_1	Fraction of component 1 in the retentate
Single stage variables $\forall s \in S$	
A_s	Membrane's area
P^{up}	Up stream pressure
P_s^{down}	Down stream pressure
f_s	Overall feed flowrate
f_s^{split}	Fresh feed flow rate entering the membrane
g_s	Total permeate flowrate
r_s	Total retentate flowrate
$x_{s,1}^{in}$	Fraction of component 1 in the feed
$y_{s,1}$	Fraction of component 1 of the permeate
$x_{s,1}$	Fraction of component 1 of the retentate
Inter-stages connection variables $\forall s \in S$	
r_s^{out}	Retentate flowrate going out the system
g_s^{out}	Permeate flowrate going out the system
g_{s,s_1}^{split}	Permeate flowrate entering into membrane s_1
r_{s,s_1}^{split}	Retentate flowrate entering into membrane s_1

on the permeate side (P_{down}) is greater or equal than 1 and lower than the pressure on the feed/retentate ($P_{down} \leq P_{up}$). Therefore, whenever a permeated flow goes in input to a stage with higher pressure P_{up}, the flow needs to go through a compressor.

The optimization variables are described one by one in Table B.11. The overall system has one given input, the feed, and two outputs: the retentate and the permeate. Similarly, the single stage constituted by a membrane $s \in S$ has one input and two output flows, retentate and permeate.

For each membrane both retentate and permeate can be split to be distributed as an input to other membrane, or to the membrane itself (self-loop) and/or sent out of the system. To represent this possibility, additional variables are needed:

- split: it represents the flowrate that goes from one stage to another, e.g. $g_{s,s1}^{split}$ represents the quantity of permeated flow of membrane s that is redirected to membrane $s1$.
- out: it expresses the flowrate that goes from one stage to the system output, e.g. g_s^{out} expresses the quantity of permeated flow of membrane s that is directed to the system output.

The model constraints can be divided in different families: flow conservation constraints that must be valid at system level and for each stage, on both the flows and the composition; connecting constraints allowing to connect (coherently) flows between different membranes; coherence in the total composition of each flow; and finally the constraints describing the separation by means of the machine learning (see Eq. (B.19)). We introduce them one by one.

Flow conservation constraints for the overall system and for each membrane s (both for total and component 1):

$$F = R + G \tag{B.1}$$

$$Fx_1^{in} = Rx_1 + Gy_1 \tag{B.2}$$

$$f_s = r_s + g_s \qquad \forall s \in S \tag{B.3}$$

$$f_s x_{s,1}^{in} = r_s x_{s,1} + g_s y_{s,1} \qquad \forall s \in S \tag{B.4}$$

Flow conservation constraints related to connections inter-stages (split coherence) and system to single stage:

$$F = \sum_{s \in S} f_s^{split} \tag{B.5}$$

$$R = \sum_{s \in S} r_s^{out} \tag{B.6}$$

$$G = \sum_{s \in S} g_s^{out} \tag{B.7}$$

$$r_s = \sum_{s_1 \in S} r_{s,s1}^{split} + r_s^{out} \qquad \forall s \in S \tag{B.8}$$

$$g_s = \sum_{s_1 \in S} f_{s,s1}^{split,perm} + g_s^{out} \qquad \forall s \in S \tag{B.9}$$

$$f_s = \sum_{s1 \in S} r_{s1,s}^{split} + g_{s1,s}^{split} + f_s^{split} \qquad \forall s \in S \tag{B.10}$$

$$f_s x_s^{in} = \sum_{s1 \in S} (r_{s1,s}^{split} z_{s1,1} + g_{s1,s}^{split} y_{s1,1} + f_s^{split} x_1^{in}) \qquad \forall s \in S \tag{B.11}$$

The membrane behavior is described by the machine learning regression model:

$$x_{s,1} = \sum_{k \in N} w_{k,x_1}^2 \cdot \sigma(\lambda_{k,s}) + b_{x_1}^2 \qquad \forall s \in S \tag{B.12}$$

$$st_s = \sum_{k \in N} w_{k,st}^2 \cdot \sigma(\lambda_{k,s}) + b_{st}^2 \qquad \forall s \in S \tag{B.13}$$

$$\lambda_{k,s} = w_{k,\phi}^1 \cdot \phi_s + w_{k,x_1^{in}}^1 \cdot x_{s,1}^{in} + w_{k,\theta}^1 \cdot \theta_s + b_k^1 \qquad \forall k \in N, s \in S \tag{B.14}$$

where N is the set of neurons in the hidden layer and

$$\sigma(t) = \frac{1}{1 + \exp(-t)}$$

Finally, the following equations connect the stage variables with the ML regression model variables:

$$G_s = \theta_s F_s \tag{B.15}$$

$$A_s = st_s \frac{F_s}{P^{up} \cdot \pi_1} \tag{B.16}$$

$$\phi_s = \frac{P_s^{down}}{P^{up}} \tag{B.17}$$

and the SVM-derived separation constraints:

$$\alpha_\phi \cdot \phi_s + \alpha_{x_1^{in}} \cdot x_{s,1}^{in} + \alpha_{k,\theta} \cdot \theta_s \leq \beta \qquad s \in S \tag{B.18}$$

where α and β are the coefficients of the separating hyperplane.

The model is completed by performance constraints on purity (percentage presence of some components in the product) and recovery (quantity of final product with respect to its availability on the input feed). For example a 0.99 of purity and 0.95 of recovery results in the following constraints:

$$(1 - x_1) \geq 0.99$$

$$R(1 - x_1) \geq 0.95F(1 - x_1^{in})$$

All variables are bounded to be in a box where the lower bound is nonnegative, and the upper bound can be derived by the physical meaning of the variable. Finally, a maximum recycling ratio is imposed to the splits of any stage towards itself (self-loops) to avoid solutions that would not lead to physically stable configurations (full self-loops).

The main difference with respect to the previous optimization approaches is in the modeling of the membrane's behavior that for us is represented by constraints (B.12)–(A.3). In the previous model, the membrane was represented by the discretization of Eqs. (A.1):

$$g_i y_{1,i} = \delta_A \pi_1 (P^{up} x_{1,i} - P^{down} y_{1,i}) \qquad i \in 1, \ldots, n \tag{B.19}$$

where δ_A is the discretized area, i.e. the area f each discretized slice of the membrane, and n is the number of slices used in the discretization. The finer the discretization, the higher the number of equations necessary to describe the membrane's behavior.

B.1. Objective function

The objective function represents the total annual separation costs, considering both capital and operational expenditure. CAPital EXpenditure (CAPEX) includes membrane area and frame, compressors, and

Table B.12
Cost model.

Capital expenditures	
$TFI = 1.344 \cdot I_{tot}$	Total facility investment
Operational expenditures	
$CMC + LTI = 0.2 \cdot TFI$	Contract and material maintenance + Local taxes and insurance
$DL + LOC = 23.65 \cdot t_{op}$	Direct labor + Labor overhead cost
$EC = t_{op} \cdot W_{tot} \cdot K_{el}$	Energy Cost
$MRC = \sum_{s \in S} A_{m_s} \cdot \nu \cdot K_{mr}$	Membrane replacement cost
$OPEX = CMC + LTI + DL + LOC + EC + MRC$	Total operational expenditures
$STC = stc_{coef} \cdot OPEX$	Start-up cost
$CAPEX = TFI + STC$	Total capital cost
Annual and specific separation costs	
$APL = 3600 \cdot 0.0224 \cdot t_{op} \cdot F^{Loss} \cdot \frac{X^{Loss}}{X^{Prod}}$	Annual product losses
$TAC = CAPEX \cdot \frac{i \cdot (1+i)^{z-1}}{(1+i)^z - 1} + OPEX + APL$	Total annual costs
Objective function	
$SC = TAC/(F^{Prod} \cdot 3600 \cdot 0.0224 \cdot t_{op})$	Specific product separation cost

Table B.13
Cost equations used to determine product gas separation cost.

Equipment costs	
Each term is intended for a single piece of equipment (stage, compressor)	
$I_m^s = K_m \cdot A_m^s$	Membrane cost
$I_{mf}^s = (A_m^s/2000)^{0.7} \cdot K_{mf} \cdot (p^{up}/55)^{0.875}$	Membrane frame cost
$I_{cp} = C_c \cdot (W_{cp}/74.6)^{0.77} \cdot (MPF_c + MF_c - 1) \cdot UF_{1968} \cdot K_{er}$	Compressor cost

vacuum pumps. The OPerational EXpenditure (OPEX) include electricity related to compression and vacuum equipment, membrane replacement, operation, and maintenance cost. It is worth noticing, that the mass flows traversing the devices (compressors, membranes) influence both the CAPEX and OPEX of such devices.

The objective function is mainly based on NETL guidelines (Zhai and Rubin, 2013) and adaptations of previous works (Macali, 2023). Table B.12 reports the expression of all the elements necessary to calculate the specific product separation cost, that is the overall annual cost normalized by the product's flow F^{Prod} and the operational time t_{op}.

The two terms I_{tot} and W_{tot} represent the equipment costs (membrane and compressors) and the total required energy. They contribute both to CAPEX and OPEX calculations.

The total equipment cost I_{tot} is the sum of the cost of the single equipment devices.

$$I_{tot} = \sum_{cp} I_{cp} + \sum_{s} (I_m^s + I_{mf}^s) \tag{B.20}$$

where the equipment's costs I_{cp}, I_m^s and I_{mf}^s are defined in Table B.13.

W_{tot} is expressed as following:

$$W_{tot} = \frac{\sum_{cp} W_{cp}}{\Phi}, \tag{B.21}$$

with:

$$W_{cp} = \frac{F_{in} \times 10^{-3}}{\eta_c} \cdot \frac{\gamma \cdot R \cdot T}{\gamma - 1} \cdot [(\frac{P_{out}}{P_{in}})^{\frac{(\gamma-1)}{\gamma}} - 1] \tag{B.22}$$

where the necessary parameters and variables are reported in Table B.14 and Table B.16.

Table B.15 reports the coefficients and parameters necessary to calculate the specific product separation cost.

Table B.14
Variables and parameters necessary to calculate the device (compressor) energy consumption.

Compressor's energy	
F_{in}	Flow entering the device
P_{in}	Pressure of the flow entering the device
P_{out}	Pressure of the flow entering the device
F_{loss}	Mass flow of lost product
X_{loss}	Percentage of lost product
X_{prod}	Percentage of product of interest

Table B.15
Cost parameters used in Table B.12.

Capital cost parameters		
C_c	$1 \times 23\,000$	USD_{1968}
K_m(polymer)	50	EUR/m^2
K_{mf}	2.86×10^5	EUR
K_{er}	0.9	EUR/USD
MPF_c	2.9	–
MF_c	5.11	–
UF_{1968}	4.99	–
Operational and annual cost parameters		
ν	0.25	–
K_{mr}(polymer)	25	EUR/m^2
t_{op}	8322	h/year
K_{el}	0.08	EUR/kWh
K_{gp}	0.8	EUR/N m^3
i	0.08	–
z	15	years
η_c	0.85	–
Φ	0.95	–
γ	1.36	–
R	8.314	J/(K mol)
T	308.15	K

Table B.16
Parameters and variables' bounds for the optimization.

Name	Description	Value/bounds	Units
π_1	Membrane gas permeability for CO_2	1000	GPU
π_2	Membrane gas permeability for H_2	85	GPU
F	(system) Feed	1000	mol/s
x_1^{in}	Input concentration of gas CO_2	0.4	–
P^{up}	The pressure on the feed side	[50, 100]	bar
P^{down}	The pressure on the permeate side	[1, 30]	bar
A	Single stage (membrane) area	[1, 10 000]	m^2

References

Alam, G., Ihsanullah, I., Naushad, M., Sillanpää, M., 2022. Applications of artificial intelligence in water treatment for optimization and automation of adsorption processes: Recent advances and prospects. Chem. Eng. J. 427, 130011.

Antoniou, A., Storkey, A., Edwards, H., 2017. Data augmentation generative adversarial networks. arXiv preprint arXiv:1711.04340.

Arias, A.M., Mussati, M.C., Mores, P.L., Scenna, N.J., Caballero, J.A., Mussati, S.F., 2016. Optimization of multi-stage membrane systems for CO2 capture from flue gas. Int. J. Greenh. Gas Control 53, 371–390.

Asghari, M., Dashti, A., Rezakazemi, M., Jokar, E., Halakoei, H., 2020. Application of neural networks in membrane separation. Rev. Chem. Eng. 36 (2), 265–310.

Baker, R.W., 2012. Membrane Technology and Applications. John Wiley & Sons.

Bhosekar, A., Ierapetritou, M., 2018. Advances in surrogate based modeling, feasibility analysis, and optimization: A review. Comput. Chem. Eng. 108, 250–267.

Bounaceur, R., Berger, E., Pfister, M., Ramirez Santos, A.A., Favre, E., 2017. Rigorous variable permeability modelling and process simulation for the design of polymeric membrane gas separation units: MEMSIC simulation tool. J. Membr. Sci. 523, 77–91.

Brendel, W., Rauber, J., Bethge, M., 2017. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. arXiv preprint arXiv:1712.04248.

Byrd, R.H., Nocedal, J., Waltz, R.A., 2006. Knitro: An integrated package for nonlinear optimization. In: Di Pillo, G., Roma, M. (Eds.), Large-Scale Nonlinear Optimization. Springer US, Boston, MA, pp. 35–59.

Cozad, A., Sahinidis, N.V., Miller, D.C., 2014. Learning surrogate models for simulation-based optimization. AIChE J..

Devabhaktuni, V.K., Zhang, Q.-J., 2000. Neural network training-driven adaptive sampling algorithm for microwave modeling. In: 2000 30th European Microwave Conference. IEEE, pp. 1–4.

Di Pretoro, A., Bruns, B., Negny, S., Grünewald, M., Riese, J., 2022. Demand response scheduling using derivative-based dynamic surrogate models. Comput. Chem. Eng. 160, 107711.

Dornier, M., Decloux, M., Trystram, G., Lebert, A., 1995a. Dynamic modeling of crossflow microfiltration using neural networks. J. Membr. Sci. 98 (3), 263–273.

Dornier, M., Decloux, M., Trystram, G., Lebert, A., 1995b. Interest of neural networks for the optimization of the crossflow filtration process. LWT - Food Sci. Technol. 28 (3), 300–309.

Eason, J., Cremaschi, S., 2014. Adaptive sequential sampling for surrogate model generation with artificial neural networks. Comput. Chem. Eng. 68, 220–232.

Fahmi, I., Cremaschi, S., 2012. Process synthesis of biodiesel production plant using artificial neural networks as the surrogate models. Comput. Chem. Eng. 46, 105–123.

Folch, J.P., Lee, R.M., Shafei, B., Walz, D., Tsay, C., van der Wilk, M., Misener, R., 2022. Combining multi-fidelity modelling and asynchronous batch Bayesian optimization. arXiv preprint arXiv:2211.06149.

Gabrielli, P., Gazzani, M., Mazzotti, M., 2017. On the optimal design of membrane-based gas separation processes. J. Membr. Sci. 526, 118–130.

Gao, X., Deng, F., Yue, X., 2020. Data augmentation in fault diagnosis based on the wasserstein generative adversarial network with gradient penalty. Neurocomputing 396, 487–494.

Gao, X., Tan, Y.-a., Jiang, H., Zhang, Q., Kuang, X., 2019. Boosting targeted black-box attacks via ensemble substitute training and linear augmentation. Appl. Sci. 9 (11).

Garud, S.S., Karimi, I., Kraft, M., 2017. Smart sampling algorithm for surrogate model development. Comput. Chem. Eng. 96, 103–114.

González, J., Dai, Z., Hennig, P., Lawrence, N., 2016. Batch Bayesian optimization via local penalization. In: Artificial Intelligence and Statistics. PMLR, pp. 648–657.

Goodfellow, I.J., Shlens, J., Szegedy, C., 2014. Explaining and harnessing adversarial examples. arXiv preprint arXiv:1412.6572.

Granacher, J., Kantor, I.D., Maréchal, F., 2021. Increasing superstructure optimization capacity through self-learning surrogate models. Front. Chem. Eng. 3.

Hamachi, M., Cabassud, M., Davin, A., Mietton Peuchot, M., 1999. Dynamic modelling of crossflow microfiltration of bentonite suspension using recurrent neural networks. Chem. Eng. Process.: Process Intensif. 38 (3), 203–210.

Henao, C.A., Maravelias, C.T., 2011. Surrogate-based superstructure optimization framework. AIChE J. 57, 1216–1366.

Himmelblau, D.M., 2000. Applications of artificial neural networks in chemical engineering. Korean J. Chem. Eng. 17, 373–392.

Hu, J., Kim, C., Halasz, P., Kim, J.F., Kim, J., Szekely, G., 2021. Artificial intelligence for performance prediction of organic solvent nanofiltration membranes. J. Membr. Sci. 619, 118513.

Ibrahim, D., Jobson, M., Li, J., Guillén-Gosálbez, G., 2018. Optimization-based design of crude oil distillation units using surrogate column models and a support vector machine. Chem. Eng. Res. Des. 134, 212–225.

Kajero, O.T., Chen, T., Yao, Y., Chuang, Y.-C., Wong, D.S.H., 2017. Meta-modelling in chemical process system engineering. J. Taiwan Inst. Chem. Eng. 73, 135–145, In Memory of Professor Yung-Cheng Chao.

Khayet, M., Cojocaru, C., 2012. Artificial neural network modeling and optimization of desalination by air gap membrane distillation. Sep. Purif. Technol. 86, 171–182.

Kim, S.H., Boukouvala, F., 2020. Surrogate-based optimization for mixed-integer nonlinear problems. Comput. Chem. Eng. 140, 106847.

Lin, H., He, Z., Sun, Z., Kniep, J., Ng, A., Baker, R.W., Merkel, T.C., 2015. CO2-selective membranes for hydrogen production and CO2 capture – Part II: Techno-economic analysis. J. Membr. Sci. 493, 794–806.

Lin, H., He, Z., Sun, Z., Vu, J., Ng, A., Mohammed, M., Kniep, J., Merkel, T.C., Wu, T., Lambrecht, R.C., 2014a. CO2-selective membranes for hydrogen production and CO2 capture – Part I: Membrane development. J. Membr. Sci. 457, 149–161.

Lin, H., He, Z., Sun, Z., Vu, J., Ng, A., Mohammed, M., Kniep, J., Merkel, T.C., Wu, T., Lambrecht, R.C., 2014b. CO2-selective membranes for hydrogen production and CO2 capture – Part I: Membrane development. J. Membr. Sci. (ISSN: 03767388) 457, 149–161. http://dx.doi.org/10.1016/j.memsci.2014.01.020, URL https://linkinghub.elsevier.com/retrieve/pii/S0376738814000337.

Liu, Q.-F., Kim, S.-H., Lee, S., 2009. Prediction of microfiltration membrane fouling using artificial neural network models. Sep. Purif. Technol. 70 (1), 96–102.

Locatelli, M., Schoen, F., 2013. Global Optimization: Theory, Algorithms, and Applications. Society for Industrial and Applied Mathematics, USA, ISBN: 1611972663.

Macali, A., 2023. Membrane Separation Processes Using Machine Learning Based Mathematical Programming Models, Vol. 3. (Ph.D. thesis). University of Lorraine, 2 rue Léonard de Vinci, 54042 Nancy Cedex.

Magar, R., Wang, Y., Lorsung, C., Liang, C., Ramasubramanian, H., Li, P., Farimani, A.B., 2022. AugLiChem: Data augmentation library of chemical structures for machine learning. Mach. Learn.: Sci. Technol..

McBride1, K., Sundmacher, K., 2018. Overview of surrogate modeling in chemical process engineering. Chem. Ing. Tech..

Mencarelli, L., Chen, Q., Pagot, A., Grossmann, I.E., 2020. A review on superstructure optimization approaches in process system engineering. Comput. Chem. Eng. 136, 106808.

Merkel, T.C., Zhou, M., Baker, R.W., 2012. Carbon dioxide capture with membranes at an IGCC power plant. J. Membr. Sci. (ISSN: 03767388) 389, 441–450. http://dx.doi.org/10.1016/j.memsci.2011.11.012, URL https://linkinghub.elsevier.com/retrieve/pii/S0376738811008283.

Neveux, T., Addis, B., Castel, C., Piccialli, V., Favre, E., 2022. A comparison of process synthesis approaches for multistage separation processes by gas permeation. In: Montastruc, L., Negny, S. (Eds.), 32nd European Symposium on Computer Aided Process Engineering. In: Computer Aided Chemical Engineering, vol. 51, Elsevier, pp. 685–690.

Niu, C., Li, X., Dai, R., Wang, Z., 2022. Artificial intelligence-incorporated membrane fouling prediction for membrane-based processes in the past 20 years: A critical review. Water Res. 216, 118299.

Nuchitprasittichai, A., Cremaschi, S., 2012. An algorithm to determine sample sizes for optimization with artificial neural networks. AIChE J..

Ohs, B., Lohaus, J., Wessling, M., 2016. Optimization of membrane based nitrogen removal from natural gas. J. Membr. Sci. 498, 291–301.

Papernot, N., McDaniel, P., Goodfellow, I., Jha, S., Celik, Z.B., Swami, A., 2017. Practical black-box attacks against machine learning. In: Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security. pp. 506–519.

Paulson, J.A., Lu, C., 2022. COBALT: Constrained Bayesian optimization of computationally expensive grey-box models exploiting derivative information. Comput. Chem. Eng. (ISSN: 0098-1354) 160, 107700. http://dx.doi.org/10.1016/j.compchemeng.2022.107700, URL https://www.sciencedirect.com/science/article/pii/S0098135422000436.

Piccialli, V., Sciandrone, M., 2022. Nonlinear optimization and support vector machines. Ann. Oper. Res. 1–33.

Pinkus, A., 1996. TDI-subspaces ofc(rd) and some density problems from neural networks. J. Approx. Theory 85 (3), 269–287.

Piron, E., Latrille, E., René, F., 1997. Application of artificial neural networks for crossflow microfiltration modelling: "black-box" and semi-physical approaches. Comput. Chem. Eng. 21 (9), 1021–1030.

Qi, R., Henson, M.A., 2000. Membrane system design for multicomponent gas mixtures via mixed-integer nonlinear programming. Comput. Chem. Eng. 24 (12), 2719–2737.

Qin, S.J., Chiang, L.H., 2019. Advances and opportunities in machine learning for process data analytics. Comput. Chem. Eng. 126, 465–473.

Rall, D., Menne, D., Schweidtmann, A.M., Kamp, J., von Kolzenberg, L., Mitsos, A., Wessling, M., 2019. Rational design of ion separation membranes. J. Membr. Sci. 569, 209–219.

Rall, D., Schweidtmann, A.M., Aumeier, B.M., Kamp, J., Karwe, J., Ostendorf, K., Mitsos, A., Wessling, M., 2020a. Simultaneous rational design of ion separation membranes and processes. J. Membr. Sci. 600, 117860.

Rall, D., Schweidtmann, A.M., Kruse, M., Evdochenko, E., Mitsos, A., Wessling, M., 2020b. Multi-scale membrane process optimization with high-fidelity ion transport models through machine learning. J. Membr. Sci. 608, 118208.

Ramírez-Santos, Á., Bozorg, M., Addis, B., Piccialli, V., Castel, C., Favre, E., 2018. Optimization of multistage membrane gas separation processes. Example of application to CO2 capture from blast furnace gas. J. Membr. Sci. 566, 346–366.

Richard Bowen, W., Jones, M.G., Welfoot, J.S., Yousef, H.N., 2000. Predicting salt rejections at nanofiltration membranes using artificial neural networks. Desalination 129 (2), 147–162.

Ritter, J.A., Ebner, A.D., 2007. State-of-the-art adsorption and membrane separation processes for hydrogen production in the chemical and petrochemical industries. Sep. Sci. Technol. 42 (6), 1123–1193.

Satyanarayana, A., Davidson, I., 2005. A dynamic adaptive sampling algorithm (DASA) for real world applications: Finger print recognition and face recognition. Found Intell. Syst. 3488, 631–640,

Schmitt, F., Banu, R., Yeom, I.-T., Do, K.-U., 2018. Development of artificial neural networks to predict membrane fouling in an anoxic-aerobic membrane bioreactor treating domestic wastewater. Biochem. Eng. J. 133, 47–58.

Scholz, M., Alders, M., Lohaus, T., Wessling, M., 2015. Structural optimization of membrane-based biogas upgrading processes. J. Membr. Sci. 474, 1–10.

Schweidtmann, A.M., Esche, E., Fischer, A., Kloft, M., Repke, J.-U., Sager, S., Mitsos, A., 2021. Machine learning in chemical engineering: A perspective. Chem. Ing. Tech. 93 (12), 2029–2039.

Sinha, A., Namkoong, H., Volpi, R., Duchi, J., 2017. Certifying some distributional robustness with principled adversarial training. arXiv preprint arXiv:1710.10571.

Song, J., Chen, Y., Yue, Y., 2019. A general framework for multi-fidelity Bayesian optimization with Gaussian processes. In: The 22nd International Conference on Artificial Intelligence and Statistics. PMLR, pp. 3158–3167.

Sridhar, S., Bee, S., Bhargava, S., 2014. Membrane-based gas separation: Principle, applications and future potential. Chem. Eng. Dig. 1 (1), 1–25.

Thebelt, A., Wiebe, J., Kronqvist, J., Tsay, C., Misener, R., 2022. Maximizing information from chemical engineering data sets: applications to machine learning. Chem. Eng. Sci. 252, 117469.

Tran, A., Aguirre, A., Crose, M., Durand, H., Christofides, P.D., 2017. Temperature balancing in steam methane reforming furnace via an integrated CFD/data-based optimization approach. Comput. Chem. Eng. (ISSN: 0098-1354) 104, 185–200. http://dx.doi.org/10.1016/j.compchemeng.2017.04.013, URL https://www.sciencedirect.com/science/article/pii/S0098135417301722.

Tsymbalov, E., Panov, M., Shapeev, A., 2018. Dropout-based active learning for regression. In: Analysis of Images, Social Networks and Texts: 7th International Conference, AIST 2018, Moscow, Russia, July 5–7, 2018, Revised Selected Papers 7. Springer, pp. 247–258.

Uppaluri, R.V., Linke, P., Kokossis, A.C., 2004. Synthesis and optimization of gas permeation membrane networks. Ind. Eng. Chem. Res. 43 (15), 4305–4322.

Volpi, R., Namkoong, H., Sener, O., Duchi, J.C., Murino, V., Savarese, S., 2018. Generalizing to unseen domains via adversarial data augmentation. Adv. Neural Inf. Process. Syst. 31.

Wales, D.J., Doye, J.P.K., 1997. Global optimization by basin-hopping and the lowest energy structures of lennard-jones clusters containing up to 110 atoms. J. Phys. Chem. A 101 (28), 5111–5116.

Wang, R., Lin, S., 2021. Pore model for nanofiltration: History, theoretical framework, key predictions, limitations, and prospects. J. Membr. Sci. 620, 118809.

Wessling, M., Mulder, M., Bos, A., van der Linden, M., Bos, M., van der Linden, W., 1994. Modelling the permeability of polymers: a neural network approach. J. Membr. Sci. 86 (1), 193–198.

Yangali-Quintanilla, V., Verliefde, A., Kim, T.-U., Sadmani, A., Kennedy, M., Amy, G., 2009. Artificial neural network models based on QSAR for predicting rejection of neutral organic compounds by polyamide nanofiltration and reverse osmosis membranes. J. Membr. Sci. 342 (1), 251–262.

Zhai, H., Rubin, E.S., 2013. Techno-economic assessment of polymer membrane systems for postcombustion carbon capture at coal-fired power plants. Environ. Sci. Technol. 47 (6), 3006–3014. http://dx.doi.org/10.1021/es3050604, ISSN 0013-936X, 1520-5851. URL https://pubs.acs.org/doi/10.1021/es3050604.

Zhou, K., Yang, Y., Hospedales, T., Xiang, T., 2020. Deep domain-adversarial image generation for domain generalisation. In: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 34. pp. 13025–13032, (07).