J. Plasma Phys. (2025), *vol*. 91, E38
Published by Cambridge University Press

doi:10.1017/S002237782400165X

Using deep learning to design high aspect ratio fusion devices

P. Curvo [1],†, **D.R. Ferreira** [1] **and R. Jorge** [2]

[1]Instituto de Plasmas e Fusão Nuclear, Instituto Superior Técnico, Universidade de Lisboa, 1049-001 Lisbon, Portugal

[2]Department of Physics, University of Wisconsin-Madison, Madison, WI 53706, USA

(Received 5 September 2024; revised 5 December 2024; accepted 6 December 2024)

The design of fusion devices is typically based on computationally expensive simulations. This can be alleviated using high aspect ratio models that employ a reduced number of free parameters, especially in the case of stellarator optimization where non-axisymmetric magnetic fields with a large parameter space are optimized to satisfy certain performance criteria. However, optimization is still required to find configurations with properties such as low elongation, high rotational transform, finite beta and good fast particle confinement. In this work, we train a machine learning model to construct configurations with favourable confinement properties by finding a solution to the inverse design problem, that is, obtaining a set of model input parameters for given desired properties. Since the solution of the inverse problem is non-unique, a probabilistic approach, based on mixture density networks, is used. It is shown that optimized configurations can be generated reliably using this method.

Key words: fusion plasma, plasma confinement

1. Introduction

Stellarators are a type of magnetic confinement fusion device that have toroidal geometry and are non-axisymmetric (see figure 1). Stellarators are inherently current-free, enabling steady-state plasma operation. Because of this, they are one of the leading candidates for future fusion energy power plants (Boozer 2020). In these devices, the magnetic field is twisted by a rotation of the poloidal cross-section of stretched flux surfaces around the torus, and by making the magnetic axis non-planar (Spitzer 1958; Mercier 1964). Due to their complex geometries, stellarators may present difficulties in confining charged particles, especially alpha particles resulting from fusion reactions (Helander 2014). Therefore, they need accurately shaped magnetic fields to confine trapped particles effectively. To achieve this, their configurations are usually optimized using numerical methods. However, the optimization process is complex due to the high-dimensional space of plasma shapes, which includes numerous local minima (Bader *et al.* 2019). While local optimization algorithms can find specific configurations, they

† Email address for correspondence: pedro.mpcurvo@gmail.com

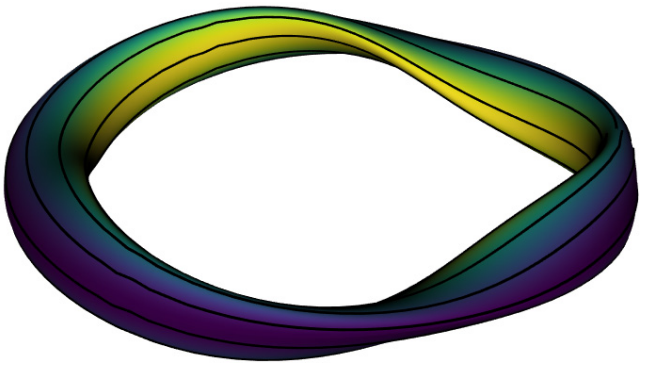

FIGURE 1. Plasma boundary of a quasisymmetric stellarator with three field periods, $n_{\mathrm{fp}} = 3$. The colours represent the magnetic field strength at the boundary and a magnetic field line is shown in black.

do not offer a global view of the solution space. The high dimensionality makes global optimization challenging and renders comprehensive parameter scans impractical (Landreman 2022).

To address these challenges, a near-axis method is commonly employed (Garren & Boozer 1991*a*; Landreman & Jorge 2020; Landreman, Medasani & Zhu 2021). This method makes use of an approximate magnetohydrodynamic (MHD) equilibrium model by expanding in powers of the distance to the axis, leading to a small set of one-dimensional ordinary differential equations (Landreman 2022), therefore reducing the computational costs significantly (Mercier 1964; Solov'ev & Shafranov 1970; Garren & Boozer 1991*b*). As a result, physical intuition can be more easily obtained, and high-resolution multidimensional parameter scans become more feasible, facilitating the generation of extensive databases of stellarator configurations.

In this work, we use the near-axis expansion to second order to generate configurations with finite plasma $\beta = 2\mu_0 p/B^2$ where p is the plasma pressure and B is the magnetic field strength. This allows us to find Mercier stable configurations by selecting devices that satisfy the Mercier criterion, $D_{\mathrm{Merc}} > 0$ (Landreman & Jorge 2020). Such configurations have a positive magnetic well and are robust against certain MHD instabilities. In addition to MHD stability, we will also target configurations with low aspect ratio, small elongation, large rotational transform and quasisymmetry, i.e. good particle confinement (Paul *et al.* 2022). Such quantities can be computed using already available software packages such as pyQSC[1] which receives a set of design parameters (such as axis shape) and computes a set of properties (such as level of quasisymmetry). However, not all configurations are desirable. For most input parameters, the resulting configuration may be unacceptable due to factors such as a too-small volume of plasma, varying levels of quasisymmetry, low rotational transform or overly large elongation. Therefore, it is essential to verify whether the configurations meet specific criteria. This verification can be time-consuming and often requires running the near-axis method multiple times to achieve a viable configuration or resort to numerical optimization. This prompts the question of whether it is possible to perform inverse design, i.e. to determine the input parameters from a given set of desired properties, hence creating a more convenient and efficient method for generating optimized stellarator configurations. This is the main goal of this work.

[1] https://github.com/landreman/pyQSC

Since analytically inverting the equations in the near-axis method is not feasible due to their differential integral character, a practical solution involves employing a machine learning model, specifically a neural network as a universal approximator (Hornik, Stinchcombe & White 1989) to tackle the inverse problem. By training on a dataset of near-axis configurations, the neural network can learn either a forward mapping, from design parameters to configuration properties, or an inverse mapping, from configuration properties to design parameters. However, this inverse problem is ill-posed as multiple sets of design parameters can yield the same configuration properties (this was also observed in the database used in this work). This means that the standard stellarator design formulation is not bijective, in that it lacks a unique, one-to-one correspondence between design parameters and configuration properties. As with other inverse design problems, using a neural network in this context can result in predictions that represent an average of multiple possible design parameter sets for given configuration properties, rather than a specific solution. This averaging effect, described by Bishop (1994), can lead to inaccurate outcomes, as the network generalizes over multiple valid solutions instead of a unique parameter set. To overcome this challenge, we approximate the probability distribution of the design parameters conditioned on the configuration properties. This distribution, which can be multimodal (McLachlan & Basford 1988), allows us to sample design parameters based on the desired configuration properties. To achieve this, a probabilistic machine learning model, namely the mixture density networks (MDNs) model (Bishop 1994), is used to solve the inverse problem of stellarator optimization, together with the near-axis expansion method[2].

2. Physical model

In this section, we describe the near-axis expansion method used to find quasisymmetric stellarators. Quasisymmetry is an effective strategy for confining trapped particles (Nuhrenberg & Zille 1988; Helander 2014) and consists of a continuous symmetry of the magnitude B of the magnetic field $\boldsymbol{B}$ that yields a conserved quantity and enhances particle confinement. Near the magnetic axis, two types of quasisymmetry are possible, namely quasiaxisymmetry, where $B = B(r, \theta)$, and quasihelical symmetry, where $B = B(r, \theta - N\varphi)$. Here, (θ, φ) are the Boozer poloidal and toroidal angles (Boozer 1981), N is an integer, r is defined as $r = \sqrt{2\psi/B_0}$ where ψ represents the magnetic toroidal flux and acts as a radial coordinate and B_0 is the magnetic field strength on the magnetic axis.

The method for generating stellarator configurations in a near-axis expansion complements traditional stellarator optimization, which typically involves parameterizing the boundary shape of a finite aspect ratio plasma and using a three-dimensional MHD equilibrium code to evaluate the objective function. The magnetic field equilibrium and plasma pressure are related via the ideal MHD equation $\boldsymbol{J} \times \boldsymbol{B} = \nabla p$ with $\boldsymbol{J} = \nabla \times \boldsymbol{B}/\mu_0$ the plasma current. Instead, in the near-axis method, we parameterize the axis curve and find the Taylor series coefficients of $\boldsymbol{B}$ in powers of r that allow for quasisymmetry (Garren & Boozer 1991*b*; Landreman & Sengupta 2019; Jorge, Sengupta & Landreman 2020). While the near-axis method is necessarily approximate, it is orders of magnitude faster than standard methods, with a reduced parameter space, therefore allowing for broader parameter scans. Ultimately combining both approaches can be advantageous: the near-axis method can identify viable configurations, which can then be refined through conventional optimization.

[2]The code developed during this work is available at https://github.com/pedrocurvo/MLStellaratorDesign

Input	Description
R_{c1}	First Fourier coefficient of $R(\phi)$ in (2.1*a*,*b*).
R_{c2}	Second Fourier coefficient of $R(\phi)$ in (2.1*a*,*b*).
R_{c3}	Third Fourier coefficient of $R(\phi)$ in (2.1*a*,*b*).
Z_{s1}	First Fourier coefficient of $R(\phi)$ in (2.1*a*,*b*).
Z_{s2}	Second Fourier coefficient of $R(\phi)$ in (2.1*a*,*b*).
Z_{s3}	Third Fourier coefficient of $R(\phi)$ in (2.1*a*,*b*).
$\bar{\eta}$	First-order Taylor series coefficient of B in (2.2)
B_{2C}	Second-order Taylor series coefficient of B in (2.2).
n_{fp}	Number of field periods of the device.
p_2	Second-order Taylor series coefficient of p in (2.3).

TABLE 1. Input parameters for the near-axis model.

In the near-axis expansion method, the magnetic axis $\boldsymbol{r}_0 = R(\phi)\boldsymbol{e}_R + Z(\phi)\boldsymbol{e}_z$ is typically represented in cylindrical coordinates (R, Z, ϕ) using a finite Fourier series,

$$R(\phi) = \sum_{n=0}^{N_F} R_{cn}\cos(n_{\mathrm{fp}}n\phi), \quad Z(\phi) = \sum_{n=1}^{N_F} Z_{sn}\sin(n_{\mathrm{fp}}n\phi), \tag{2.1a,b}$$

where n_{fp} is the number of field periods and a finite maximum Fourier number N_F is chosen. Stellarator symmetry is assumed. The remaining input parameters are the coefficients of the magnetic field strength

$$B = B_0[1 + r\bar{\eta}\cos(\vartheta)] + r^2[B_{20} + B_{2c}\cos(2\vartheta)], \tag{2.2}$$

namely B_0, $\bar{\eta}$ and B_{2c}, and the plasma pressure

$$p = p_2 r^2, \tag{2.3}$$

with $\vartheta = \theta - N\varphi$. Here, B_0 is chosen to be 1 T and, following Landreman & Sengupta (2019), B_{20} is taken to be a function of φ, with exact quasisymmetry corresponding to B_{20} being a scalar constant. The total plasma current on-axis is taken to be $I_2 = 0$. Henceforth, the input parameter space for optimization consists of $\{R_{cn}, Z_{sn}, n_{\mathrm{fp}}, \bar{\eta}, B_{2c}, p_2\}$, as described in table 1. The output properties are presented in table 2. The magnetic field equilibrium and plasma pressure are related via the ideal MHD equation $\mathbf{J}\times\mathbf{B} = \nabla p$ with $\mathbf{J} = \nabla\times\mathbf{B}/\mu_0$ the plasma current. The proxy used for the maximum plasma radius is $r_{\mathrm{singularity}}$ (see Landreman 2021), and the proxy used for the plasma β is the volume-averaged $\langle\beta\rangle = -\mu_0 p_2 r^2_{\mathrm{singularity}}/B_0$ (see Landreman 2022). The number of degrees of freedom used to optimize stellarator devices has then been reduced from typically $\sim$100 plasma boundary coefficients to $\sim$10 near-axis coefficients.

Although the near-axis expansion substantially reduces the number of free parameters, the optimization process may be computationally expensive, depending on the target parameters. Furthermore, it is necessary to compute the mapping between Boozer and Cartesian coordinates for a given surface to identify if the configuration possesses self-intersecting surfaces, making the process both time-consuming and resource-intensive. For this work, a viable configuration is one that meets the specific criteria outlined in table 3. Such parameters are similar to the ones outlined in Landreman (2022). Those parameters also benefit the overall stability of the stellarators, e.g. $L_{\nabla B}$

Output	Description
axis length	Length of the magnetic axis
ι	Rotational transform on-axis.
max elongation	Ratio of the major to minor semiaxis cross-section.
$\min L_{\nabla B}$	Scale length of the magnetic field gradient.
$\min R_0$	Minimum of the radial coordinate R of the axis.
$r_{\text{singularity}}$	Maximum allowed radial coordinate for the boundary.
$L_{\nabla\nabla B}$	Scale length of the magnetic field Hessian.
$B20_{\text{variation}}$	Degree of quasisymmetry.
β	Volume-averaged plasma beta $\langle\beta\rangle = -\mu_0 p_2 r^2_{\text{singularity}}/B_0^2$.
$D_{\text{Merc}} \times r^2$	Lowest-order Mercier criterion coefficient.

TABLE 2. Output parameters from the near-axis model.

Output Property	Range
axis length	> 0.0
$\lvert\iota\rvert$	≥ 0.2
max elongation	≤ 10.0
$\min L_{\nabla B}$	≥ 0.1
$\min R_0$	≥ 0.3
$r_{\text{singularity}}$	≥ 0.05
$L_{\nabla\nabla B}$	≥ 0.1
$B20_{\text{variation}}$	≤ 5.0
β	$\geq 10^{-4}$
$D_{\text{Merc}} \times r^2$	> 0.0

TABLE 3. Criteria for good stellarators with the major radius fixed at $R_{c0} = 1$m and magnetic field on-axis of $B_0 = 1$ T.

is positively correlated with the coil-to-plasma distance, as demonstrated by Kappel, Landreman & Malhotra (2024), hence by constraining to larger values, we can obtain solutions with improved stability. Informally, we will refer to stellarators that meet these criteria as good stellarators, and those that do not as bad stellarators.

3. Mixture models and density networks

When dealing with non-unique inverse problems, we often encounter situations where there are multiple possible solutions for a given input. To effectively address these problems, it is essential to have a statistical distribution over the possible solutions rather than a single deterministic answer. The normal distribution is a common way to construct probability distributions, but in cases with multiple solutions, we require a multimodal distribution, which can be achieved through a mixture model (McLachlan & Basford 1988). This model provides concentrated probabilities at various points, representing the different solutions. In this section, we describe the probabilistic models used in this work, namely mixture models, Gaussian models and multivariate Gaussian mixtures.

A mixture model (McLachlan & Basford 1988) is a statistical tool used to describe a population comprised of multiple subgroups without prior knowledge of individual data point memberships. It constructs a combined probability distribution for the entire

population by integrating the probability distributions of each subgroup. Mixture models enable us to understand the characteristics of these subgroups using data from the entire population, even when the subgroup for each data point is unknown. These models are typically applied in clustering tasks, where data points are grouped into clusters, and density estimation, which involves estimating the distribution of the data itself.

A typical finite-dimensional mixture model $p(y \mid \lambda)$ is a combination of simple distributions $p_i(y)$ that can be represented as follows:

$$p(y \mid \lambda) = \sum_{i=1}^{K} \pi_i p_i(y), \tag{3.1}$$

where p_i is the ith component distribution, π_i is the mixture weight of the ith component and K is the number of components in the mixture. The mixture weights are non-negative and sum to 1, i.e. $0 \leq \pi_i \leq 1$ and $\sum_i \pi_i = 1$.

To better understand mixture models, we re-express the model in a hierarchical framework. This involves introducing a latent variable $z \in \{1, \ldots, K\}$ representing the component from which each data point is generated. This hierarchical approach not only provides a clear structure but also facilitates the inference process. Henceforth, each data point y is associated with a latent variable z that indicates the component it originates from. The prior distribution over the latent variables is governed by the parameters $\pi = (\pi_1, \ldots, \pi_K)$, where π_i represents the probability that a data point belongs to component i. Formally, we write

$$p(z = k \mid \lambda) = \pi_k. \tag{3.2}$$

Given that a data point y comes from component i, it is generated according to a component-specific distribution $p(y \mid \lambda_i)$. Thus, the conditional distribution of y given the latent variable z and the parameters λ is

$$p(y \mid z = i, \lambda) = p_i(y) = p(y \mid \lambda_i). \tag{3.3}$$

The complete set of parameters for this hierarchical model is $\lambda = (\pi_1, \ldots, \pi_K, \lambda_1, \ldots, \lambda_K)$, where π represents the mixing proportions and λ_i represents the parameters specific to the ith component.

The generative process for the data involves first selecting a specific component z and then drawing a sample y from the chosen component. By marginalizing over the latent variable, i.e. by summing over all possible states of z, we obtain the marginal distribution $p(y \mid \lambda)$ of the observed data

$$p(y \mid \lambda) = \sum_{i=1}^{K} p(z = i \mid \lambda) p(y \mid z = i, \lambda) = \sum_{i=1}^{K} \pi_i p(y \mid \lambda_i). \tag{3.4}$$

This formulation allows us to model complex, multimodal data distributions effectively, capturing the diverse characteristics of the data through the combined influence of multiple simple components.

One of the most widely used mixture models, due to its simplicity and effectiveness in modelling complex data distributions, is the Gaussian mixture model (GMM), a specific type of mixture model where the component distributions are Gaussian distributions. The

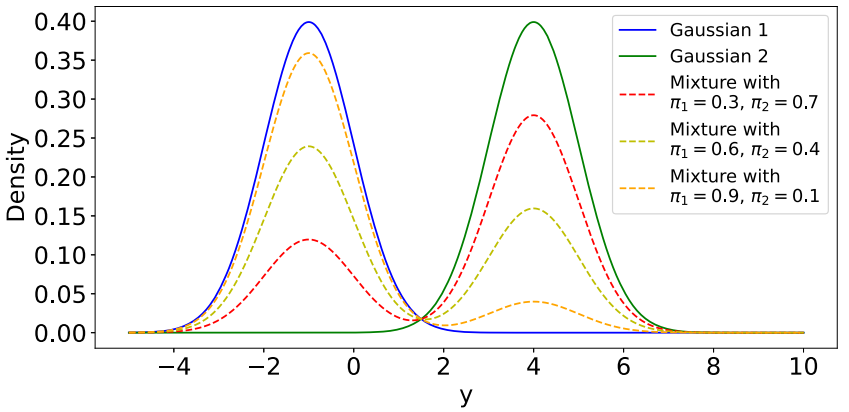

FIGURE 2. Example of mixture models with two components, each represented by a Gaussian distribution, illustrating how a mixture model forms from two distributions and the influence of mixture weights on data distribution modelling.

GMM is defined as

$$p(y \mid \lambda) = \sum_{i=1}^{K} \pi_i \mathcal{N}(y \mid \mu_i, {\sigma_i}^2), \tag{3.5}$$

where $\mathcal{N}(y \mid \mu_i, {\sigma_i}^2)$ is a Gaussian distribution with mean μ_i and variance σ_i^2, namely

$$\mathcal{N}(y \mid \mu_i, \sigma_i^2) = \frac{1}{\sqrt{2\pi\sigma_i^2}} \exp\left(-\frac{(y-\mu_i)^2}{2\sigma_i^2}\right). \tag{3.6}$$

The GMM can approximate any continuous distribution to any arbitrary degree of accuracy by using a sufficient number of components (Goodfellow, Bengio & Courville 2016). It is particularly useful for clustering and density estimation tasks, where the data distribution is complex and multimodal. An example of a GMM with two components and different mixture weights is shown in figure 2. This figure illustrates how the GMM combines two Gaussian distributions with distinct means and variances, demonstrating three separate mixtures where each mixture is characterized by specific mixing coefficients, π_1 and π_2. These coefficients determine the relative influence of each Gaussian component in modelling the observed data distribution, showcasing the GMM's ability to represent complex data patterns through weighted combinations of simpler Gaussian distributions.

A generalization of the one-dimensional Gaussian distribution to multiple dimensions is called multivariate normal (MVN), also known as the multivariate Gaussian distribution. The MVN is one of the most widely used joint probability distributions for continuous random variables (Murphy 2023). This popularity is due to its mathematical convenience and versatile applicability across a wide range of scenarios. Indeed, if we know the mean and variance of a dataset, but do not have other information such as, for example, skewness, kurtosis, domain-specific constraints, temporal dependencies, spatial correlations or known outliers, the Gaussian distribution is the most unbiased choice because it maximizes entropy under these constraints (Cover & Thomas 2012).

The multivariate Gaussian distribution is defined as

$$\mathcal{N}(y \mid \mu, \Sigma) = \frac{1}{(2\pi)^{D/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(y-\mu)^{\mathrm{T}}\Sigma^{-1}(y-\mu)\right), \tag{3.7}$$

where y is a D-dimensional vector, $\mu = \mathbb{E}[y]$ is the mean vector and $\Sigma = \mathrm{Cov}[y]$ is the $D \times D$ covariance matrix. The normalization constant is given by $(2\pi)^{D/2}|\Sigma|^{1/2}$ to ensure that the distribution has a unit volume integral. The covariance matrices Σ, in the context of multivariate Gaussian distributions, can be categorized into three groups. First, full covariance matrices are matrices with $D(D+1)/2$ parameters, which are symmetric and positive definite, allowing them to capture existing correlations between variables. Second, diagonal covariance matrices are matrices with D parameters, which are diagonal with zero off-diagonal elements. These matrices assume that the variables are independent of each other. Third, there are spherical covariance matrices with one parameter, which are a scalar multiple of the identity matrix in the form $\sigma^2 I_D$. These matrices assume that the variables have equal variance and are isotropic.

The full covariance matrix is the most general form of the multivariate Gaussian distribution, and it can represent existing correlations between variables. However, it is also the most computationally expensive, since it requires the inversion of a $D \times D$ matrix. The diagonal covariance matrix, on the other hand, assumes that the variables are independent, and the spherical covariance matrix assumes that the variables are isotropic, both simplifying computation but potentially oversimplifying real-world correlations.

Using multiple MVNs as components in a mixture model results in what is known as the multivariate GMM (MGMM). This model is a generalization of the GMM to the multivariate case and is defined as

$$p(y \mid \lambda) = \sum_{i=1}^{K} \pi_i \mathcal{N}(y \mid \mu_i, \Sigma_i), \tag{3.8}$$

where $\mathcal{N}(y \mid \mu_i, \Sigma_i)$ is the MVN distribution with mean vector μ_i and covariance matrix Σ_i. This capability allows MGMMs to accurately capture complex data structures where variables are interdependent, providing a more realistic representation of real-world data distributions (McLachlan & Peel 2004). Unlike univariate models that assume independence, MGMMs are particularly effective in scenarios requiring flexible and scalable modelling of multidimensional data, such as in image processing (Bueno & Kragic 2006). By accommodating these correlations, MGMMs enhance clustering and classification tasks, enabling more meaningful groupings in several applications where multiple correlated features influence outcomes. Furthermore, MGMMs excel in accurate density estimation for multivariate data, which is crucial in fields like environmental science for modelling spatial distributions of pollutants or genetics to analyse complex gene expression profiles.

This work involves the use of multivariate data containing intrinsic correlations between the variables, making MGMMs one of the best options to accurately estimate the density of our data. By leveraging the ability of MGMMs to model these correlations through covariance matrices, we can achieve a more realistic and precise representation of the data distribution, which is crucial for our analysis. Furthermore, we can enhance our modelling capabilities by combining the approximation properties of neural networks with the flexibility of mixture models (Bishop 1994). This approach allows us to model complex density estimations without requiring any prior knowledge of their distributions.

4. Mixture density networks

Neural networks are computer models inspired by the structure of the human brain (Hornik *et al.* 1989). They are made up of layers of connected neurons or nodes. Such layers are used to process input data, with each neuron applying an activation function and a weighted sum to produce an output. Through training, neural networks can discover intricate patterns and relationships in data.

However, in problems involving continuous variables where the same input values may produce different output values, neural networks tend to predict the mean of the target variable. This can be regarded as an approximation to the conditional average of the target variable given the input. This conditional average provides a very limited description of the statistical properties of the data and is often inadequate for many applications. This is particularly true for non-unique inverse problems, where a conventional neural network with a least-squares approach might yield an inaccurate solution as the mean of multiple, possibly more accurate solutions.

In our case, averaging parameters such as R_{cn} and Z_{sn} tends to yield suboptimal results due to their complex interdependencies. Both variables exhibit multimodal distributions centred around symmetric values. Averaging these values tends to converge towards zero, which may lead to the generation of bad stellarator designs. Consequently, there is a need for a neural network to be capable of probabilistically selecting R_{cn} or Z_{sn} from their respective subdistributions, depending on the context. This leads to the use of a probabilistic model capable of representing multimodal distributions. Such a model would not average the distributions but instead sample from them, thereby preserving the distinct characteristics of each mode and enabling more accurate predictions and good stellarator designs.

To address these requirements, MDNs (Bishop 1994) present a compelling solution. Mixture density networks are a class of neural networks designed to overcome the limitations of conventional neural networks in modelling complex, multimodal data distributions. They combine the flexibility of neural networks with the robustness of mixture models, where the neural network estimates the parameters for the mixture model. Mixture density networks allow a neural network to learn arbitrary conditional distributions as opposed to only learning the mean. This enables MDNs to provide a more comprehensive and accurate modelling approach for complex data distributions.

In MDNs, the probability density of the target data is represented as a linear combination of components, as in (3.1). Various choices for these components are possible, but for the purpose of this work, we focus on MGMMs, as in (3.8), to approximate the conditional distribution of the target variables given the inputs, because, as seen in § 3, it effectively captures complex data structures where variables are interdependent, and excels in accurate density estimation for multivariate data, which is crucial for our case.

For any given values of the input x, the MDN provides a systematic method for modelling an arbitrary conditional distribution $p(y \mid x)$. The model parameters, namely the mixing coefficients π_i, the mean vectors μ_i and the covariance matrices Σ_i are modelled as continuous functions of x. This is achieved by having π_i, μ_i, Σ_i as the outputs of a conventional neural network, which takes x as its input. The combined structure of a feed-forward network and a mixture model is the essence of an MDN. The basic structure of the feedforward neural network responsible for modelling the parameters of the mixture as a continuous function of the input parameters is illustrated in figure 3. This architecture enables the network to dynamically adjust the mixture parameters based on the input data while capturing complex, nonlinear relationships in the data.

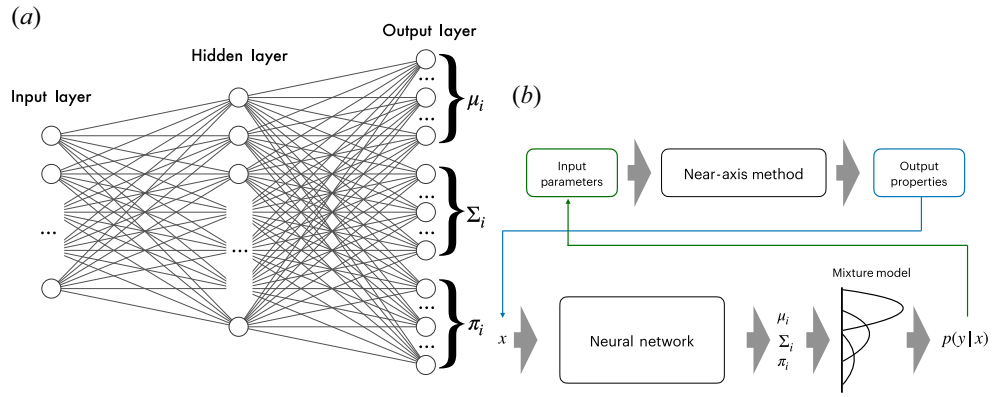

FIGURE 3. (*a*) Sketch of the neural network architecture used in this work to estimate the parameters of a mixture model. (*b*) Architecture of the mixed density network as an inverse model for the near-axis method.

Layer	Size	Activation function
Input	10	—
Hidden 1	64	tanh
Hidden 2	128	tanh
Hidden 3	256	tanh
Hidden 4	512	tanh
Hidden 5	1024	tanh
Hidden 6	2048	tanh
Output	4092	tanh for μ and $\Sigma_{ij,j>i}$, ELU + 1 for $\Sigma_{ij,i=j}$, softmax for π

TABLE 4. Layers of the mixture density network used in this work.

By choosing a mixture model with a large enough number of components, and a neural network with a large enough number of hidden units (Uzair & Jamil 2020), the MDN can approximate any conditional density $p(y\,|\,x)$ as closely as desired (Lu & Lu 2020). In this work, we use a mixture model with 62 components. This choice was empirically determined to provide an optimal balance between model complexity and performance. It was observed that increasing both the number of layers and the width of each layer, as well as incorporating more components, provided severe improvements in the model's performance. The architecture of the mixture density network used in this work is illustrated in figure 3 and in table 4, showcasing the detailed configuration and activation functions employed at various layers.

The neural network's input layer contains 10 neurons, corresponding to the 10 input parameters. This is followed by a series of hidden layers with progressively increasing sizes, namely 64, 128, 256, 512, 1024 and 2048. The output layer consists of 4092 nodes that are allocated as follows: 62 nodes represent the mixture weights of the 62 components; 620 nodes represent the mean vector of the 10 outputs for each of the 62 components; 3410 nodes represent the 55 parameters (the upper triangular part) of the 10×10 covariance matrix for each of the 62 components.

The calculation for the number of parameters for each covariance matrix uses the formula $D(D+1)/2$, where D is the dimension of the covariance matrix. With $D = 10$,

each covariance matrix requires 55 parameters, resulting in a total of 3410 parameters for the 62 components. We employ the hyperbolic tangent, tanh, activation function to the hidden layers to prevent numerical issues that may arise from large values propagating through the network, which could lead to unstable computations and vanishing gradients, causing non-positive definite covariance matrices.

In the output layer, the neural network uses different activation functions tailored to the nature of each parameter type. The means of the Gaussian components are mapped to the range $]-1, 1[$ using the tanh activation function, benefiting from the data normalization process (a standard scaler) which is applied to the input data, i.e. when normalized, we expect the data to become centred around zero, which agrees with the zero-centred nature of the tanh activation function and, at the same time, limits the potential for large numerical values propagating through the network. The mixture weights are computed using the softmax function (Bridle 1990; Jacobs *et al.* 1991) to ensure that they sum to unity. The covariance matrices are computed with the diagonal elements using a modified exponential linear unit (ELU) (Clevert, Unterthiner & Hochreiter 2016) function (ELU + 1) function to ensure positivity and the off-diagonal elements are constrained between $]-1, 1[$ using the tanh activation function, for the same reasons presented before. With this established architecture, the next step involves training the MDN on a dataset of stellarator configurations to adjust the network weights, thereby enhancing the model's predictive capabilities.

5. Data generation and training

To train the MDN, we generate a dataset of stellarators using the near-axis expansion method. The dataset is a collection of records containing the input parameters provided to the near-axis method and corresponding output properties generated from these inputs. These are listed in tables 1 and 2.

To generate the dataset, we sample the input parameters from uniform distributions, with the ranges listed in table 5 and find the output parameters listed in table 2. The range of parameters R_{c2}, R_{c3}, Z_{c2} and Z_{c3} follows the empirical observation in previous near-axis configurations and in the parameter scans done here that the Fourier coefficients generally decrease with increasing order. This allows us to restrict the database to feasible designs. By sampling the input parameters from uniform distributions, we find that most configurations consist of bad stellarators. In fact, by applying the set of criteria shown in table 3, it is seen that the percentage of good stellarators is extremely low, with only one in approximately 100 000 samples found to comply with all the desired criteria. This illustrates how difficult it is to find good stellarators by random search, and is one of the main drivers for the use of an inverse model to find the input parameters from a set of desired properties.

Following the generation of the dataset, we begin by normalizing the dataset using a standard scaler to account for the different scales of the input and output parameters. The dataset was then split into training and validation sets with an 80 % and 20 % split, respectively. Next, we initialize the weights of the neural network using the Xavier Glorot initialization method (Glorot & Bengio 2010), which is effective for deep neural networks as it helps prevent vanishing or exploding gradients during training. Additionally, we employ the Adam optimizer (Kingma & Ba 2017) with a learning rate of 10^{-3} and a batch size of 10 000 samples.

The output properties are then sampled from the mixture model. We compute the negative log-likelihood of these samples, which serves as the loss function to be minimized during training. Since we use a mixture model composed of multiple Gaussian

Input parameter	Range
R_{c1}	$[-1, 1]$ $(= [-R_{c0}, R_{c0}])$
R_{c2}	$[-\lvert R_{c1}\rvert, \lvert R_{c1}\rvert]$
R_{c3}	$[-\lvert R_{c2}\rvert, \lvert R_{c2}\rvert]$
Z_{s1}	$[-1, 1]$ $(= [-R_{c0}, R_{c0}])$
Z_{s2}	$[-\lvert Z_{s1}\rvert, \lvert Z_{s1}\rvert]$
Z_{s3}	$[-\lvert Z_{s2}\rvert, \lvert Z_{s2}\rvert]$
$\lvert\bar{\eta}\rvert$	$[0.01, 3.0]$
$\lvert B_{2C}\rvert$	$[0.01, 3.0]$
n_{fp}	$[0, 10]$
p_2	$[-4 \times 10^6, 0.0]$

TABLE 5. Uniform distributions defining the input parameter ranges used for dataset generation. Each parameter is sampled within the interval shown in the second column.

components, the loss function is given by

$$\mathrm{Loss} = -\frac{1}{N}\sum_{j=1}^{N} \log\left(\sum_{i=1}^{K} \pi_i \mathcal{N}(y_j \mid \mu_i, \Sigma_i)\right), \tag{5.1}$$

where N is the number of samples, i.e. the batch size, and y_j is an output vector.

Despite using the Adam optimizer (Kingma & Ba 2017), the training process was more challenging than anticipated due to numerical instabilities, such as vanishing gradients, that caused the covariance matrices to become non-positive definite. To address this issue, a multistep learning rate scheduler was employed, which adjusted the learning rate at specific training epochs (10, 20, 30, 40 and 50) by a factor of 0.5. This schedule initially allowed the model to explore the parameter space with a higher learning rate, then gradually refined as training progressed. By reducing the learning rate in steps, the model avoided abrupt changes in parameter updates, leading to a more stable convergence. The loss and validation curves can be seen in figure 4. Notably, the curves indicate that as the learning rate decreases, the loss function values also decrease. This trend suggests that lower learning rates contribute to a more stable and gradual convergence, resulting in better model performance and lower loss.

However, as mentioned earlier, the percentage of good stellarators obtained by random sampling was very low. To address this issue, we adopted an iterative training approach, where the trained model was used to support the generation of a new dataset. This new dataset can be used to retrain the model, which in turn can be used to support the generation of a further dataset.

The uniform distributions in table 3 have been used only once to generate the initial dataset. Once the model is trained, we use it to draw samples of input parameters of good stellarators to then provide to the near-axis method. At first, the model only had a small number of good stellarators (0.04 % after the first training). However, over the course of several training iterations, the percentage of good stellarators in the dataset keeps increasing. This is shown in table 6 where, at the end of the fifth iteration, the percentage of good stellarators reaches approximately 20 %. The resulting model is analysed in the next section.

The evolution of the distribution of the R_{c1} variable during the training of the model is shown in figure 5. The initial uniform distribution used to create the dataset

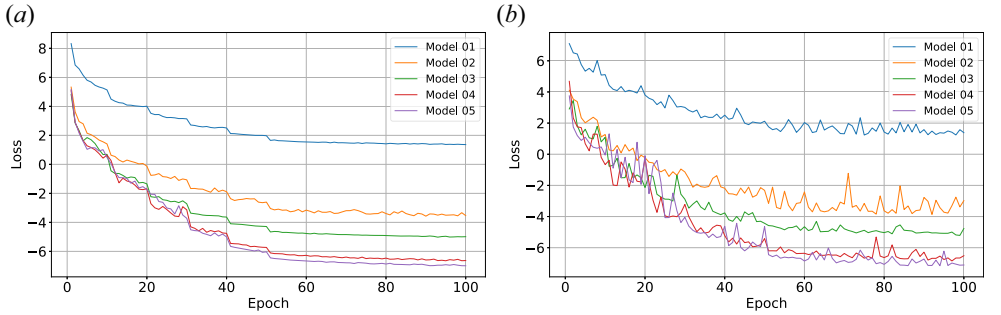

FIGURE 4. Loss (*a*) and validation loss (*b*) curves during training for the different models. The initial learning rate, 1×10^{-3}, was decreased with a scheduler in epochs 10, 20, 30, 40, 50 with $\gamma = 0.5$.

Dataset	Good stellarators (%)
Before training (uniform sampling)	0.0018
After the first training iteration	0.0406
After the second training iteration	1.3788
After the third training iteration	9.0024
After the fourth training iteration	12.3903
After the fifth training iteration	20.2670

TABLE 6. Percentage of good stellarators in each iteration dataset.

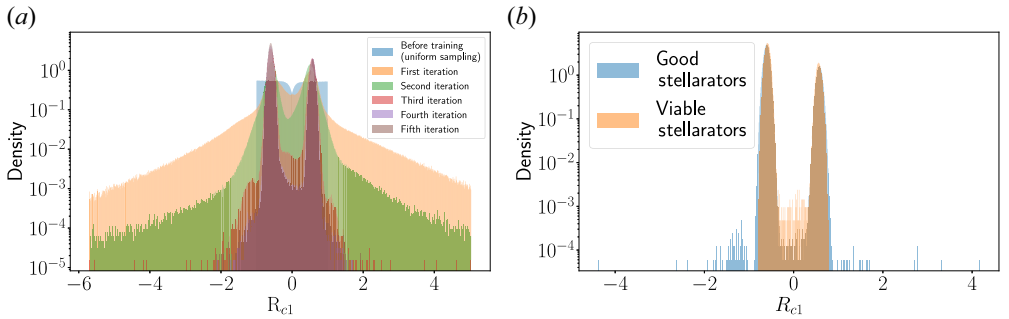

FIGURE 5. (*a*) Distribution of the R_{c1} variable during the iterative process. (*b*) Distribution of the R_{c1} variable for the good stellarators and the viable stellarators.

gradually transitions to a bimodal Gaussian-like distribution. This transformation aligns more closely with our objective of focusing on the region where good stellarators are found. This transition also simplified the training of the model, as GMMs can more effectively approximate it compared with a uniform distribution, which would require more components with wider covariances. Here, we find that the final distribution of the R_{c1} variable has two peaks, one around -0.8 and another around 0.8, with a higher peak at -0.8.

Desired properties (input to MDN)		Design parameters (output of MDN /input to pyQSC)		Actual properties (output of pyQSC)
axis length	12.23	R_{c1}	−0.492168	12.67
ι	−2.24	R_{c2}	0.003776	−2.07
max elongation	6.56	R_{c3}	−0.000132	8.20
min $L_{\nabla B}$	0.41	Z_{s1}	−0.652899	0.44
min R0	0.48	Z_{s2}	0.006861	0.51
$R_{\text{singularity}}$	0.14	Z_{s3}	−0.005334	0.11
$L_{\nabla\nabla B}$	0.25	n_{fp}	3	0.28
$B20_{\text{variation}}$	1.68	$\bar{\eta}$	−0.844595	3.94
β	0.005	B2c	1.662730	0.003
$D_{\text{Merc}} \times r^2$	0.09	p2	−162 627	−0.15

TABLE 7. Sample results for given desired properties. A random stellarator configuration was selected from the test dataset, and its properties were used as input to the model to predict the design parameters. These predicted design parameters were then fed into the near-axis method, which returned the actual properties. The resulting actual properties closely matched the desired ones.

6. Model performance

We now show how the model can be used to predict the input parameters needed to obtain optimized stellarators with desired output properties. First, the user provides the desired properties such as the volume-averaged plasma β and rotational transform, and the model produces the design parameters that are likely to yield those properties such as magnetic axis and $\bar{\eta}$. Then, the user feeds the predicted design parameters to the near-axis expansion method, to generate the corresponding properties. Finally, the user verifies that the actual properties generated by the near-expansion method agree with the desired properties. A randomly selected example from the dataset is presented in table 7, while an example using the frontier conditions from table 3 is shown in table 8.

However, while the model is able to yield configurations that satisfy the requirements listed in table 3, it is not guaranteed that all configurations have a set of nested, non-intersecting flux surfaces up to the parameter $r_{\text{singularity}}$. This is because $r_{\text{singularity}}$ is only a proxy for the minimum aspect ratio of the device. Only by computing the surface in Cartesian coordinates, as opposed to the near-axis Boozer coordinates used throughout this work, can we verify the existence of such a surface. Such an evaluation is crucial in using such configurations in practice. We then take all the good stellarators and generate a surface at a radial distance of $r = 0.1R_{c0}$. Here, the existence of such a surface is defined as the existence of a numerical solution of the mapping from the toroidal Boozer coordinate φ on-axis to a cylindrical angle ϕ off-axis with tolerance at or below 10^{-15} after a maximum of 1000 iterations. We will refer to the good stellarators that meet this additional criterion as viable stellarators.

Next, keeping the standard normalization on the dataset, we employed the Huber loss and the mean absolute error (MAE) as evaluation metrics to compare the predicted output properties from the model with the output properties from the near-axis model on 10 000 samples. Both are metrics used in regression tasks to quantify the difference between predicted values and actual observations. Huber loss combines the advantages of MAE for

Desired properties (input to MDN)		Design parameters (output of MDN /input to pyQSC)		Actual properties (output of pyQSC)
axis length	0.00	R_{c1}	0.614602	7.99
ι	0.20	R_{c2}	−0.058358	−0.79
max elongation	10.0	R_{c3}	0.020746	14.65
min $L_{\nabla B}$	0.10	Z_{s1}	0.804627	0.21
min R0	0.30	Z_{s2}	0.013770	0.31
$R_{\text{singularity}}$	0.05	Z_{s3}	0.013673	0.03
$L_{\nabla\nabla B}$	0.10	n_{fp}	1	0.10
$B20_{\text{variation}}$	5.00	$\bar{\eta}$	0.509180	5.51
β	0.001	B2c	−1.257980	0.00014
$D_{\text{Merc}} \times r^2$	0	p2	−154 446	0.10

TABLE 8. Sample results for given desired properties that were the boundary conditions in table 3. The properties of the given stellarator were used as input to the model to predict the design parameters. These predicted design parameters were then fed into the near-axis method, which returned the actual properties. The resulting actual properties closely matched the desired ones.

	Viable		Good		Bad	
	Metric		Metric		Metric	
Viable	Huber loss	MSE	Huber loss	MSE	Huber loss	MSE
axis length	0.031	0.0618	0.0342	0.083	1.33	10.2
ι	0.0267	0.0534	0.0233	0.0495	0.909	4.2
max elongation	0.000456	0.0113	0.000326	0.068	0.138	17.8
min $L_{\nabla B}$	0.266	0.735	0.227	0.578	0.869	3.82
min R_0	0.00617	0.0531	0.00665	0.274	2.72	32.6
$r_{\text{singularity}}$	0.632	1.72	0.907	3.11	0.0098	0.0292
$L_{\nabla\nabla B}$	0.432	1.12	0.415	1.08	0.149	0.366
$B20_{\text{variation}}$	0.000604	0.0046	8.99×10^{-5}	0.000218	3.9	37.4
β	0.321	1.1	0.658	3.12	0.00454	0.0165
$D_{\text{Merc}} \times r^2$	3.4×10^{-11}	5.94×10^{-5}	1.65×10^{-10}	9.94×10^{-12}	83.7	124.3
Average	0.172	0.486	0.227	0.837	9.37	23.073

TABLE 9. Model accuracy on bad, good and viable stellarators.

robustness to outliers and mean squared error (MSE) for sensitivity to small errors. The results for bad, good and viable stellarators are presented in table 9.

As illustrated in table 9 for viable stellarators, the model accuracy was found to be satisfactory. For the variables axis length, ι, max elongation, $B20_{\text{variation}}$, and $D_{Merc} \times r^2$, the model showed a good performance, evidenced by a low Huber and MSE losses, 0.172 and 0.486, respectively, with the MSE being higher than the Huber loss, as expected. Regarding the variables min $L_{\nabla B}$, min R_0 and $L_{\nabla\nabla B}$, the model displayed moderate accuracy under the Huber loss metric. However, the MSE was higher, indicating that the model underperforms in these variables. The variables β and $r_{\text{singularity}}$ exhibited the poorest accuracy, with both metrics indicating suboptimal results. A possible explanation for this

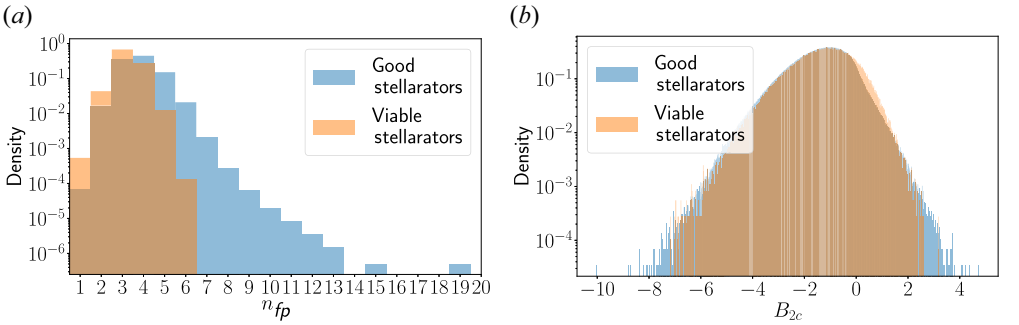

FIGURE 6. Distribution of (*a*) number of field periods n_{fp} and (*b*) B_{2c} variable for good and viable stellarators.

outcome might be due to trade-offs in variable correlations, i.e. maximizing performance for some variables may require sacrificing accuracy in others.

Beyond the model performance, understanding the relationships between variables is crucial for interpreting the behaviour of output properties and their interdependencies. This knowledge significantly influences how the model should be used to predict input parameters. When output properties are strongly correlated, the model must carefully balance these correlations to achieve the desired outputs. Additionally, being aware of the distribution of variables is essential to ensure the model operates within familiar data spaces; otherwise, it may perform poorly. Therefore, analysing the distributions of the variables and their correlations is vital.

Henceforth, the iterative training process described in § 5 was monitored to check if the distributions of both input and output variables were being restricted to a narrower space, which was to be expected since we wanted to restrict the dataset to the space of good stellarators. We evaluated the distribution of variables for the dataset containing all the good stellarators and all the viable stellarators. We show in figures 5 and 6 the ones that provide a better understanding of the dataset and that are more relevant.

The distribution of the n_{fp} variable for both the good stellarators and the viable stellarators is depicted in figure 6. The data shows that good stellarators tend to cluster around $n_{\mathrm{fp}} = 4$, although there is a notable variation with several other n_{fp} values present. An aspect of these results is that the model, despite being trained on a dataset where the n_{fp} ranged from 1 to 10, successfully predicted n_{fp} values for good stellarators that exceeded this range. As illustrated in figure 6, there are configurations with n_{fp} values extending up to 19.

For the viable stellarators, the distribution of the number of field periods, n_{fp}, is more narrowly centred around the value of 3, and none of the configurations exhibit n_{fp} values above 6. This suggests a more constrained and specific range for n_{fp} in the viable stellarator subset, indicating that these configurations are more consistent in this regard. The fact that an optimized stellarator with a higher number of field periods is hard to find, as it was also observed in Landreman (2022), may be related to the fact that such n_{fp} usually require a significant excursion of the axis and an associated larger axis length. Furthermore, the recent study by Kappel *et al.* (2024) has shown a correlation between the number of field periods n_p and $L_{\nabla B}$, indicating that small values of n_p may lead to more optimized configurations.

We also examine the B_{2c} parameter. The distribution of this variable for both good stellarators and viable stellarators is illustrated in figure 6. The data reveals that B_{2c} exhibits

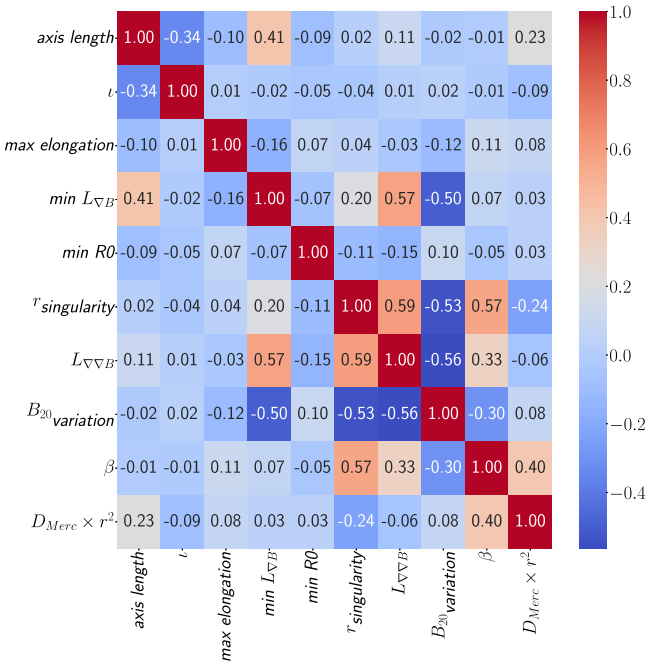

FIGURE 7. Correlation matrix for the output properties of good stellarators using the Spearman coefficient. The values range from −1 to 1, where negative values indicate negative correlations and positive values indicate positive correlations. The absolute values represent the correlation strength: values from 0 to 0.3 indicate a weak correlation; from 0.4 to 0.6 indicate a moderate correlation; from 0.7 to 1 indicate a strong correlation.

a noticeable shift towards negative values. This indicates a distinct characteristic in the B_{2c} distribution for good stellarators compared with the overall dataset.

The observed shifts in the distributions of variables for the good stellarators and the viable stellarators, whether towards negative or positive values, suggest that maximizing or minimizing certain variables can influence others in similar or opposing ways. This prompts us to evaluate the correlations between variables. While correlation does not imply causation, it provides valuable insights into the relationships between variables. We show in figure 7 the correlation matrix for the output properties of good stellarators, which is similar to viable stellarators. This matrix reveals a strong positive correlation between $r_{\text{singularity}}$ and $L_{\nabla\nabla B}$. This indicates that as the axis length increases, the maximum elongation also increases. Conversely, the $\min L_{\nabla B}$ and $B20_{\text{variation}}$ display a strong negative correlation, meaning that an increase in the minimum $\min L_{\nabla B}$ results in a decrease in the minimum $B20_{\text{variation}}$. These relationships significantly impact model performance, as the model must balance them to achieve the desired properties. As an example, if a user requests a stellarator with a high $\min L_{\nabla B}$ and a low $B20_{\text{variation}}$, the model must navigate the positive correlation between these properties. Since they are not independent, the model must find a compromise to generate appropriate input parameters that align with the desired output properties.

7. Conclusions

This work introduces an MDN designed to tackle the inverse stellarator optimization problem using the near-axis method. The model was trained on a dataset of near-axis

configurations generated through the near-axis expansion method. However, the dataset initially contained a very low percentage of desirable stellarators, specifically only 0.001 %. To address this limitation, an iterative data augmentation technique was employed. This iterative approach successfully enhanced the representation of high-quality stellarators within the dataset, thereby improving the model's capability to predict parameters crucial for optimal stellarator designs.

Despite achieving good performance in predicting some variables, the model faced challenges with variables derived from the second-order near-axis expansion method, as assessed using Huber loss and MAE metrics. Nevertheless, overall, the MDN proved effective as a tool for predicting desired properties of stellarators. Our model can also return the covariance matrix to compute uncertainties associated with each prediction and obtain statistical insight.

Moreover, the creation of a large database of high-quality stellarators facilitated detailed analyses of variable distributions and correlations. These analyses revealed that optimal stellarators tend to cluster within specific ranges of variable space, such as an n_{fp} value around 3 or 4, and a preference for negative values in B_{2c}. The correlation matrix further highlighted strong interdependencies among variables, crucial for accurately predicting input parameters to achieve desired output properties.

As a future work, an ablation study would be crucial to simplify the model, as the increasing complexity of the hidden layer geometry may not be optimal. Adding to this, we intend to integrate the near-axis expansion method directly into the neural network training process, potentially as a differentiable layer. This advancement could leverage techniques like neural network approximations or automatic differentiation tools such as JAX (Bradbury *et al.* 2018). Such enhancements would support the adoption of variational autoencoders, graph neural networks and transformers. Such models could also be extended for future optimizations and designs, integrating them with an ideal MHD model rather than relying solely on a near-axis method. Additionally, a model could be developed to map between the near-axis method and an ideal MHD model. This approach would enable leveraging machine learning models for solving the inverse problem using a near-axis method and subsequently mapping the results to a full ideal MHD optimization.

Acknowledgements

We would like to thank R. Hashmani and M. Padidar for their insightful discussions throughout this work. R.J. would like to acknowledge the support of EUROfusion through an Enabling Research Grant, and the support of FCT – Fundação para a Ciência e Tecnologia, I.P. through project reference 2021.02213.CEECIND/CP1651/CT0004. This material is based upon work supported by the National Science Foundation under grant no. 2409066. This work has been carried out within the framework of the EUROfusion Consortium, funded by the European Union via the Euratom Research and Training Programme (grant agreement no. 101052200 – EUROfusion). Views and opinions expressed are, however, those of the author(s) only and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the European Commission can be held responsible for them. This work used Jetstream2 at Indiana University through allocation PHY240054 from the Advanced Cyberinfrastructure Coordination Ecosystem: Services and Support (ACCESS) program, which is supported by National Science Foundation grants #213859, #2138286, #2138307, #2137603 and #2138296. This research used resources of the National Energy Research Scientific Computing Center, a DOE Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy under contract no. DE-AC02-05CH11231 using NERSC award NERSC DDR-ERCAP0030134. This

research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under contract no. DE-AC05-00OR22725. IPFN activities were supported by FCT - Fundação para a Ciência e Tecnologia, I.P. by project reference UIDB/50010/2020 and DOI identifier 10.54499/UIDB/50010/2020, by project reference UIDP/50010/2020 and DOI identifier 10.54499/UIDP/50010/2020 and by project reference LA/P/0061/2020 and DOI identifier 10.54499/LA/P/0061/2020.

Editor P. Ricci thanks the referees for their advice in evaluating this article.

Supplementary data

Supplementary material is available at https://zenodo.org/records/13623959.

Declaration of interests

The authors report no conflict of interest.

Data availability statement

The data that support the findings of this study are openly available in MLStellaratorDesign at https://github.com/pedrocurvo/MLStellaratorDesign.

REFERENCES

BADER, A., DREVLAK, M., ANDERSON, D.T., FABER, B.J., HEGNA, C.C., LIKIN, K.M., SCHMITT, J.C. & TALMADGE, J.N. 2019 Stellarator equilibria with reactor relevant energetic particle losses. *J. Plasma Phys*. **85** (5), 905850508.

BISHOP, C. 1994 Mixture density networks. *Tech. Rep.* NCRG/94/004. Aston University.

BOOZER, A. 2020 Why carbon dioxide makes stellarators so important. *Nucl. Fusion* **60** (6), 065001.

BOOZER, A.H. 1981 Plasma equilibrium with rational magnetic surfaces. *Phys. Fluids* **24** (11), 1999.

BRADBURY, J., FROSTIG, R., HAWKINS, P., JOHNSON, M.J., LEARY, C., MACLAURIN, D., NECULA, G., PASZKE, A., VANDERPLAS, J., WANDERMAN-MILNE, S., *et al.* 2018 JAX: composable transformations of Python+NumPy programs.

BRIDLE, J. 1990 Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In *Neurocomputing* (ed. F.F. Soulié & J. Hérault), NATO ASI Series, vol. 68.

BUENO, J.I. & KRAGIC, D. 2006 Integration of tracking and adaptive gaussian mixture models for posture recognition. In *ROMAN 2006 - The 15th IEEE International Symposium on Robot and Human Interactive Communication*, pp. 623–628.

CLEVERT, D., UNTERTHINER, T. & HOCHREITER, S. 2016 Fast and accurate deep network learning by exponential linear units (ELUs). arXiv:1511.07289.

COVER, T.M. & THOMAS, J.A. 2012 *Elements of Information Theory*. Wiley.

GARREN, D.A. & BOOZER, A.H. 1991*a* Existence of quasihelically symmetric stellarators. *Phys. Fluids* B **3** (10), 2822.

GARREN, D.A. & BOOZER, A.H. 1991*b* Magnetic field strength of toroidal plasma equilibria. *Phys. Fluids* B **3** (10), 2805.

GLOROT, X. & BENGIO, Y. 2010 Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, PMLR, vol. 9, pp. 249–256.

GOODFELLOW, I., BENGIO, Y. & COURVILLE, A. 2016 *Deep Learning*. MIT Press.

HELANDER, P. 2014 Theory of plasma confinement in non-axisymmetric magnetic fields. *Rep. Prog. Phys*. **77** (8), 087001.

HORNIK, K., STINCHCOMBE, M. & WHITE, H. 1989 Multilayer feedforward networks are universal approximators. *Neural Netw*. **2** (5), 359–366.

Jacobs, R., Jordan, M., Nowlan, S. & Hinton, G. 1991 Adaptive mixtures of local experts. *Neural Comput*. **3** (1), 79–87.

Jorge, R., Sengupta, W. & Landreman, M. 2020 Near-axis expansion of stellarator equilibrium at arbitrary order in the distance to the axis. *J. Plasma Phys*. **86** (1), 905860106.

Kappel, J., Landreman, M. & Malhotra, D. 2024 The magnetic gradient scale length explains why certain plasmas require close external magnetic coils. *Plasma Phys. Control. Fusion* **66** (2), 025018.

Kingma, D.P. & Ba, J. 2017 Adam: a method for stochastic optimization. arXiv:1412.6980.

Landreman, M. 2021 Figures of merit for stellarators near the magnetic axis. *J. Plasma Phys*. **87** (1), 905870112.

Landreman, M. 2022 Mapping the space of quasisymmetric stellarators using optimized near-axis expansion. *J. Plasma Phys*. **88** (6).

Landreman, M. & Jorge, R. 2020 Magnetic well and Mercier stability of stellarators near the magnetic axis. *J. Plasma Phys*. **86** (5), 905860510.

Landreman, M., Medasani, B. & Zhu, C. 2021 Stellarator optimization for good magnetic surfaces at the same time as quasisymmetry. *Phys. Plasmas* **28** (9), 092505.

Landreman, M. & Sengupta, W. 2019 Constructing stellarators with quasisymmetry to high order. *J. Plasma Phys*. **85** (6), 815850601.

Lu, Y. & Lu, J. 2020 A universal approximation theorem of deep neural networks for expressing probability distributions. arXiv:2004.08867.

McLachlan, G.J. & Basford, K.E. 1988 *Mixture Models: Inference and Applications to Clustering*. Marcel Dekker.

McLachlan, G.J. & Peel, D. 2004 *Finite Mixture Models*. Wiley Series in Probability and Statistics, vol. 1. Wiley.

Mercier, C. 1964 Equilibrium and stability of a toroidal magnetohydrodynamic system in the neighbourhood of a magnetic axis. *Nucl. Fusion* **4** (3), 213.

Murphy, K. 2023 *Probabilistic Machine Learning: Advanced Topics*. MIT Press.

Nuhrenberg, J. & Zille, R. 1988 Quasi-helically symmetric toroidal stellarators. *Phys. Lett.* A **129** (2), 113.

Paul, E.J., Bhattacharjee, A., Landreman, M., Alex, D., Velasco, J.L. & Nies, R. 2022 Energetic particle loss mechanisms in reactor-scale equilibria close to quasisymmetry. *Nucl. Fusion* **62** (12), 126054.

Solov'ev, L.S. & Shafranov, V.D. 1970 Plasma confinement in closed magnetic systems. In *Reviews of Plasma Physics* (ed. M.A. Leontovich), vol. 5, pp. 1–247. Springer.

Spitzer, L. 1958 The stellarator concept. *Phys. Fluids* **1** (4), 253.

Uzair, M. & Jamil, N. 2020 Effects of hidden layers on the efficiency of neural networks. In *2020 IEEE 23rd International Multitopic Conference*, pp. 1–6.