Nuclear Inst. and Methods in Physics Research, A 1050 (2023) 168192

Contents lists available at ScienceDirect

Nuclear Inst. and Methods in Physics Research, A

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

Full Length Article

Demonstration of machine learning-enhanced multi-objective optimization of ultrahigh-brightness lattices for 4th-generation synchrotron light sources

Y. Lu [a], S.C. Leemann [a,*], C. Sun [a], M.P. Ehrlichman [a], H. Nishimura [a], M. Venturini [a], T. Hellert [b]

[a] Lawrence Berkeley National Laboratory, University of California, Berkeley, CA 94720, USA
[b] Deutsches Elektronen Synchrotron DESY, Notkestraße 85, 22607, Hamburg, Germany

ARTICLE INFO

Keywords:
Synchrotron light source
Storage ring
Beam dynamics
Lattice design
Multi-objective optimization
Machine learning

ABSTRACT

Fourth-generation storage rings enabled by multi-bend achromat lattices are being inaugurated worldwide and many more are planned for the next decade. These sources deliver stable ultra-high brightness radiation with unmatched levels of transverse coherence by virtue of their highly advanced magnetic lattices. Optimization of these challenging and strongly nonlinear lattices with many degrees of freedom bounded by extensive sets of constraints and multiple often conflicting optimization goals is highly demanding and requires application of the most advanced numerical tools available to the community. While multi-objective genetic algorithms have been very successful in supporting these optimization efforts, the algorithms suffer from a fundamental limitation of their stochastic nature: an exceedingly vast number of candidate lattices, most of which eventually are rejected, has to be fully evaluated. This comes at immense computational cost and thus drives excessive runtime despite use of large supercomputing clusters. We therefore propose to employ deep learning techniques and iterative retraining of neural networks to massively accelerate such lattice evaluation, thereby allowing lattice optimization to rely on far fewer a priori assumptions, open up to larger search ranges, and include right from the start and in parallel multiple error distributions to find truly global optima, all while completing a full optimization campaign in weeks rather than months. In this paper we present the neural network designs, the deep learning approach, iterative retraining procedures, and demonstrate how these machine learning techniques can be incorporated into existing state-of-the-art optimization workflows with only minimal changes applied to the optimization pipeline itself and none at all to the employed tracking codes.

1. Introduction

Storage-ring based synchrotron light sources around the world are presently undergoing a massive transformation. Pioneered in MAX IV [1], the multi-bend achromat (MBA) [2] lattice has ushered in the era of 4th-generation storage rings (4GSRs): a class of ring-based light sources capable of delivering stable ultra-high brightness diffraction-limited synchrotron radiation with a high degree of transverse coherence simultaneously to dozens of beamlines. The MBA lattice—presently foreseen by almost every new source and upgrade project—is composed of many small-aperture magnets with high field gradients capable of providing the strong focusing necessary to achieve ultra-low emittance. This strong focusing reduces the dispersion and drives the natural chromaticity in the lattice. Combined, this calls for very strong sextupoles leading to highly nonlinear lattices exhibiting limited dynamic aperture (DA) and momentum aperture (MA) compared to those of 3rd-generation light sources. Apart from the many engineering difficulties in the design of a 4GSR, the beam physics and lattice optimization itself present a significant challenge due to the large number of magnets that need to be tuned in a multi-variate and multi-objective optimization process. Apart from lattice design expertise, this usually calls for the most advanced numerical and analytical resources available to the community.

Multi-objective genetic algorithms (MOGA) [3] have proven to be one of the most successful and commonly used tools for the optimization of modern light source lattices [4–6]. Multiple variants of MOGA are available, among which the Pareto-based algorithm NSGA-II is the most popular [7,8]. Optimization of an MBA lattice with MOGA is highly non-trivial since ultra-high brightness, lifetime, and injection efficiency are usually in direct competition and a suitable trade-off needs to be carefully established, taking into account an exceedingly large number of constraints. While MOGA is extremely well equipped to undertake such optimization, it suffers from the fundamental limitation that—as a stochastic process—it requires a vast number of candidate lattices to be evaluated. Nonlinear lattice evaluation based on many-turn particle tracking is very CPU-expensive and nevertheless, almost all evaluated lattices are eventually rejected by MOGA. This weakness,

* Corresponding author.
E-mail address: SCLeemann@lbl.gov (S.C. Leemann).

https://doi.org/10.1016/j.nima.2023.168192
Received 7 December 2022; Received in revised form 21 February 2023; Accepted 6 March 2023
Available online 8 March 2023

inherent to MOGA optimization, usually calls for the lattice designer to either a priori limit their search ranges or to make initial assumptions about the constitution of the volume in search space leading to promising solutions. This, however, competes directly with the desire to enlarge and continuously re-evaluate search space in order to maintain sample diversity so as not to get trapped in local minima. And even once a MOGA optimization process has led to attractive solutions, the robustness of such solutions under perturbations—such as realistically to be expected manufacturing, calibration, and alignment errors in the actual as-built machine—have to be considered. Often times it would be desirable to design right from the start for a lattice perturbed by an at least minimal error set, but this then would call for many such error distributions to be simulated since most such errors follow a stochastic distribution and the exact final error set remains unknown until the machine has been built and commissioned. However, optimizing right from the start for a multitude of error distributions is presently not feasible due to the vast CPU requirements for MOGA optimization of just a single lattice. We have set out to change this.

Attempts have been made to accelerate the MOGA process by using proxies for nonlinear properties (specific tune shifts, DA evaluation for certain energies, etc., e.g. [9]) instead of computing exact solutions through many-turn tracking. Others have attempted to optimize nonlinear lattice properties solely based on tuning nonlinear elements in the lattice, but without freedom to also adjust linear optics within certain boundaries (e.g. [10]). Here we shall instead prefer to optimize directly the relevant physics quantities by tuning linear *and* nonlinear magnet families concurrently. We propose to achieve the massive acceleration in MOGA required to do such optimization through Machine Learning (ML) techniques involving primarily deep learning and iterative retraining. We will demonstrate how we use deep neural networks (DNNs) to replace computationally expensive many-turn tracking, thereby allowing an individual MOGA optimization to complete within hours rather than weeks. This in turn means that parameter search ranges can be opened up, fewer initial assumptions made and, most importantly, that optimizations can be carried out in parallel for many different error distributions hence ensuring that the final solution corresponds indeed to a global optimum for all expected conditions of the as-built machine.

The application of ML to storage ring lattice optimization is not new. There have been a couple attempts in the past, e.g. [9–12]. However, we believe they are not entirely suitable to global optimization of 4GSR lattices—where we optimize linear and nonlinear lattice properties concurrently (i.e. search space consists of all quadrupoles and sextupoles/octupoles in the lattice)—as we attempt to outline below.

- While we employ ML to replace tracking itself, we do not attempt to apply it to recognize patterns in the population or regions of the Pareto-optimal front as proposed in [11]. In order to keep this ML-enhanced optimization as generic as possible, we prefer not to assume that such clusters must exist in either solution or input space. However, since our ML-enhanced optimization process is intended to augment a conventional MOGA optimization, there is no reason ML-enhanced MOGA cannot be combined with pattern recognition as described in [11] when warranted due to the underlying physics. In fact, for the method proposed here, adding an additional step as suggested in [11] could be used to increase the density of elite candidates thereby leading to a better distribution of candidates along the Pareto-optimal front.
- Unlike the process described in [12], we are not so much interested in enlarging the MOGA population size with ML in order to increase diversity. Throughout our optimization campaigns, we do not see population diversity as a substantial issue as long as we use sufficient samples per generation, we make sure to train our DNN models on sufficiently large population sets, and we then retrain as necessary. While [12] limits itself to training on small numbers of samples (100× the number of input variables is suggested), we choose a substantially larger number in order to ensure that DNN surrogate models are highly accurate. Naturally, this approach comes with a CPU penalty since generation of training data requires tracking which is computationally expensive. On the other hand, accurate surrogate predictions allow our ML-enhanced optimization to converge very quickly with a large number of samples per generation (unlike the assumptions made in [10]), thereby ensuring that population diversity does not become a restriction.
- Similar to the approach suggested in [9], we also propose to use deep learning and retraining to generate surrogate models that are used to speed up computationally expensive particle tracking in massively parallelized MOGA optimization. We also employ continuous retraining to ensure that surrogate models are steadily improved during the optimization process thereby constantly improving solution quality. However, unlike [9], we suggest to optimize directly on relevant physics quantities, i.e. instead of relying on tune footprint or the distance to certain resonances, we prefer to optimize directly the DA; and instead of relying on the DA for two specific momentum deviations, we prefer to optimize directly on MA. While this is computationally more expensive, we consider it more reliable for lattice optimization: separation from certain resonances or small tune shifts by themselves do not guarantee large DA, whereas large DA is absolutely required in order to ensure sufficient injection efficiency and lifetime. We will show that by employing ML-enhanced optimization, the achieved speedup over conventional many-turn tracking is several orders or magnitude and therefore more than makes up for the added computational effort of direct DA/MA optimization. We believe a further advantage of our proposed method is that it does not require changes to the tracking code. No additional properties to serve as optimization proxies need to be calculated. Our approach will modify the MOGA code in only a single instance: calls to the tracking code to calculate DA and MA will be replaced by a simple look-up call to the surrogate model.

The remainder of this paper is structured as follows: the following section summarizes the classical MOGA approach using the ALS-U lattice optimization as a typical example for a 4GSR. Then, Section 3 will introduce the ML modeling approach designed to replace many-turn particle tracking used by MOGA. Section 4 describes in detail how an ML-based MOGA workflow is constructed, relying on deep learning and iterative retraining, and how it is kept efficient by close monitoring of convergence. This is followed by Section 5 where results are shown when this full ML-based MOGA workflow is applied to a 4GSR lattice optimization, again using the ALS-U lattice as a typical 4GSR example, thereby allowing for direct comparison with the conventional MOGA approach. Finally, the paper concludes with a brief summary and outlook.

2. Conventional multi-objective design of 4th-generation storage ring lattices

In order to transform to a 4th-generation synchrotron source, the Advanced Light Source (ALS) at Lawrence Berkeley National Laboratory is upgrading the existing triple-bend achromat (TBA) storage ring lattice to a nine-bend achromat (9BA) lattice. The upgraded storage ring will be constrained to within the current ALS footprint, having the same 12-fold period and nearly the same circumference. The magnet layout for one sector is shown in Fig. 1. In order to meet requirements for ≈ 100 pm rad emittance and due to severe space constraints, a pseudo-hybrid[1] type 9BA lattice with two dispersion bumps on both ends of the central arc section is adopted for the ALS upgrade. It consists of 9 combined-function dipoles (B1-B9), 6 focusing quadrupole (QF1–QF6)

[1] The phase advance between the dispersion bumps does not constitute the $-I$ transformer of the conventional hybrid MBA.

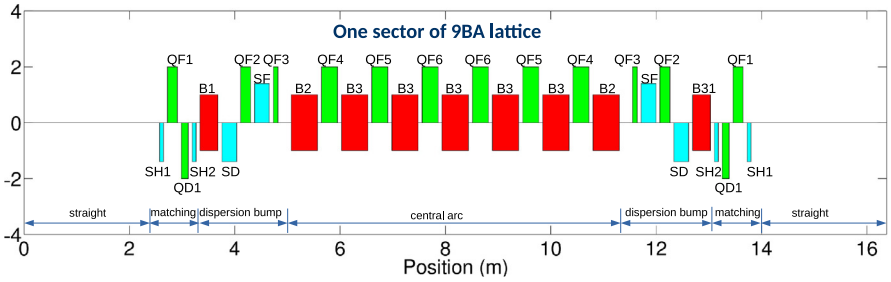

Fig. 1. Magnet layout of one sector of the ALS-U lattice.

families, 1 defocusing quadrupole (QD1) family, 2 chromatic sextupoles (SF and SD) families inside dispersion bumps for chromaticity correction, and 2 harmonic sextupole (SH1 and SH2) families in the straight section for nonlinear dynamics optimization. In the early design of the lattice, not only magnet strengths but also their length and positions were adjusted to optimize the lattice properties and to accommodate space and engineering requirements.

After this initial design phase results in a preliminary design, the magnet layout is fixed while only the magnet strengths are tuned. There are a total of 11 free tuning knobs[2]: 9 quadrupole gradients (QF1–QF6, QD1, B1 and B2/B3) plus 2 harmonic sextupole strengths (SH1 and SH2). The two chromatic sextupoles (SF and SD) are tuned for chromaticity correction to +1 in both planes. The lattice optimization has to balance competing objectives while satisfying a number of constraints posed by technological limitations and physics requirements. The most basic trade-off is the one between the objectives of attaining small emittance vs. sufficient DA and MA. Optimization of such a multi-objective, multi-variable and highly constrained problem is nontrivial. MOGA is well suitable for this kind of optimization problem. The MOGA concept was introduced to the accelerator community about two decades ago [3–6,14,15] and is now widely used to design and optimize all kinds of accelerator systems.

In our initial approach to optimize the ALS-U lattices, the resonance driving terms, amplitude dependent tune shift (ADTS) and chromatic tune spread, which do not involve intensive particle tracking, were minimized to improve lattice properties. However, this approach did not by itself render entirely satisfactory results, especially for nonlinear properties of the lattice such as DA and MA. To this end, direct evaluations of nonlinear properties by tracking are adopted for lattice optimizations. MOGA has been extensively used for the optimization. For ALS-U studies, we have relied on MOGA based on NSGA-II [7,8] where Tracy [16] is used to calculate beam properties and perform many-turn particle tracking for DA and MA calculations. With MOGA, ideally the initial lattice population should be uniformly and randomly populated over a wide subset of the accessible parameter space. Unfortunately, because the evaluation of the nonlinear properties of these lattices is so time consuming, this tends to lead to a very slow convergence to the point of impracticality. We found that a multi-stage approach, as briefly described below, can considerably speed up the process.

2.1. Conventional MOGA optimization for ALS-U

First, we carry out the optimization for the linear properties alone, by only varying the 9 quadruple gradients. The purpose of this step is to narrow down the search ranges of the quadrupole gradients, thus excluding parameters that lead to non-physical (unstable) solutions or violate our linear property targets. Because the nonlinear properties are not evaluated, this stage is very fast. The optimization objectives are the natural emittance and beta functions in the straight-section midpoints, which are directly related to the brightness of the machine, the ultimate goal of this lattice optimization. We also set a 150 pm rad upper-limit cut-off for the natural emittance and reject lattice solutions with straight section beta functions larger than 3 m or less than 1 m. Horizontal and vertical tunes are forced to be nearly identical in anticipation of operation at coupling resonance. Instead of letting the optimization run its full course, we monitor the evolution of the lattice population and stop the run when we determine that the emittance and beta functions spread over a sufficiently small, but not too narrow range. Then, the last generation is selected as an initial population for the next stage: linear and nonlinear lattice optimization.

Table 1
Constraints for ALS-U lattice optimization.

Natural emittance	$\varepsilon_0 < 155$ pm rad
Maximum beta	$\beta_{x,y} < 30$ m
Maximum dispersion	$\eta_x < 15$ cm
Fractional tunes	$0.1 < \nu_{x,y} < 0.4$
Dispersion at center of straight	$\lvert\eta^*_x\rvert < 1$ mm
Beta at center of straight	$1\,\text{m} < \beta^*_{x,y} < 5\,\text{m}$
Beta in central arc bends (B3)	$\beta^{B3}_{x,y} < 4$ m
Fractional tune difference	$\lvert\nu_x - \nu_y\rvert < 0.01$
Chromatic sextupole strength (SF, SD)	$b_2 < 900\,\text{m}^{-3}$

In this 2nd stage, both linear and nonlinear properties of the lattice are optimized simultaneously. The linear property objectives are the same as before, except that beta functions are no longer optimized but rather constrained. A full list of the applied constraints is given in Table 1.

The nonlinear properties to be optimized are DA and MA which are related to machine performance through injection efficiency and Touschek lifetime. We do not directly optimize the injection efficiency since its evaluation is very time consuming and depends on the exact injection method. It also strongly depends on the exact longitudinal phase space which in turn can be heavily affected by harmonic cavities; at such an early stage in the design process we prefer not to make assumptions about such systems yet. In practice, DA can be evaluated either by 6D tracking to estimate DA area or by 4D tracking using frequency map techniques to estimate the total diffusion rate [17–19]. The latter method is used in our optimization since it has been observed to render superior lattice performance.[3] The evaluation of Touschek lifetime requires MA evaluations along the machine which is extremely time consuming [20,21]. Instead, averaged MA at select points along one sector is used as a proxy for Touschek lifetime. In this 2nd optimization stage the tuning knobs consist of all 9 quadrupole gradients plus 2 harmonic sextupole strengths. The chromatic sextupoles are tuned by fitting chromaticity to +1 in both planes during the optimization. The

[2] The final ALS-U design also includes reverse bending [13] in the arc quadrupoles which adds another three free parameters as well as high-field bend magnets with an additional two free parameters. For the purpose of this study, we will, however, ignore reverse bending and high-field bends which reflects the state of the ALS-U design efforts around 2018.

[3] A smaller but contiguous area of low diffusion is preferred over a larger DA that contains many areas of elevated diffusion (indicating onset of chaotic motion) [17–19].

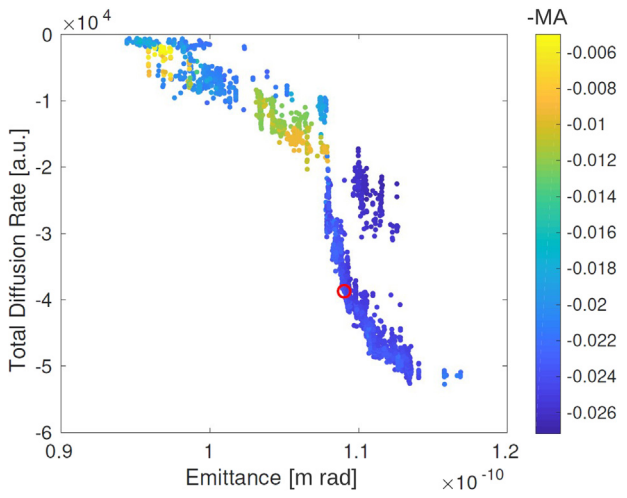

Fig. 2. Pareto front resulting from 2nd stage optimization of linear and nonlinear degrees of freedom in the ALS-U lattice (negative MA is shown here since optimization applies a minimizer). Large negative diffusion rates are desirable. The red circle represents the solution that was chosen in order to prioritize Touschek lifetime [23].

same constraints as used in the linear lattice optimization are again applied here in the 2nd stage.

Both MA and DA are evaluated by including random linear gradient and skew errors in the lattices that simulate typical residual beta beating (2%–3%) and linear coupling (about 1%), as they are commonly determined in real machines after carrying out lattice calibration and correction using orbit-response matrix analysis. Specifically, the relative normal gradient errors with a sigma of 2×10^{-4} and skew gradient error of 5×10^{-4} are applied to all quadrupoles and combined-function dipoles. A Gaussian distribution with 2-sigma truncation is applied when the gradient and skew errors are populated. These error distributions are retained for the entire optimization stage and only upon its completion, with a candidate lattice in hand, alternate error distributions are applied and it is verified that these alternate errors in the chosen candidate still render comparable performance to the originally optimized lattice.

The initial population for this 2nd stage optimization is taken from the final generation of the previous linear optimization stage along with random sextupole gradients initially supplied to the first generation. The behavior and convergence of MOGA can be greatly affected by the hyperparameters of the algorithm such as probabilities and index of mutation and crossover, which determine how much the parent and child generations differ from each other and how frequent they should be mutated and crossed over. We found that, for best results, different tuning of these hyperparameters are more appropriate at different stages of the lattice population evolution. Therefore, the optimizations are broken down into several independent runs, where the population generated at the end of one run is used as the initial population for the next, and the hyperparameters are re-tuned after each run. In the earlier runs we set higher mutation and crossover probabilities in order to encourage the exploration over wider ranges; later on, lower probabilities are effective to boost convergence speed. Each run typically spawns 200 generations. With a typical population size of 5000, it usually takes about 2–3 days to complete a single run with 1000 computing cores on the ALSACC cluster, which is hosted by the LBNL Supercluster and has a mixture of different CPU architectures and memory configurations [22]. Usually, several runs are required to achieve a fully converged Pareto front. Therefore, this whole optimization process for the 2nd stage typically takes about a week or two.

A typical Pareto front resulting from this 2nd stage of lattice optimization is shown in Fig. 2. It indicates a clear trade-off between emittance, DA, and MA. Once a Pareto front has been found, lattice solutions meeting design requirements are identified and further evaluations are carried out to validate the results. In later stages, the ALS-U lattice search is then expanded to include reverse bending and high-field bends in three sectors (along with dedicated local QD families) to further enhance overall lattice performance. But it is this 2nd stage of optimization, combining linear and nonlinear lattice properties, that the remainder of this article will focus on. Once the ML-enhanced optimization is established, integrating ML into later stages of the lattice optimization, possibly including additional degrees of freedom (DoF), becomes a natural extension as the following sections will show.

3. ML-based modeling approach to enhance multi-objective design

Machine Learning has gained popularity in a wide range of fields, not just computer science. It has proven the ability to solve many complex problems that traditional methods cannot handle [24]. Several ML-based methods have been proposed to improve multi-objective lattice design [9,11,12]. However, as discussed in the Introduction, we recognize a serious bottleneck of MOGA lies in the many-turn tracking evaluation of the nonlinear properties of lattices which in our case consist of DA and MA. Thus, we introduce a neural network (NN) to serve as the surrogate model for DA and MA prediction in order to massively accelerate MOGA thereby rendering an ML-based MOGA. Ideally, the goal is to reduce tracking time for DA and MA calculation from minutes per child on one core (resulting in many weeks for the full campaign) to milliseconds for a simple NN lookup (NN prediction).

3.1. A first 2-dimensional ML-based nonlinear optimization

Early studies based on optimizing DA and MA for only two inputs (2 harmonic sextupoles) [25] indicated that each could be modeled well by its own NN, starting with only two fully-connected hidden layers and later expanding to seven which rendered excellent prediction quality with minimal training effort. For the two-dimensional problem, our initial efforts on generating training data focused on random sampling of parameter space as well as equidistant sampling whereby the sampling density was reduced until prediction errors were observed to start to grow. This resulted in a situation where a NN for DA or MA prediction required only ≈13,000 samples (115 samples for each input) for training, compared to ≈250,000 samples required for a comparable conventional MOGA approach. Once the tracking results for the sampled points in input space had been generated, the training effort itself was as little as minutes on a conventional desktop CPU and resulted in DA and MA predictions to within 1%–2% rms of typical Pareto-optimal results. Because the training effort is so short compared to the CPU time required for the direct MOGA optimization—even in just two dimensions—this resulted in a direct speedup of roughly two orders of magnitude.[4] For this simple 2-dimensional input space, the equidistant sampling of input space provided consistently lower prediction errors from the resulting NNs compared to using MOGA data for training.

3.2. Full 11-dimensional ML-based models

However, once the full 11-dimensional problem including all quadrupoles needs to be modeled, generating appropriate training data becomes much more difficult. While the 9 additional DoF are introduced to the DNN simply by adding 9 fully connected inputs to the first hidden layer, training data can no longer be generated by equidistant sampling of the entire 11-dimensional input hypervolume. Ideally, sampling should instead be guided by a "wise choice", but since the

[4] A direct speedup by a factor 650 was demonstrated [25] after further improvements had been made to training data collection.

approach presented here is designed to accommodate arbitrary lattice optimization, we do not want to introduce too much physics-based intuition. Instead, we resort to a user-independent method wherein initial MOGA optimization lends itself to guiding which parts of the input hypervolume need to be sampled: we employ already available data in the form of the first few generations of conventional tracking-based MOGA as our initial training data. The following section will include further detail on the exact generation and selection of appropriate training data.

For the full 11-dimensional problem of the 2nd stage lattice optimization, an 8-layer fully-connected NN architecture with ReLU as the activation function [26] was chosen resulting from extensive numerical studies, as was the width of the individual layers (for the DNN models used herein we have arrived at: 11, 32, 64, 128, 256, 128, 64, 32, 1). An input size of 11 corresponds to the 11 DoF. For our ML-based MOGA (which we will refer to as ML-MOGA), we use two DNNs implemented in PyTorch, one to model DA and one to model MA, as a function of the 11 inputs. These DNN models for DA and MA will replace the tracking routines used by Tracy for DA and MA evaluation in the original tracking-based MOGA (which we will refer to as Tr-MOGA). The rest of the Tracy code and the MOGA optimizer shall remain unchanged.

As a result of thorough hyperparameter tuning studies, we set batch size to 128 and learning rate to 0.001 with decay rate 0.8 during the training process. The Adam optimizer [27] is applied with 500 epochs to train the DNN models. We split the sample data into 80% for training and 20% for test. Typical training data sizes are around 50,000 samples (for details on training data preparation, cf. Section 4). No regularization such as dropout or early stopping was required, however, automatic learning rate reduction (ReduceLROnPlateau [28]) was exploited to improve training results. Several attempts were made to inspect for possible overfitting, using various data sets and models, however, no overfitting could be detected. In the context of lattice optimization, the training data is in fact rather limited in size (unlike in image processing) due to the stated goal of performing as little tracking as possible. The quick training process (it takes only several minutes to train a model using our DNN architecture on a common desktop CPU) in conjunction with our choice of 500 epochs for training resulted in entirely satisfactory training performance. Here we employ mean absolute error (MAE) as the loss rate, i.e. a measure for loss between each element of prediction and target. The aforementioned hyperparameter choices all result from hyperparameter studies aimed at minimizing this loss. Once we have the trained models, the prediction process (a DNN look-up) takes only several ms per child. With typical DA/MA evaluation by tracking for a single child requiring on the order of minutes, there thus exists the potential for a reduction of computational effort by *orders of magnitude* compared to conventional Tr-MOGA.[5] It is this vast speedup here that opens up the potential for evaluating huge population sizes once ML-MOGA is employed, thereby ensuring that the multi-objective lattice optimization does not risk missing good solutions or taking overly long to converge towards best solutions.

To better evaluate our trained models, we present two figures: Fig. 3 shows a comparison based on test data between actual DA results from tracking and DA as predicted by the DNN surrogate model. Almost all data lies on the diagonal with no extreme outliers. Fig. 4 shows the distribution of DA prediction error based on test data. The error distribution shows no systematic offset between prediction and ground truth. For this distribution, we calculate the root mean square error (RMSE) which is the rms difference between prediction and ground truth for all data in the test population. The goal of all good training is to firstly, minimize this error for the training population once training has converged, and secondly, to show that the RMSE also remains low

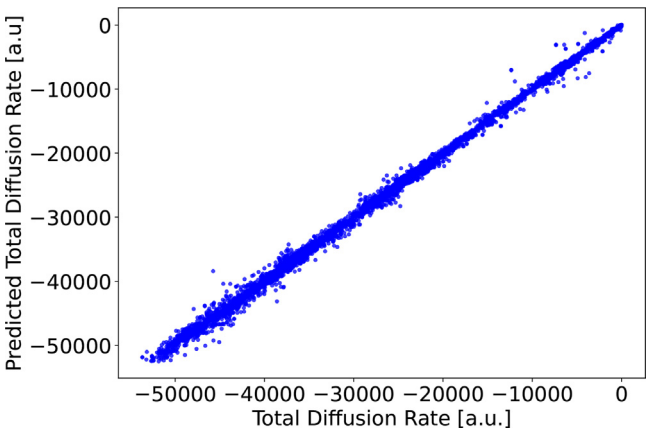

Fig. 3. Comparison between actual DA from tracking and DNN prediction of DA based on test data.

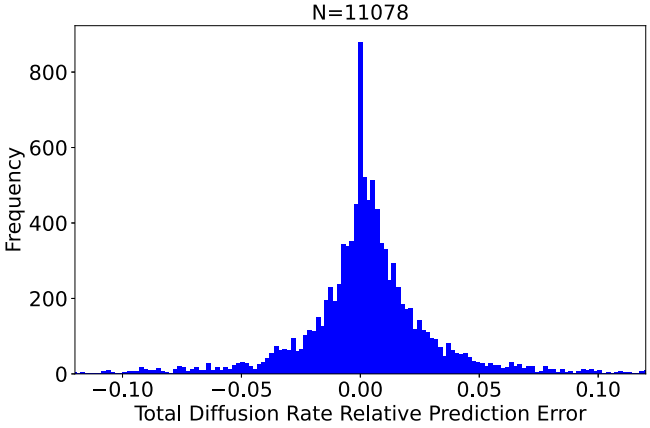

Fig. 4. Histogram of relative prediction error of the DNN surrogate model for DA compared to the ground truth for test data (11,078 samples).

when calculated for the test data set. In the example shown here, the RMSE is 459. To better illustrate the meaning of this error, it is helpful to introduce the relative prediction error (RPE), in which an RMSE is normalized by the absolute value of a near-optimal value among the entire population.[6] In this case the best observed DA is around -5.4×10^4 so that the RPE becomes 0.9%.[7] These figures show that this DNN surrogate model for DA performs extremely well for this test data. The DNN model for MA is assessed in the same manner and shows similar prediction performance.

4. Constructing an ML-based multi-objective design workflow for 4th-gen storage ring lattices

As described in Section 2.1, the first lattice optimization stage is based on narrowing down the search ranges of the linear input parameters and is identical for both ML-MOGA and Tr-MOGA. Since linear

[5] The CPU time per child is dominated by the time required for DA and MA tracking. Calculation of linear properties from one-turn maps is, by comparison, instantaneous and therefore presents only minimal potential for improvement.

[6] Normalizing with the best values of the population rather than, for example, using the average value at a certain generation leads to less change due to increasing generation number alone. Ideally, changes in the RPE reflect changes in prediction quality, not just population movement in objective space as the optimization run progresses.

[7] Throughout our studies, we have observed that DNN models perform well as surrogates for DA and MA when RPEs remain on the order of 2% or less. Larger values can be accepted in order to reduce the amount of data generation required for training (since actual training time is typically very short compared to the time required to generate training data). On the other hand, requiring smaller PREs shows only little potential benefit since data generation for retraining is usually quick compared to the effort to generate the initial training data.

optics are optimized here first (and only linear optics constraints have to be satisfied), calculations in this first stage are quick to calculate from one-turn maps and do not require multi-turn tracking. We use the final generation from this stage as input for the second stage MOGA runs, in which linear and nonlinear properties of the lattice are optimized simultaneously. For Tr-MOGA, the run then continues until it converges.

For ML-MOGA, however, we first need to prepare sample data to train the models. As mentioned above, with 11 DoF, it becomes impossible to evenly and adequately sample the input hypervolume in order to generate the required training data due to massive computational cost.[8] Thus we instead use the first 10 generations from the second stage Tr-MOGA to train DNN models for use in ML-MOGA and use the 10th generation as input for the initial ML-MOGA run. Studies have shown that including the entire population of the first n generations of MOGA in the training data pool typically leads to poor accuracy of the DNN predictions. The reason for this is that on the order of half of the training data acquired in this way contains solutions that violate constraints (cf. Table 1), either simple constraints on allowable optics (e.g. maximum allowable beta function in the ID source points) or unphysical solutions (e.g. unstable solutions where the one-turn map shows $|\mathrm{Tr}(\mathcal{M})| > 2$). However, once we remove such candidates from the training pool, the resulting training converges faster and leads to DNNs with lower prediction errors. When DNN models are later retrained using tracking results from candidates in later phases of MOGA (cf. below), there are usually only very few constraint violations so that hardly any samples have to be removed before retraining. The aforementioned choice of $n = 10$ (and including rejection of roughly half of the samples due to violated constraints) is of course somewhat specific to the optimization problem and requires careful analysis. Studies have shown that n should be chosen such that the resulting RPEs remain below ≈2%. For the ALS-U optimization case, $n = 10$ represents a good compromise between runtime and sufficiently accurate DNN predictions for DA and MA. Note, the required tracking effort for $n = 10$ compares very favorably to the almost 650 generations required for traditional Tr-MOGA to converge in this 2nd stage.

4.1. First application of ML to MOGA

With training data extracted from the first 10 generations of conventional Tr-MOGA, DNN surrogate models for DA and MA are rendered showing RPEs on the order of 1%–2%. Equipped with such high prediction accuracy, we convert these DNN models with TorchScript via tracing and serializing script modules [29] so that they can then be used by MOGA in lieu of the Tracy DA and MA tracking routines (effectively translating the DNN from a training environment in Python to C++ for direct use in MOGA). The CPU time required for each child, which in conventional MOGA is dominated by the effort to perform tracking for DA and MA calculation, is drastically sped up due to the near-instantaneous DNN lookup. Equipped in this manner, the ML-MOGA optimization can be run until it converges. However, one drawback of such an approach is that the DNN surrogate model is based on just the initial MOGA generations, which cover only a limited region in input parameter space and most certainly, unless in the case of a very fortunate coincidence, not the part of input space that leads to Pareto-optimal solutions. These DNN models have no prior knowledge of the rest of parameter space and as the optimizer converges towards Pareto-optimal solutions, prediction errors will grow if the relevant input parameter space cannot be re-sampled in the vicinity. While DNNs have shown excellent performance in terms of interpolation in quasi-smooth distributions, they cannot be expected to extrapolate well beyond the input space they were trained on, and most certainly they will fail at extrapolation when the input subvolume associated with Pareto-optimal solutions is disjoint and/or fractal in nature. As an illustration of this problem, Fig. 5 shows the outcome of a first ML-MOGA run for the ALS-U optimization after it has converged (at generation 732), in comparison to the corresponding conventional Tr-MOGA run (converged at generation 643) based on the same initial population and input variable constraints. Despite both runs showing clear progress from the initial population towards optimal solutions, and despite final generations showing non-dominated solutions only (rank = 1), ML-MOGA in its converged state does not match the results based on tracking.

An additional problem stems from the fact that despite very low RPEs, as the optimization approaches the Pareto-optimal front, very small differences in output (by e.g. prediction error) can lead to substantial changes in the ranking MOGA applies to all children within a generation, thereby affecting the breading of future generations, and thus convergence towards the Pareto-optimal front. In order to alleviate this deficiency, we introduce a single tracking step once ML-MOGA appears to have converged (cf. next Section). This tracking step delivers accurate values for DA and MA of the final ML-MOGA generation and thus it serves as a *validation step*. This validation step therefore comes at the runtime expense of one additional generation of tracking. Once the validation results have been computed, each sample's rank needs to be re-evaluated, since exact solutions will no longer all be non-dominated. An example for this is shown in Fig. 6 where all rank-1 solutions of the aforementioned final ML-MOGA generation are displayed.

4.2. Retraining & iterative applications of ML to MOGA

Despite solutions in the validated final ML-MOGA population (cf. Fig. 6) corresponding to those of the conventional Tr-MOGA results (cf. Fig. 5, left), there are regions of input space that are not covered by ML-MOGA and hence its objective space distribution does not extend all the way to (and match) the Pareto-optimal front derived by Tr-MOGA.

[8] In the original 2-dimensional nonlinear optimization example mentioned in Section 3, each input dimension was evenly sampled in 115 steps. For the 11-dimensional problem, a similar sampling would result in almost 5×10^{22} grid points that would require evaluation by tracking in order to generate a first batch of training data. If, on the other hand, training data were restricted to 100,000 samples to be tracked, this would allow for less than 3 samples per input.

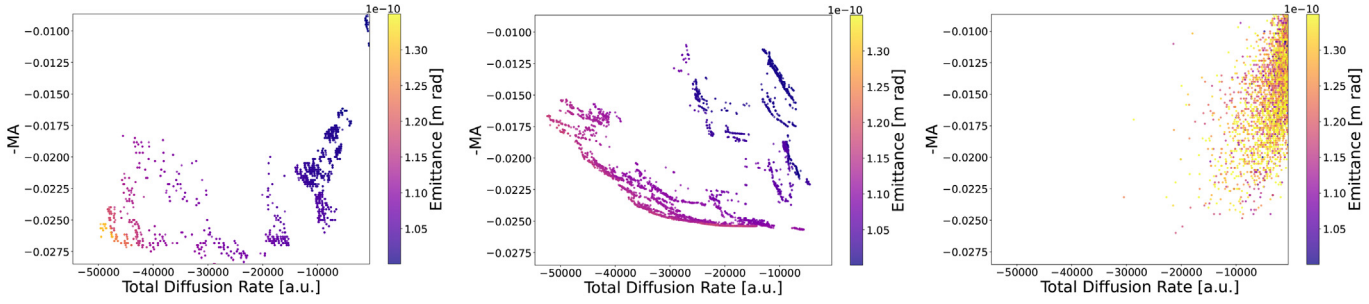

Fig. 5. Comparison in objective space between conventional Tr-MOGA run (left) and converged initial run of ML-MOGA with DNNs trained on first 10 generations of the Tr-MOGA run (middle). Note that both MOGA runs were seeded with the same initial population (right). Negative MA is displayed here since optimization relies on a minimizer.

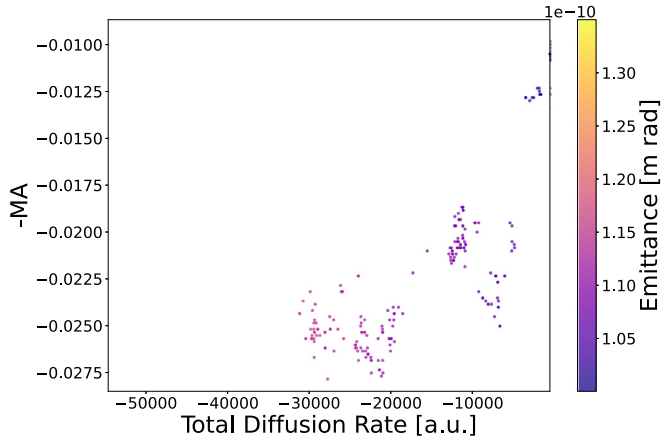

Fig. 6. Objective space plot for the validated rank-1 solutions of the final generation of the converged initial run of ML-MOGA. Input variables related to this distribution are identical to those of Fig. 5 (middle).

Final ML-MOGA input distributions are close to the final Tr-MOGA input distributions, but they are not yet identical. This does not really come as a surprise, however, considering that the data used for training the DNNs employed by ML-MOGA was derived from only the earliest generations of the optimization, which did not yet include those areas leading to the best solutions, let alone with dense sampling. In order to overcome this limitation, we set out to retrain our DNN surrogate models by combining the original training data with the data acquired from tracking during the validation step.[9] This allows the models to be updated with solutions approaching the Pareto-optimal front, thereby giving them better predictive capabilities in the vicinity of the areas of parameter space we are most interested in, while at the same time coming at near-zero computational cost. The expense of tracking a single generation has already been accrued, so that the only additional CPU time consists of the retraining of the two DNN models. And since training of the DNN models is very quick compared to the effort to perform tracking for one generation, this presents very little overhead.

Once the DNN surrogate models have been retrained on one additional generation of input data, they can again be inserted into an ML-MOGA run which takes the final generation of the first run as seed and optimizes it until it again converges. At this point, a validation step can be repeated and the DNNs can be retrained again. We recognize this allows for an iterative process until, ideally, the overall ML-MOGA–validation–retraining process converges, resulting in a generation that, when validated with tracking, confirms the Pareto-optimal front has been reached. The details of this iterative process shall be the topic of Section 5.

4.3. Convergence and distance metrics

Crucial to designing such an iterative workflow is the definition of distance metrics in input and output space that can be used to assess convergence of ML-MOGA, and progress made from retraining the DNNs and launching an additional ML-MOGA run. Since in regular use, no Tr-MOGA campaign already exists to define what the true Pareto-optimal front is, we need to rely on a process using ML-MOGA data alone to determine when the overall iterative process has asymptotically reached a stable final state. It is important here to note that since every iteration requires retraining of the DNN surrogate models and that in turn requires a full generation of inputs has to be tracked at significant computational expense, it is of utmost importance to determine how many such iterations are actually necessary and at which point retraining no longer leads to a significant improvement.

As such distance metrics, we introduce two Euclidean norms, one each for input and output space, respectively. The first metric, $\delta_i(m)$ describes for generation m how far inputs have been shifted through input hypervolume since generation $m-1$. Since for each variable we normalize this movement with the overall range of this variable in its dimension of input space, the relative distances can be summed resulting in one overall relative measure of movement in input hypervolume. Eq. (1) shows this distance metric for input variables.

$$\delta_i(m) = \frac{1}{n(n-1)} \sum_{j=1}^{n} \sum_{k=1}^{n} \sqrt{\sum_{l=1}^{N} \left(\frac{a_{jl}^{(m)} - a_{kl}^{(m-1)}}{c_l} \right)^2}. \tag{1}$$

Here, we assume there are n children per generation, N dimensions in input space, and $a_{jl}^{(m)}$ is input variable l of child j in generation m. Similarly, $a_{kl}^{(m-1)}$ is input variable l of child k from previous generation $m-1$. Finally, c_l is the parameter range of variable l from the first MOGA stage as mentioned above.

For the distance metric in output space (objective space), we again wish to employ a unit-free metric that contains all objectives in one scalar norm. For this purpose, we introduce a "golden target" which presents a quasi-optimal solution for each objective by itself as if it were unconstrained by other competing objectives. This choice leaves some freedom to the lattice designer: studies have shown that an aggressive but not entirely unrealistic choice of these values is most beneficial. For the case of the so far discussed ALS-U lattice optimization, we have arrived at a target emittance $\varepsilon_0 = 90$ pm rad, target MA of $M_0 = 3\%$, and target DA $D_0 = -60,000$. We then define the distance metric for output space in generation m as $\delta_o(m)$ such that we record the movement of the population at generation m towards this golden target. Since the movement in each dimension of objective space can be normalized by the golden targets serving as a *reference value*,[10] this metric can be applied equally to both Tr-MOGA and ML-MOGA. In fact, it provides for a direct comparison as long as the reference values remain unchanged. Changing the reference values does not by itself affect determination of convergence, however, since for this purpose, the change of $\delta_{i,o}(m)$ is observed, not its absolute value (cf. below). Eq. (2) shows the distance metric for the objective variables emittance, DA, and MA compared to the chosen reference values $\{\varepsilon_0, D_0, M_0\}$ presenting ideal target values.

$$\delta_o(m) = \frac{1}{n} \sqrt{\sum_{j=1}^{n} \left[\left(\frac{\varepsilon_{mj} - \varepsilon_0}{\varepsilon_0} \right)^2 + \left(\frac{D_{mj} - D_0}{D_0} \right)^2 + \left(\frac{M_{mj} - M_0}{M_0} \right)^2 \right]} \tag{2}$$

Here D_{mj} is the DA of child j in generation m and similarly for MA, M_{mj}, and emittance, ε_{mj}.

While absolute values of $\delta_{i,o}$ as a function of m present limited value, we can consider

$$\Delta\delta_{i,o}(m) \longrightarrow 0, \quad \text{for } m \gg 1 \tag{3}$$

an indication of convergence of the MOGA run. When the derivative of these metrics approaches zero, we know that firstly, the input distributions are no longer being substantially shifted from one generation to the next and secondly (or consequentially), that distributions in objective space are no longer being further advanced in direction of the golden target values by adding additional generations. Calculation

[9] This is similar to what was proposed in [30], but note that here, we are dealing with a class of problems characterized by high dimensionality of input space (due to the global and concurrent optimization of the linear and nonlinear lattice) as well as the resulting discontinuities in the input subspace corresponding to optimized solutions, and hence, we cannot anticipate smooth or evenly populated Pareto-optimal fronts. Instead, we will pick up solutions distributed throughout the entire Pareto-optimal front for retraining.

[10] This reference value is similar to the $\gamma^\star$ suggested in [12], but here we rely on a golden target instead of best non-dominated front, and we normalize in each objective dimension such that the metric becomes useful for comparisons across multiple generations or among different optimization algorithms.

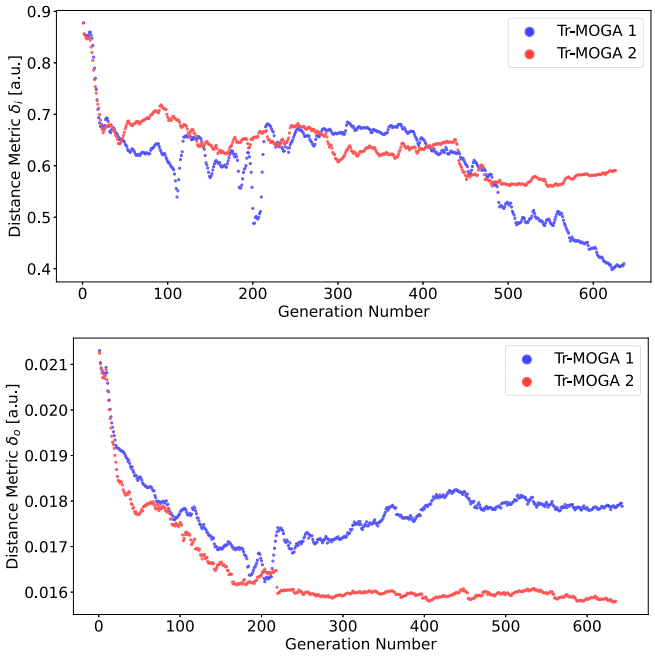

Fig. 7. Distance metric $\delta_i(m)$ for input variables (top) and $\delta_o(m)$ for objective space (bottom) for two Tr-MOGA runs with different random seeds.

of these metrics is very quick and can be carried out immediately once a generation has completed in MOGA. In this manner, we can define a metric derivative threshold at which MOGA can be terminated since it is considered converged. Obviously, input and output metrics both need to stabilize individually in order to consider the overall MOGA run converged. Furthermore, this threshold definition needs to account for the numerical noise common to such optimization processes (cf. examples below).

Once both metrics have stabilized to sufficient degree, we can consider the MOGA run converged. Fig. 7 shows an example for distance metrics for input and objective space for two Tr-MOGA runs with different MOGA random seeds.[11] By comparing these metrics for the two cases, we can conclude that Tr-MOGA run 2 converges at roughly generation 500 as both $\delta_i(m)$ and $\delta_o(m)$ become stable for $m \gtrsim 500$. For Tr-MOGA run 1, although the outputs appear to indicate convergence after $m \gtrsim 500$, the input metric $\delta_i(m)$ clearly shows that the optimization process has not yet fully converged on appropriate inputs to reach Pareto-optimal solutions.

5. Application of the ML-based multi-objective workflow to the ALS-U lattice case

From a comparison of the first ML-MOGA run (Fig. 6) with the final generation of conventional Tr-MOGA (Fig. 5, left), it becomes immediately clear that ML-MOGA has not yet reached a Pareto-optimal front in its initial configuration. However, retraining of the DNNs based on tracking data from validation runs and iterative application of ML-MOGA, as detailed in the previous section, can be shown to lead to ML-MOGA reaching a Pareto-optimal front that matches that of Tr-MOGA in only a fraction of the computation time required by the conventional approach. The critical ingredient here is the introduction of distance metrics as convergence criteria. Whenever these convergence criteria show an individual ML-MOGA run has converged, its final generation can be validated through tracking, this tracking data used to retrain the DNN surrogate models for DA and MA, and the retrained models then inserted into the next ML-MOGA run that is seeded with the population of the final generation of its predecessor run.

5.1. Convergence of iterative applications of ML-MOGA

At this point the remaining unknown is how many of these ML-MOGA–validation–retraining iterations are required until retraining of the DNNs renders no more benefit since ML-MOGA has already converged sufficiently close to the Pareto-optimal front (we will expand below on what we mean exactly by "sufficiently close"). This is a very important criterion because firstly, every additional iteration requires one more generation to be tracked which adds to the computational effort required by ML-MOGA and thus considerably affects its potential for overall speedup compared to Tr-MOGA, and secondly, because in the actual use case the Pareto-optimal front from Tr-MOGA is of course unknown so that one lacks any guidance as to how well the ML-MOGA problem has converged towards fully optimized results. The resolution to this challenge again lies in application of the Euclidean norm defined as a distance metric for objective space (cf. Eq. (2)). If we consider that the distance in objective space, as defined by this Euclidean norm, between the distribution of the final generation of an ML-MOGA run and the distribution of its validated results will gradually shrink as the benefit of retraining reduces, (because, e.g. the DNNs show less improvement from retraining once they are already well enough trained in the vicinity of those inputs that lead to Pareto-optimal solutions), we can use the reduction of this difference as an indicator of how may iterations are required to achieve full convergence of the ML-MOGA iteration campaign. For each ML-MOGA iteration, we can compute $\delta_o(m_f)$ of the final population m_f and its counterpart $\delta_o^v(m_f)$ for the validated population (i.e. the distribution in objective space as calculated by tracking the inputs of generation m_f) and inspecting their difference $\Delta_f = \delta_o^v(m_f) - \delta_o(m_f)$. Defined in this manner, this difference describes how far validated results are pushed from the golden target value in objective space compared to initial DNN-based predictions. As the retraining effort pay-off reduces when DNN predictions become accurate enough in the vicinity of the actual Pareto-optimal front, the difference between DNN prediction and actual tracking results will diminish and thus Δ_f from one iteration to the next will asymptotically reach its minimum. At that point, we can consider the iteration process complete and additional retraining of no further benefit, and the ML-MOGA optimization campaign fully converged.

5.2. Iterative application of ML-MOGA for optimization of the ALS-U lattice

In the following we will show, again using the 2nd MOGA stage for the ALS-U lattice optimization (9 quadrupole and 2 sextupole DoF), in detail how this iterative retraining process leads to convergence of ML-MOGA towards the Pareto-optimal front as illustrated by the conventional Tr-MOGA approach. Starting with the final population of the first iteration as detailed above (cf. Fig. 5, middle), this population is tracked for validation purposes and this data is combined with the initial training data so that the DNN surrogate models can be retrained. This retraining process uses the same framework and tools applied for the initial training process. It results again in excellent models with low RPEs for both DA and MA. The retraining process is still very quick since the training data is only incrementally increased.[12] As an

[11] This MOGA random seed is used for the breading process within MOGA (cross-over, mutation). It is not to be confused with the random seed used for error distribution within the lattice. Therefore, the perturbation applied here affects only the optimization process, not the underlying physics.

[12] In the first retraining step we add one generation of data to the initial 10 generations used for the original training process. Since roughly half of the children in this initial population had been rejected before training due to constraint violations, the data added for retraining corresponds to an about 20% increase of the training data. This relative increase in training data size obviously reduces with every additional iteration and retraining CPU time therefore remains low.

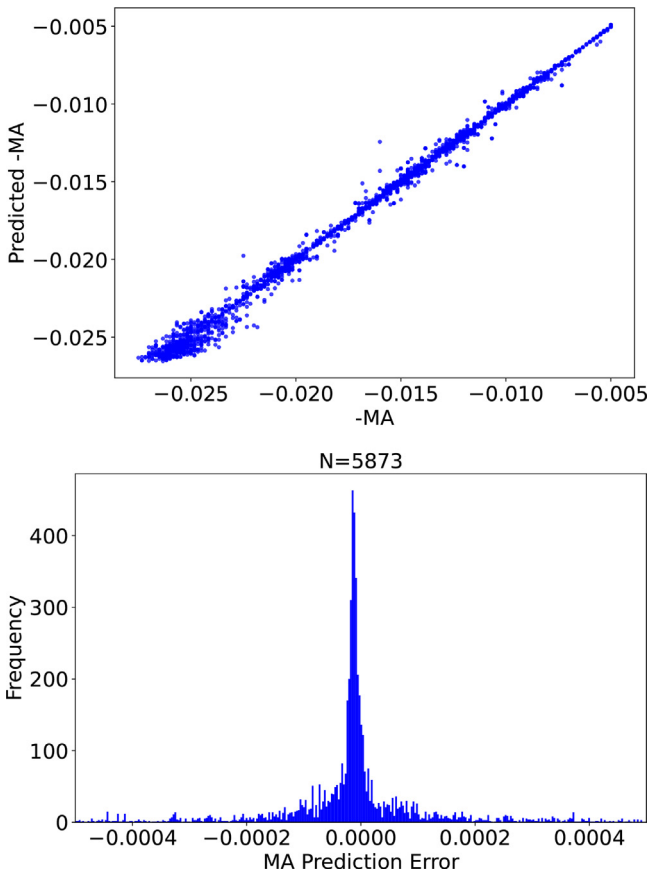

Fig. 8. Retraining of a DNN to serve a surrogate model for MA. Top: Comparison between actual MA from tracking and DNN prediction of MA based on the test data set. Bottom: Histogram of prediction error of the DNN surrogate model compared to the ground truth for the test data set (5873 samples).

example of this retraining effort, Fig. 8 shows the outcome of retraining for the MA surrogate DNN at this first retraining iteration. The error distribution is again very narrow, shows no systematic offset, and an RPE at this stage of just 1% is recorded. Once the DNN surrogate models have been successfully retrained, they are inserted into the ML-MOGA optimization and a second iteration can be launched using the final population of the first iteration as the seed. ML-MOGA is allowed to run for as many generations as required to converge, again determined by the derivatives of $\delta_{i,o}(m)$ both reducing to near-zero values. At that point this iteration is considered complete, the final generation validated through tracking, and this data can be used to retrain DNN models that are then inserted into the next ML-MOGA iteration which is again seeded with the final generation of the previous iteration. Such iterations can be repeated many times in order to push ML-MOGA results all the way to the Pareto-optimal front. Fig. 9 shows an example of the rank-1 validated solutions for several iterations of this retraining process. It is important to recognize here that an individual ML-MOGA run is quick (even if hundreds of generations are required for convergence, no tracking has to be carried out and DNN model lookup is near instantaneous), but that in this context it is the number of required additional iterations that comes with more significant computational cost since each iteration involves tracking the final generation for validation and retraining purposes.

Using the shift in distance metrics introduced above, Δ_f, we can observe how for this example, the retraining effort gradually improves the DNN surrogate models' predictive fidelity up to roughly iteration 6 (cf. Fig. 10) after which we recognize that additional iterations only lead to very small improvements ($\lesssim$15% reduction of Δ_f when adding additional iterations): at this point the validated solutions match the solutions predicted by DNN surrogate models so well that ML-MOGA can fully converge towards the Pareto-optimal front. For demonstration purposes, we have run additional iterations to show the evolution of Δ_f up to iteration 8. Finally, a comparison in objective space of ML-MOGA iteration 8 as per its DNN surrogate model predictions vs. the corresponding tracking-validated population is displayed in Fig. 11. And Fig. 12 shows convergence of ML-MOGA for each of the three variable pairs in objective space: significant improvements are noted up to roughly generations 4–6, whereas after generation 6 only very incremental changes can be observed.

5.3. Comparison of ML-MOGA results to conventional MOGA results

Comparison of the rank-1 validated solutions and the DNN-based predicted solutions shows just how reliable DNN predictions have become after 8 iterations of DNN retraining. Furthermore, a comparison of ML-MOGA validated rank-1 solutions (cf. Fig. 11, bottom) with the original Tr-MOGA Pareto-optimal front (cf. Fig. 5, left) reveals that ML-MOGA has managed to converge to fully optimized solutions. It is at this point crucial to compare also the computational effort behind these two approaches. While the original Tr-MOGA run required 643 generations to converge, the ML-MOGA campaign shown here relied on training data from 10 initial generations as well as another 6–8 generations of tracking data acquired at intermediate validation steps. With DNN lookup time and DNN retraining time being negligible compared to the CPU time required for tracking to reveal DA or MA, the ratio of these two generation numbers corresponds directly to the achieved speedup: a factor 36–40 depending on exact choice of cut-off Δ_f for retraining convergence (and a factor 40–45 for the second example shown in Fig. 10).

The result of the iterative ML-MOGA campaign does not just lead to good agreement of the Pareto-optimal front in objective space as compared to the conventional Tr-MOGA results, we also note that it renders the same occupied hypervolume in input space compared to the outcome of the original Tr-MOGA run. The iterative ML-MOGA campaign manages to identify the same quadrupole and sextupole gradients as the conventional Tr-MOGA approach that resulted in the most desirable solutions for this ALS-U lattice example. Finally, Fig. 13 displays this excellent agreement between the conventional Tr-MOGA and the iterative ML-MOGA campaigns in another manner: for each objective, sorted distributions of all children at the final generation are displayed and compared to the final distribution for the Tr-MOGA run. Note how not only the same ranges are covered for each dimension in objective space, but also that relative distributions within these ranges become nearly identical as the population approaches the best solutions. Apart from ML-MOGA slightly but systematically outperforming Tr-MOGA for emittance (cf. Fig. 13, left), discrepancies are noted primarily towards the poorest performing solutions, i.e. in areas farthest away from those around which retraining efforts have been focused.

5.4. Numerical stability and physics fidelity of the ML-MOGA optimization approach

In the aforementioned comparisons, it however also becomes clear that the results are not perfectly identical between conventional Tr-MOGA and ML-MOGA despite iterating retraining up to the point where no more substantial benefit is realized. In objective space, we note that while the validated rank-1 solutions at the final iteration (cf. Fig. 11, bottom) cover the same areas as the final solutions of the original Tr-MOGA run (cf. Fig. 5, left), we can see that the distribution density around the Pareto-optimal front is not identical. This is corroborated by Fig. 13, where one can clearly recognize that the distribution of results starts to deviate when moving farther from the most optimal results. Specifically, it appears that ML-MOGA favors slightly lower emittance at the expense of what appears to be somewhat reduced DA, as if the these two MOGA runs find solutions according to a slightly different relative weighting between objectives. This points to

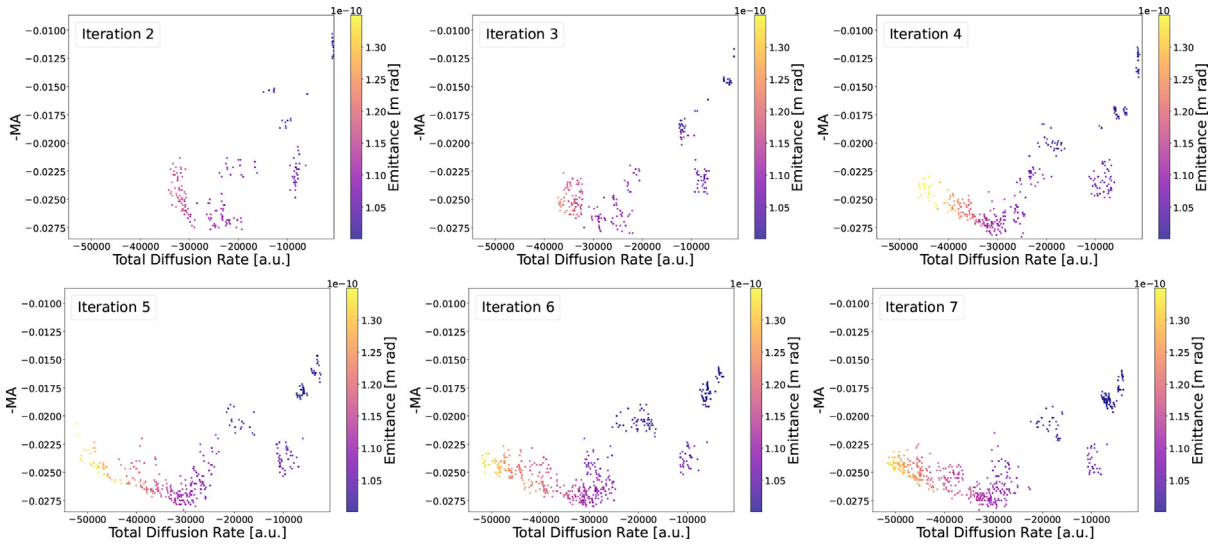

Fig. 9. Objective space plots for the validated rank-1 (i.e. non-dominated) solutions of the final generation of ML-MOGA iterations 2–7 based on retraining of the DNN surrogate models.

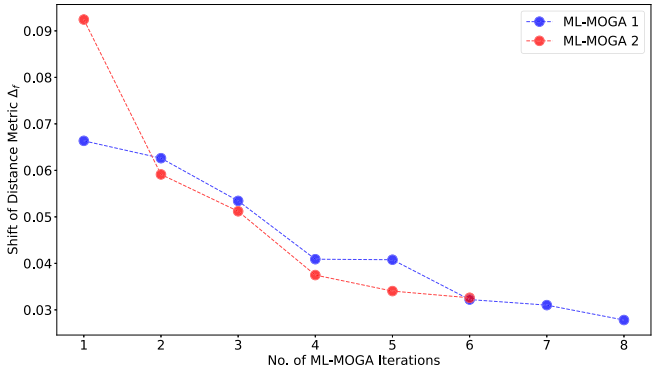

Fig. 10. Reduction of the distance metric shift Δ_f with increasing number of iterations of the ML-MOGA–validation–retraining cycle. Two cases with different MOGA random seeds are shown. In the case of ML-MOGA 1, significant progress from DNN retraining is observed up to roughly iteration 6. For the case of ML-MOGA 2, already at iteration 4 sufficient retraining appears to have been carried out.

the stochastic nature of the genetic algorithm as a possible source of these discrepancies.

In order to quantify the observed differences we present two alternate MOGA campaigns: one in which we modify the random seed used by the MOGA process for breading (mutation and cross-over) and the other in which we modify the random seed used by Tracy when it applies the initial error distribution to the lattice used for tracking. The former affects only the stochastics of the MOGA process, while the latter will affect the underlying physics of the lattice optimization problem. In the case of the former, we would expect to find the same Pareto-optimal front resulting from the same occupied hypervolume in input space provided we have converged to a stable solution, while in the latter case it is not a priori clear that solutions would lie on the same Pareto-optimal front or call for the same inputs to reach this front. In fact, the lattice designer has to assume that their initial choice of lattice error distribution can indeed affect the outcome of the lattice optimization and therefore, one of the first cross-checks commonly applied to an optimized lattice is to verify that using alternate error distributions still leads to the same desired lattice performance using the same (or at least very similar) configurations in input space.

Fig. 14 compares Tr-MOGA and rank-1 validated ML-MOGA runs at the final generation for three cases: the original lattice optimization, the same optimization run but using a different MOGA random seed, and a run where the random seed for the underlying lattice error distribution

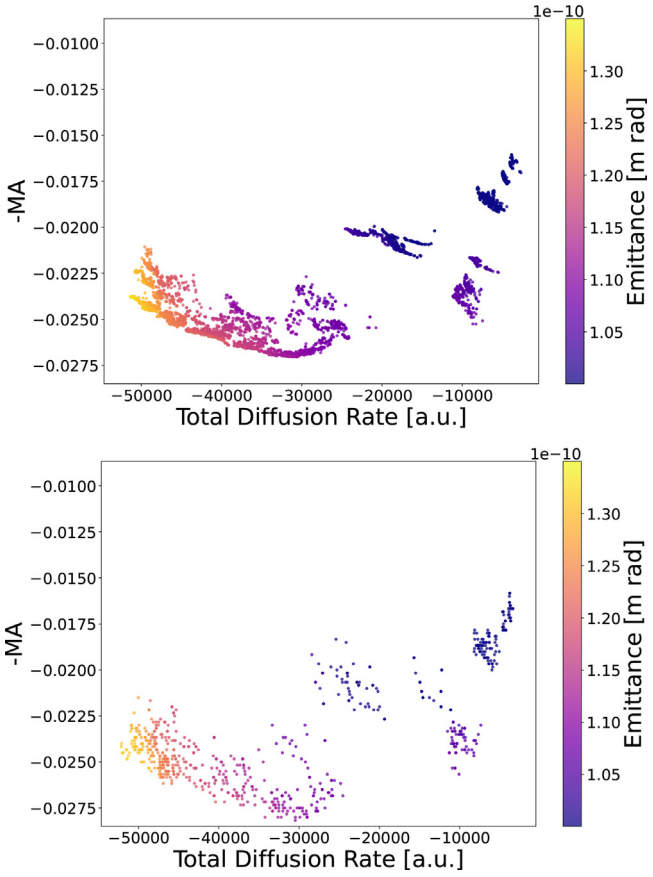

Fig. 11. Objective space plots for ML-MOGA iteration 8 comparing results as predicted by the DNN surrogate models (top) with rank-1 validated results (bottom).

has been altered. We note that the objective space distributions of conventional Tr-MOGA when changing only the MOGA random seed already show perceivable differences. Although the Pareto-optimal fronts are similar, there are clearly also differences in terms of density in objective space. In fact, a class of solutions with large DA, moderate emittance, but only limited MA can be recognized in the run with the alternate random seed (Fig. 14, top middle) that hardly shows up in the original Tr-MOGA run (Fig. 14, top left). Similarly, the

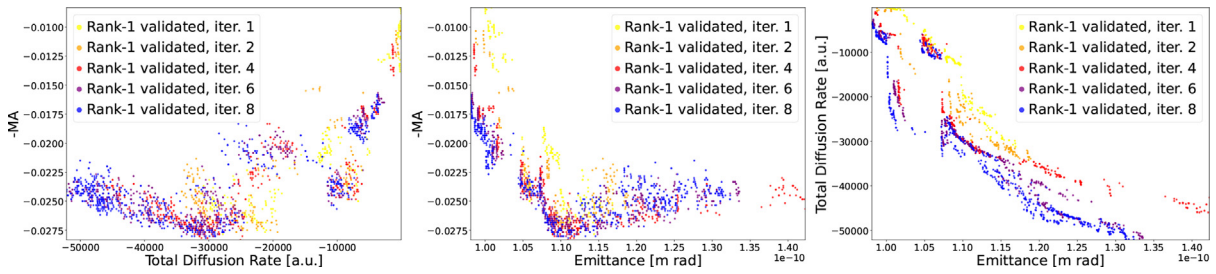

Fig. 12. Rapid convergence of ML-MOGA with iterative retraining. Rank-1 validated solutions at the final generation of iteration 1 to 8 are shown for each of the three pairs of objective variables.

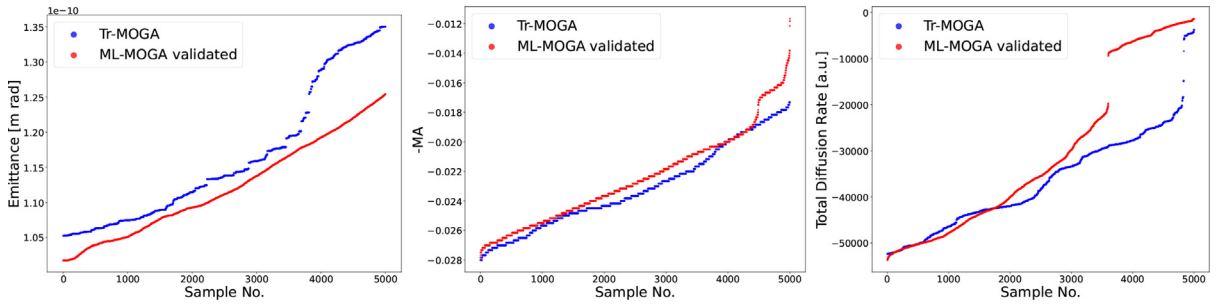

Fig. 13. Comparison between Tr-MOGA and ML-MOGA runs in each dimension of objective space. In each case, the children of the population have been sorted according to their result in that dimension of objective space. In the case of the ML-MOGA populations, tracking-validated results are displayed.

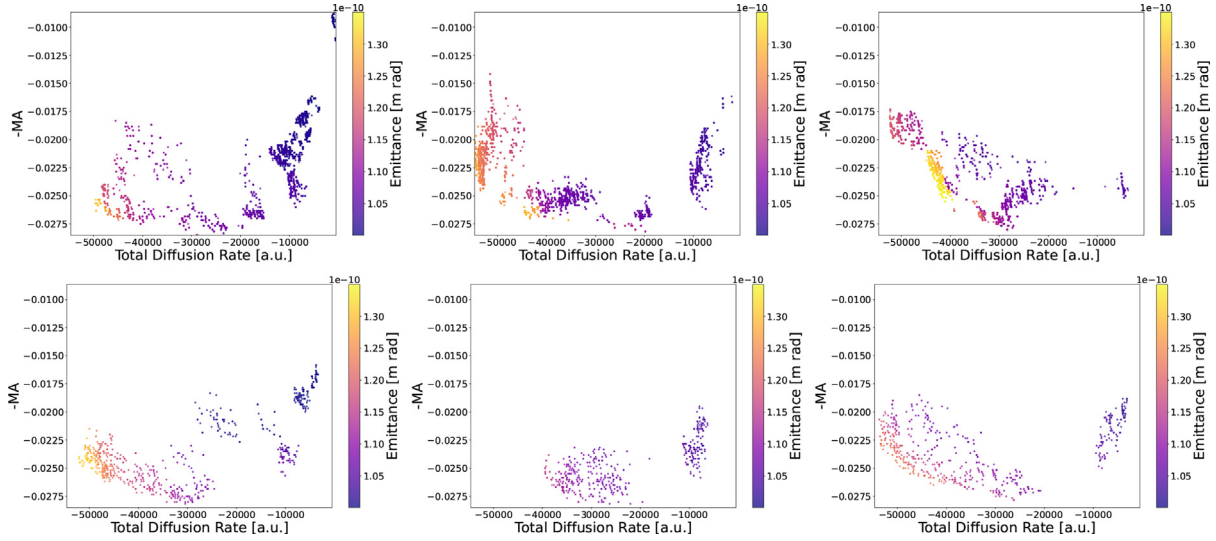

Fig. 14. Comparison between Tr-MOGA (top) and rank-1 validated ML-MOGA (bottom) runs in objective space at their final generations. Left: the original lattice optimization. Middle: same optimization but with different MOGA random seed. Right: optimization with different random seed for underlying error distribution.

original Tr-MOGA run results in a class of solutions with very low emittance at near-zero DA and with only 1% MA. This class of solutions does not appear in the Tr-MOGA run with alternate random seed. In comparison to such deviations, the discrepancies between ML-MOGA and Tr-MOGA runs based on the same random seed appear small—in fact, they are no larger than the differences that result from different MOGA stochastics alone as demonstrated by comparing the original Tr-MOGA run (Fig. 14, top left) with the Tr-MOGA run where only the random MOGA seed has been altered (Fig. 14, top middle). When comparing the results of Tr-MOGA based on a different lattice error seed, we notice a clearly different Pareto-optimal front. The initial case shows only a weak trade-off between MA and DA (instead the trade-off appears primarily between DA and emittance), whereas in the case of the alternate error lattice seed, it is evident that best MA can only be achieved at intermediate levels of DA while the highest DA results render reduced MA and elevated emittance. Note also, that in this case of the alternate lattice error seed, the ML-MOGA results (Fig. 14, bottom right) accurately reproduce this trade-off as its Pareto-optimal front closely matches that of the corresponding Tr-MOGA run (Fig. 14, top right). And if we focus on just ML-MOGA results, the different trade-off behavior revealed by changing the lattice error seed can be clearly recognized. Such differences in trade-off (i.e. varying levels of correlation between different objective variables), are very clearly illustrated by an alternate representation of objective space: Fig. 15 shows a direct comparison of Tr-MOGA and ML-MOGA results for two different error lattice seeds according to all three variable pairs in objective space. Clearly, most deviations between ML-MOGA and Tr-MOGA are small and in fact all deviations are considerably smaller than differences that result from a change in the underlying physics (lattice error distribution).

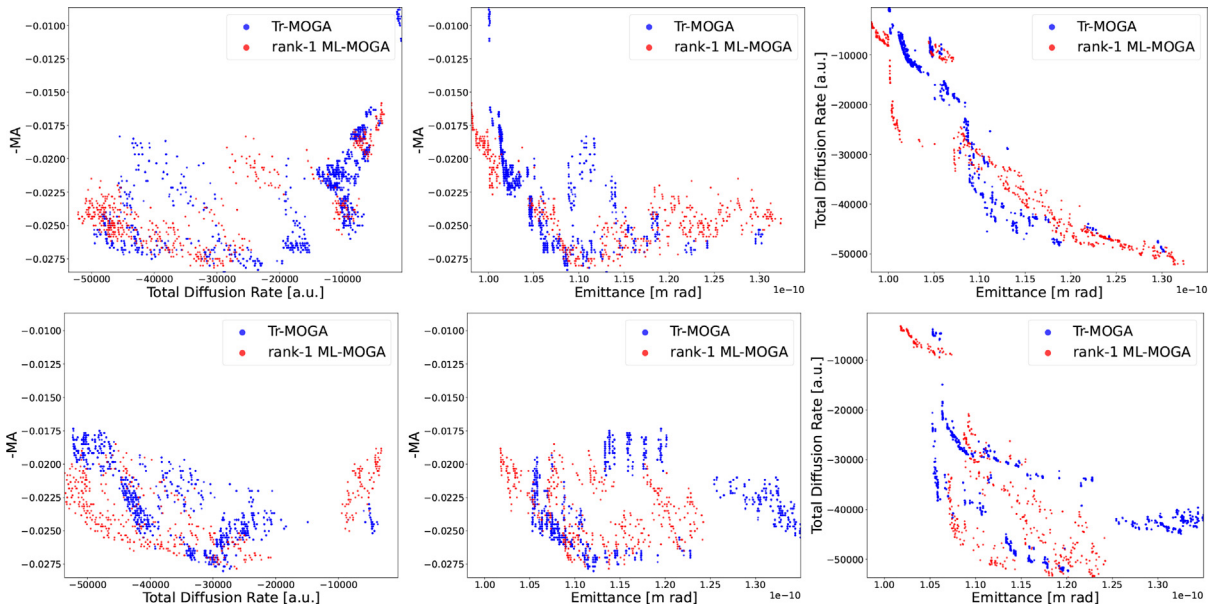

Fig. 15. Comparison between Tr-MOGA and rank-1 validated ML-MOGA runs at their final generations for each of the three pairs of objective variables. Top: the original lattice optimization. Bottom: optimization with different random seed for underlying error distribution.

6. Conclusion & outlook

We can conclude from these studies, that ML-MOGA—assuming sufficient DNN retraining iterations have been carried out—indeed reproduces the Pareto-optimal front rendered by Tr-MOGA. We also conclude that differences noted between Tr-MOGA and ML-MOGA are small. In fact, they are no larger than discrepancies arising from numerical noise due to the stochastic nature of the conventional MOGA optimization process. The results of the ML-MOGA optimization process are true to the underlying physics: differences resulting from variations in the error distributions are reproduced. Such differences due to changes in the underlying physics remain large compared to differences arising from just the stochastics of the optimization process. Therefore, we can employ ML-MOGA as a full-fledged replacement for traditional Tr-MOGA in the optimization of 4GSR lattices.

We have shown how ML, and DNNs specifically, can be incorporated into an existing MOGA workflow based on many-turn particle tracking with only minimal changes to the MOGA code and without requiring any changes to the tracking code itself. We have demonstrated how iterative deep learning retraining processes allow the DNN-based surrogate models to continuously improve to the point where ML-MOGA can fully converge towards the actual Pareto-optimal front. Furthermore, by virtue of model-independent convergence metrics, we can demonstrate that such convergence can be realized at minimum computational cost. The overall speedup provided by ML-MOGA over the conventional Tr-MOGA approach is almost a factor 50 without sacrificing fidelity to the underlying physics. Such tremendous speedup of what traditionally is an excessively lengthy process, opens up entirely new possibilities.

For a full lattice optimization strategy, the lattice designer would ideally strive to optimize several lattices based on different error seeds simultaneously, however, in present-day design campaigns this is usually not possible due to excessive computation time. The massive speedup provided by ML-MOGA in fact now finally enables such an approach. Ultimately, a candidate lattice can be selected corresponding to a small region in input hypervolume where, for all investigated lattice error distributions, the desired high performance results. Note, that this is not feasible in a conventional Tr-MOGA campaign without accepting massive runtime increases (far beyond 1–2 months on a large cluster).[13] Whereas in the classical Tr-MOGA approach, one error lattice is used to run the complete and lengthy optimization, and errors are then perhaps changed once or twice (rarely more due to excessive runtime) to confirm that performance ends up reasonably close, with ML-MOGA we have the opportunity to—right from the start—optimize in parallel for many error distributions. Since we a priori do not know which error seed will end up realized in the as-built machine, optimizing in this way for many error distributions ensures convergence onto a truly *global* optimum. Following this reasoning then dictates how optimal lattice candidates are selected: candidate lattice are chosen by selecting desirable seeds from the Pareto-optimal front for which input space subvolumes show good overlap across many lattice error seed runs.

Not only does such an approach ensure that the real machine (for which the error lattice is unknown in advance of commissioning) will be able to achieve the targeted and promised beam parameters, but it also has profound implications on engineering design and commissioning simulation requirements. The engineering design does not have to accommodate for the large tuning capability required by calling for various errors to be compensated such that a specific design error lattice can be realized. Instead, the engineering design only has to provide for a tuning range that corresponds to the error distributions applied during the design optimization. The engineering design can then be extended if, and only if, it can be shown to lead to a performance gain that could not otherwise be realized. Likewise, a broad ML-MOGA campaign based on parallel optimization of multiple error seeds relaxes the risk associated with commissioning and thus the effort that has to be invested into commissioning simulation. We note here, finally, that in the aforementioned ALS-U lattice optimization case, the ML-MOGA speedup enables to optimize for 10 error lattices in parallel and still achieves an overall speedup of a factor 4 over the conventional Tr-MOGA process for just a single error lattice. This brings the overall lattice design period down from many months (in spite of a massive cluster) to just a couple weeks.

By virtue of the manner in which this ML-MOGA workflow has been designed and by employing normalized convergence criteria independent of the specific optimization problem, it can also easily be

[13] Also, it is important to recognize here, that efficient physics–engineering iterations during the design phase become hard to realize if lattice optimization itself reaches the scale of months.

expanded to include additional DoF such as reverse bending, high-field bend magnets, octupoles, etc. As long as a DNN can be tuned to deliver good predictions for the targeted objectives, this workflow can be realized. Also, provided DNN models are available (or their generation sufficiently well automated [31]), the lattice designer running the optimization is not required to be equipped with specific ML knowledge. Inclusion of the DNN models in the MOGA optimization is trivial and retraining of these models can be scripted with readily available open source tools such that a lattice designer can remain focused solely on the lattice optimization itself.

Future work should focus on more advanced data selection for retraining purposes. Since the realized speedup is strongly affected by how many tracking validation steps have to be undertaken, any method that leads to better retraining has the potential to accelerate ML-MOGA convergence, thus requiring fewer retraining iterations and thereby increasing the direct speedup. Furthermore, we should continue to investigate the robustness of the ML-MOGA approach, specifically extending these studies to investigating how strongly DNN predictions are affected by minor changes in the underlying lattice, such as small changes of magnet position or length, as commonly encountered during physics–engineering integration in the later stages of light source optimization. If small changes do not require fresh training of the DNNs so that they can be reused despite small perturbations, this has the potential to again save significant tracking effort and thus additionally accelerate the overall MOGA-based design process. Finally, the potential to automate this entire process is highly attractive.

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

Data will be made available on request.

Acknowledgments

The authors would like to thank Rob Ryne for his generous support of our supercomputing efforts. We are grateful to Fernando Sannibale for his support and encouragement, as well as proofreading of the manuscript. This research is funded by the US Department of Energy (BES & ASCR Programs), and supported by the Director of the Office of Science of the US Department of Energy under Contract No. DEAC02-05CH11231.

References

[1] MAX IV Detailed Design Report, available at https://www.maxiv.lu.se/beamlines-accelerators/accelerators/accelerator-documentation-2/.
[2] S.C. Leemann, et al., Beam dynamics and expected performance of Sweden's new storage-ring light source: MAX IV, Phys. Rev. ST Accel. Beams 12 (2009) 120701, http://dx.doi.org/10.1103/PhysRevSTAB.12.120701.
[3] M. Tadahiko, I. Hisao, MOGA: Multi-objective genetic algorithms, in: Proc. IEEE International Conference on Evolutionary Computation, 1995, pp. 289–294, http://dx.doi.org/10.1109/icec.1995.489161.
[4] Lingyun Yang, David Robin, Fernando Sannibale, Christoph Steier, Weishi Wan, Global optimization of an accelerator lattice using multiobjective genetic algorithms, Nucl. Instrum. Methods Phys. Res. Sect. A 609 (1) (2009) 50–57, http://dx.doi.org/10.1016/j.nima.2009.08.027.
[5] Weiwei Gao, Lin Wang, Weimin Li, Simultaneous optimization of beam emittance and dynamic aperture for electron storage ring using genetic algorithm, Phys. Rev. ST. Accel. Beams 14 (9) (2011) 094001, http://dx.doi.org/10.1103/PhysRevSTAB.14.094001.
[6] C. Sun, D.S. Robin, H. Nishimura, C. Steier, W. Wan, Small-emittance and low-beta lattice designs and optimizations, Phys. Rev. ST Accel. Beams 15 (5) (2012) 054001, http://dx.doi.org/10.1103/PhysRevSTAB.15.054001.
[7] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, IEEE Trans. Evol. Comput. 6 (2) (2002) 182–197, http://dx.doi.org/10.1109/4235.996017.
[8] M.T.M. Emmerich, A.H. Deutz, Nat. Comput. 17 (3) (2018) 585–609, http://dx.doi.org/10.1007/s11047-018-9685-y.
[9] M. Kranjčević, B. Riemann, A. Adelmann, A. Streun, Multiobjective optimization of the dynamic aperture using surrogate models based on artificial neural networks, Phys. Rev. Accel. Beams 24 (1) (2021) 014601, http://dx.doi.org/10.1103/PhysRevAccelBeams.24.014601.
[10] Minghao Song, Xiaobiao Huang, Linda Spentzouris, Zhe Zhang, Storage ring nonlinear dynamics optimization with multi-objective multi-generation Gaussian process optimizer, Nucl. Instrum. Methods Phys. Res. Sect. A 976 (2020) 164273, http://dx.doi.org/10.1016/j.nima.2020.164273.
[11] Yongjun Li, Weixing Cheng, Lihua Yu, Robert Rainer, Genetic algorithm enhanced by machine learning in dynamic aperture optimization, Phys. Rev. Accel. Beams 21 (5) (2018) 054601, http://dx.doi.org/10.1103/PhysRevAccelBeams.21.054601.
[12] Jinyu Wan, Paul Chu, Yi Jiao, Neural network-based multiobjective optimization algorithm for nonlinear beam dynamics, Phys. Rev. Accel. Beams 23 (8) (2020) 081601, http://dx.doi.org/10.1103/PhysRevAccelBeams.23.081601.
[13] A. Streun, Nucl. Instr. and Meth. A 737 (2014) 148, http://dx.doi.org/10.1016/j.nima.2013.11.064.
[14] L. Emery, Global optimization of damping ring designs using a multi-objective evolutionary algorithm, in: Proceedings of 2005 Particle Accelerator Conference, Knoxville, TN, USA, pp. 2962–2964, https://accelconf.web.cern.ch/p05/PAPERS/RPPP047.PDF.
[15] Ivan V. Bazarov, Charles K. Sinclair, Phys. Rev. ST Accel. Beams 8 (2005) 034202, http://dx.doi.org/10.1103/PhysRevSTAB.8.034202.
[16] Hiroshi Nishimura, TRACY: A tool for accelerator design and analysis, in: Proceedings of the 1st European Particle Accelerator Conference (EPAC 88), Vol. 7–11, Rome, Italy, 1988, pp. 803–805, https://accelconf.web.cern.ch/e88/PDF/EPAC1988_0803.PDF.
[17] C. Sun, D.S. Robin, H. Nishimura, C. Steier, W. Wan, Phys. Rev. ST Accel. Beams 15 (2012) 054001, http://dx.doi.org/10.1103/PhysRevSTAB.15.054001.
[18] C. Sun, et al., Design of the ALS-u storage ring lattice, in: Proc. 8th Int. Particle Accelerator Conf. (IPAC'17), Copenhagen, Denmark, 2017, pp. 2827–2829, http://dx.doi.org/10.18429/JACoW-IPAC2017-WEPAB105.
[19] C. Sun, Optimization of nonlinear dynamics for ALS-u lattices, in: ICFA Mini-Workshop on Dynamic Apertures of Circular Accelerators, IHEP, Beijing, 2017, pp. 1–3, https://indico.ihep.ac.cn/event/7021/session/9/contribution/30/material/slides/0.pdf.
[20] C. Bernadini, et al., Phys. Rev. Lett. 10 (1963) 407, http://dx.doi.org/10.1103/PhysRevLett.10.407.
[21] C. Montag, J. Bengtsson, B. Nash, Touschek lifetime calculations and simulations for NSLS-II, in: Proceedings of PAC07, Albuquerque, New Mexico, USA, pp. 4375–4377, https://accelconf.web.cern.ch/p07/PAPERS/FRPMS113.PDF.
[22] https://it.lbl.gov/resource/hpc/supported-research-clusters/alsacc/.
[23] ALS-U Conceptual Design Report, LBNL, 2017.
[24] Yann LeCun, Yoshua Bengio, Geoffrey Hinton, Deep learning, Nature 521 (7553) (2015) 436–444, http://dx.doi.org/10.1038/nature14539.
[25] Y. Lu, et al., Enhancing the MOGA optimization process at ALS-u with machine learning, MOPAB106, in: Proceedings of IPAC2021, Campinas, Brazil, 2021, pp. 387–390, http://dx.doi.org/10.18429/JACoW-IPAC2021-MOPAB106.
[26] Vinod Nair, Geoffrey E. Hinton, Rectified Linear Units Improve Restricted Boltzmann Machines, ICML, 2010.
[27] Diederik P. Kingma, Jimmy Ba, Adam: A method for stochastic optimization, 2014, arXiv preprint arXiv:1412.6980.
[28] https://pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.ReduceLROnPlateau.html.
[29] https://pytorch.org/docs/stable/jit.html.
[30] Auralee Edelen, Nicole Neveu, Matthias Frey, Yannick Huber, Christopher Mayes, Andreas Adelmann, Machine learning for orders of magnitude speedup in multiobjective optimization of particle accelerator systems, Phys. Rev. Accel. Beams 23 (2020) 044601, http://dx.doi.org/10.1103/PhysRevAccelBeams.23.044601.
[31] Haifeng Jin, Qingquan Song, Xia Hu, Auto-Keras: An efficient neural architecture search system, 2019, arXiv preprint arXiv:1806.10282v3.