Received 24 January 2024; accepted 14 February 2024. Date of publication 16 February 2024; date of current version 4 March 2024.

Digital Object Identifier 10.1109/OJCOMS.2024.3366770

Enhancing Adaptive Beamforming in 3-D Space Through Self-Improving Neural Network Techniques

IOANNIS MALLIORAS [1,2] (Graduate Student Member, IEEE), TRAIANOS V. YIOULTSIS [1] (Member, IEEE), NIKOLAOS V. KANTARTZIS [1] (Senior Member, IEEE), PAVLOS I. LAZARIDIS [3] (Senior Member, IEEE), AND ZAHARIAS D. ZAHARIS [1] (Senior Member, IEEE)

[1]School of Electrical and Computer Engineering, Aristotle University of Thessaloniki, 54124 Thessaloniki, Greece
[2]R&D Department, Maggioli SpA, 47822 Santarcangelo Di Romagna, Rimini, Italy
[3]School of Computing and Engineering, University of Huddersfield, HD1 3DH Huddersfield, U.K.

CORRESPONDING AUTHOR: I. MALLIORAS (e-mail: mallioras@auth.gr)

This work was supported in part by the European Union through the Horizon 2020 Marie Skłodowska-Curie Innovative Training Networks Programme "Mobility and Training for Beyond 5G Ecosystems (MOTOR5G)" under Grant 861219, and in part by HEAL-Link.

ABSTRACT In the rapidly evolving domain of wireless networks, adaptive beamforming stands as a cornerstone for achieving higher data rates, enhanced network capacity, and reduced latency. This study introduces a novel integration of deep neural networks (NNs) into adaptive beamforming, specifically for uniform planar arrays (UPAs). We embark on an exploration of different NN architectures (a deep feedforward NN, a gated recurrent unit, and a long short-term memory based recurrent NN) and hyperparameter tuning techniques (grid search, Bayesian optimization, and genetic algorithm) to assemble a model that yields the most promising zero forcing beamforming performance. This thorough examination not only unveils the most prominent model but also highlights the most efficient hyperparameter tuning method. A comprehensive dataset generated using the null steering beamforming algorithm applied to an 8×8 UPA, serves as the bedrock for training these networks. Our evaluation benchmarks the NN-based beamformers against traditional methods, assessing them on crucial metrics such as beamforming competence, response time, and adaptability under various noise and interference scenarios. A significant innovation of this research is the introduction of a continual learning approach, enhancing the NN beamformers' performance in dynamically changing environments. This self-improving algorithm represents a stride towards more adaptable and efficient beamforming, showcasing improvement on underperforming scenarios without compromising accuracy. Our findings demonstrate the potential of neural networks in reshaping the future of adaptive beamforming, offering a blend of speed and precision that is paramount in modern wireless networks.

INDEX TERMS Adaptive beamforming, null-steering beamforming, Bayesian optimization, deep learning, gated recurrent unit (GRU), genetic algorithms, long short-term memory (LSTM), neural networks, planar antenna arrays, recurrent neural networks.

I. INTRODUCTION

THE NEXT generation of wireless networks is expected to offer great capabilities and facilitate a wide range of applications. Amongst the enabling factors promised with 6G, are increased data rates, greater network capacity, and lower latency. To achieve the high standards set for 6G, fundamental antenna operations must be optimized or implemented with a completely novel approach. In this work, we focus on one of these smart antenna operations, and specifically a physical layer operation called adaptive beamforming.

Adaptive beamforming (ABF) [1] is an essential real-time process of smart antennas which will play a pivotal role in future smart antenna operation. It is a method by which an antenna regulates and steers its radiating energy dynamically, to maintain a strong connection with

the end-user. Establishing a strong and reliable connection depends on the ability of a smart antenna to control and direct its radiation in a desired manner. In this work, we are interested in a uniform planar array (UPA) which is comprised of many small radiating elements arranged in a flat, two-dimensional configuration. Each element can be connected to a feeding network. By manipulating the amplitude and phase of the feeding currents we can control the radiation of each element, and in turn, the collective radiation of the UPA.

This way, the point of maximum gain (the peak of the main lobe) can be steered towards directions we intend to send and receive signals from (i.e., signals of interest or SoIs). At the same time, unwanted incoming signals (i.e., signals of avoidance or SoAs) can be rejected when points of minimum gain, known as "nulls", are formed towards their directions. To adequately reject an SoA, it is necessary to create a null or a low sensitivity region towards its direction. Maximizing the energy of a beam towards a desired user while rejecting interfering signals in a noisy environment, improves the signal-to-interference-plus-noise ratio (SINR), the data-rate, and even the capacity of the system [2]. That is because it enables a more efficient utilization of the available spectrum, consequently supporting a larger number of users. A control system that adjusts these feeding signals based on different scenarios of SoI and SoAs is called a beamformer. In other words, a beamformer can be an algorithm that given a set of angles of arrival (AoAs) of incoming signals, calculates the proper feeding weights that produce the radiation pattern that is needed.

Many deterministic algorithms capable of providing accurate results have been developed, such as the minimum variance distortionless response algorithm (MVDR) or the null-steering beamforming algorithm (NSB) also known as the zero-forcing algorithm [3], [4], [5], [6], [7]. However, these algorithms either rely on highly complex mathematical operations, or demand multiple iterations before converging to the desired values of the beamforming weight vector. Therefore, while effective, these algorithms are constrained by their slow temporal response and limited adaptability in dynamic environments. These limitations become increasingly pronounced in next-generation wireless networks, where real-time processing and adaptability are crucial.

To address these challenges, our research turns to the realm of machine learning, known for its prowess in handling complex, multi-objective tasks and its ability to learn and adapt from experience. Because of the inherently complex and highly non-linear nature of the problem at hand, and considering prior research in this field has shown that shallow neural networks (NNs) struggle to cope with its demanding complexity [8], we opt to utilize deeper models within the Deep Learning (DL) domain. Unlike conventional beamforming algorithms that rely on predefined mathematical models, deep NNs have the ability to learn the optimal beamforming strategy directly from data. They can extract meaningful patterns from large datasets, adapt

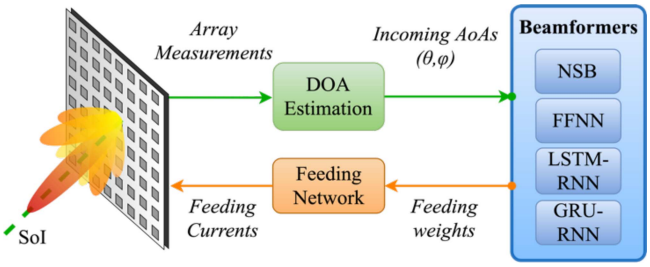

FIGURE 1. Adaptive beamforming cycle.

to changing signal conditions, and make real-time decisions. This makes them ideal candidates for overcoming the limitations of traditional beamforming methods [9]. Their most time-consuming process, training a DL model, happens offline, without interfering with the operation of the NNs when deployed. As shown in [10], the use of DL, and specifically NNs, has the potential to significantly improve the response time and efficiency of ABF.

Developments in direction of arrival (DOA) estimation algorithms [11] indicate that implementing beamforming using AI can be a two-step assignment (also shown in Fig. 1). At the first step, a capable DOA estimator implemented with NNs predicts the AoAs of incoming signals given the array measurements and channel state information (CSI) [12], [13], [14], [15]. The second step consists of a NN beamformer, producing the appropriate feeding currents that through the feeding network will give the desired radiation pattern for each scenario [16]. As demonstrated in Fig. 1, we consider a modular approach where the DOA estimation is not incorporated into the beamforming operation of the proposed NN but could be assigned to a different NN or DOA estimation technique. Thus, while imperfect CSI does not directly impact the operation of the proposed NN-beamformer, it can have an indirect effect, primarily if the accuracy of DOA estimation is compromised. For example, the misinterpretation of the true AoAs of incoming signals due to scatterers in the channel can be an indirect threat to the accuracy of our approach. To put it simply, if the DOA estimation fails to predict the true AoAs then our model's response will be equally misaligned. Our scheme relies upon a robust DOA estimation technique, able to deal with scatterers and interference. However, this is not the focus of our study.

The primary objective of the proposed NN beamformer is to accurately map the input AoAs to desired weight vectors independent of the CSI use. In order to better define the purpose of our work, we need to clarify some distinctions concerning the term *beamforming*, as also described in [17]. First, the term *beam-shaping* refers to the capability of the antenna to reform the shape of the radiation pattern without altering the direction of the main lobe. Conversely, *beam-steering* implies solely the change of the direction of the main lobe of the radiation pattern. Furthermore, the term *non-continuous beam-steering* is used to express the ability of an antenna system to steer the main lobe towards

a limited number of directions. In contrast, *continuous beam-steering* includes all possible directions within the antennas spatial operating range. Our goal is to identify a NN model that allows *adaptive, continuous beam-steering* capable of *beam-shaping*, when nulls are placed towards the directions of unwanted interfering signals. This approach greatly differs from the fixed beamforming approach (e.g., code-book method) as it aims to dynamically produce the beamforming weight vector in real time.

Our research makes several key contributions to the field:

1) *Comparative Analysis of NN Architectures*. We conduct an in-depth comparison of three distinct neural network architectures – deep feedforward networks (FFNN), gated recurrent units (GRUs), and long short-term memory (LSTM) networks – in the context of adaptive beamforming. This analysis provides valuable insights into the suitability and performance of each architecture for beamforming tasks in 3D space.
2) *Dataset Generation and Hyperparameter Optimization.* Utilizing a substantial dataset generated via the NSB algorithm applied to an 8×8 UPA, our study not only explores the optimal neural network configurations but also emphasizes the importance of hyperparameter tuning. We employ advanced techniques such as grid search, Bayesian optimization, and genetic algorithms to identify the most effective hyperparameters, thereby enhancing the beamforming performance.
3) *Performance Evaluation in Various Scenarios*. We evaluate the effectiveness of the NN-based beamformers by comparing them with a traditional beamforming algorithm. This comparison spans several criteria, including beamforming competence, response time, and robustness under diverse noise and interference scenarios.
4) *Introduction of a Continual-Learning Approach.* In light of the dynamic nature of wireless network environments, we introduce a novel continual-learning technique. This approach is designed to further enhance the NN beamformers' adaptability and performance by continually training them on increasingly complex cases, thereby ensuring their effectiveness in a dynamically changing operational environment.

II. RELATED WORK

There have been several works where machine learning and deep learning techniques have been applied on this field. The authors of [18] propose a deep learning technique to predict the beamforming vectors from sub-6 GHz uplink channel state prediction as input, by exploiting the features in the latter. In [19] a joint channel predictor and beamforming model is proposed which aims to maximize the long-term expected sum rate. The proposed reinforcement learning (RL) based method utilizes a zero-forcing precoding method to generate the beamforming vectors. A convolutional NN (CNN) is proposed in [20] which, for a given uplink channel estimate as input, outputs downlink channel information to be used for beamforming. The CNN model significantly improves performance (lower bit-error rate) compared to traditional zero-forcing beamforming based on the uplink channel estimate. Our work fundamentally diverges from similar literature in the metrics used to evaluate beamforming schemes. While the previous studies rely on indirect metrics such as bit-error rate or average sum rate, our research prioritizes metrics directly related to the physical aspects of beamforming. Even though these works incorporate channel state predictors instead of relying on DOA estimation algorithms, they do not delve into the accuracy of the radiation patterns regarding the main lobe and null steering. In addition, they mainly deal with uniform linear arrays (ULA) where in our work we focus on the much more demanding task of beamforming on UPAs.

Applications more closely aligned with our own, consider using either the autocorrelation matrix of the signals or their respective AoAs to produce the beamforming vectors. Comparisons between different NN architectures applied to 16-element realistic ULAs are performed in [16] and [21] with very promising results. In [22], an Invasive Weed Optimization (IWO) variant is used to train a feedforward neural network (FFNN) for accurate null-steering beamforming on ULAs. The results show that for low noise conditions, the model is able to achieve high angular accuracy on main lobe and null steering. A radial basis function NN (RBFNN) is proposed in [8] to predict the weights of a 6-element ULA. This work attempts to use a NN to extract the angles of arrival (AoAs) of the incoming signals from their autocorrelation matrix. These angles are in turn fed to an RBFNN to predict the feeding weights. Despite the temporal advantage of using shallow NNs and a relatively simplistic beamforming scenario which could lead to fast training and good accuracy, the author concludes that NNs inability to perform adequate null-steering beamforming relies on the lack of training samples. In [23] a FFNN approach is presented to predict the direction of the main lobe in a fixed beamforming scenario for a ULA without addressing the null-steering concept. In [24] a CNN is employed to predict the weight vectors of a ULA. The CNN of this case takes the autocorrelation matrix as input. The results show how promising a NN can be in terms of time response compared to conventional deterministic beamformers. MVDR and loaded sample matrix inversion (LSMI) are compared in this study. The work concludes with the proposed CNN-beamformer having an output SINR only 0.5 dB below the optimum SINR. In our work, we further decrease this degradation. The necessity of a larger training dataset is also raised in this work.

Already, the SINR levels NNs can achieve on UPA implementations shows a lot of potential [25]. A UPA beamforming application is presented in [26] where a CNN is used to predict the desired phase shifts for an 8×8 UPA given the desired radiation pattern as input. The beamforming

accuracy of this model is portrayed merely by an image comparison of the desired radiation pattern and the one produced by the CNN. In our work, we present a full statistical analysis on the accuracy of our NN-beamformers on a wide range of scenarios. The authors of [27] propose a dense NN where partial and perfect channel state information are given as input, and beamforming vectors for confidential information signal are taken as output for a 4×4 planar array, aiming at maximizing the average secrecy rate of the considered unmanned aerial vehicle communication system. Once again, the accuracy aspect of zero-forcing beamforming is not addressed. In [28] a DL-enabled beamforming system controlled by a central cloud processor is proposed. On this application a deep FFNN is used to pick the most appropriate beamforming vector out of a predefined codebook of 1024 beams. This work demonstrates how the low complexity of DL algorithms allows multiple base stations to operate under the control of a single cloud processor, which elevates the importance of DL in highly mobile systems. No zero-forcing strategy has been implemented in this work and the discrete space of 1024 beams is not comparable to our continuous beamforming approach. Another codebook-based beamforming system is shown in [29] implemented with DL and RL algorithms. This beam-steering application employs RL to further enhance the capabilities of DL by letting it constantly learn from its surroundings. Despite portraying the capabilities of DL in ABF, these works consider the non-continuous, fixed beam-steering approach. The DL algorithms are used to accurately select between predefined beamforming weight vectors for each occurring scenario. In our application we are interested in using DL to design these beamforming weight vectors in real-time for each individual case. Finally, the authors of [6] present an alternative method to produce the beamforming weights of a UPA employing a variation of the particle swarm optimization algorithm (PSO). Despite presenting outstanding beamforming performance, this approach concerns an evolutionary beamforming method, which is highly time-consuming and therefore not applicable for real-time ABF.

There is a big difference in terms of complexity between null-steering ABF on two dimensions (as in the case of ULAs) and 3D space. For instance, we previously mentioned how our work further decreases the SINR degradation presented in [24]. However, this comparison may not be entirely compatible as this work concerned ULAs and we are dealing with the much more demanding UPA scenario. Several works have also demonstrated angular accuracy on a ULA beamforming scenario [16], [22]. In [22] an angular divergence of 0.6 and 0.25 degrees is achieved in main lobe and null placement respectively, with a FFNN applied on a ULA for an incoming SNR of 10dB. Once again, these results concern the case of ULAs and do not compare with the increased complexity of 3D beamforming. To the best of the authors knowledge, no prior work has demonstrated the accuracy of their proposed beamformer on a UPA, with a statistical analysis on angular divergence.

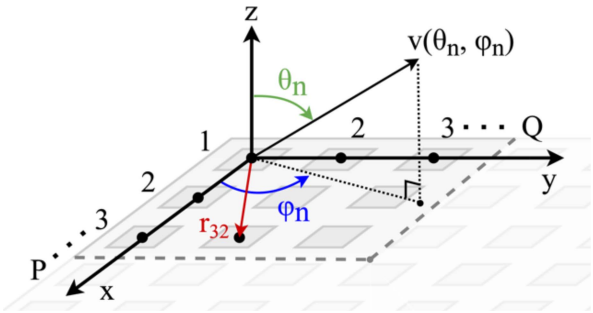

FIGURE 2. Planar antenna array geometry.

From identifying nulls to the actual mapping task assigned to the beamformers and the NNs, the complexity increases dramatically and demands a thorough examination both in how the measurements and dataset production will be realized and in finetuning the hyperparameters of the NN models to establish the most efficient training and inference. Our approach aims to fill in the gaps previously mentioned, attempting to provide a robust and accurate continuous ABF solution based on DL.

III. THEORETICAL NSB ALGORITHM

The theoretical NSB algorithm considers an array, composed by single isotropic point sources. The advantage of using NSB for beamforming comes from its ability to place precise nulls while performing accurate beam-steering [30]. Below, we will go through some fundamental equations of the algorithm to elucidate its operation [3].

Let us consider a $P\times Q$ element UPA, where P is the number of elements on the x-axis and Q is the number of elements on the y-axis. Each element can be represented by the coordinates (p, q) ($p = 1, \ldots, P$ and $q = 1, \ldots, Q$). If I_{pq} is the current used to feed the (p, q) array element, then the UPA radiation pattern is expressed by the array factor as follows:

$$AF(\theta_n, \varphi_n) = \sum_{p=1}^{P}\sum_{q=1}^{Q} I_{pq} \exp\left(j\beta \mathbf{r}_{pq} \cdot \mathbf{v}(\theta_n, \varphi_n)\right) \qquad (1)$$

where β is the propagation phase constant ($\beta = 2\pi/\lambda$ with λ being the wavelength of the operating frequency), $\vec{r}_{p,q}$ is the position vector of the (p, q) array element (its expression depends on the array geometry), and $v(\theta_n,\ \varphi_n)$ is the unit vector towards direction of an incoming signal ($\theta_n,\ \varphi_n$) as shown in Fig. 2.

When the array is in reception mode, the currents $I_{p,q}$ become multipliers of the input signals on each element. Consequently, we may consider that $I_{pq} = w^*_{pq}$, where w_{pq} is a complex weight that expresses the conjugate value of the current at the input port of the (p, q) element. If $N+1$ is the number of incoming signals, and ($\theta_n,\ \varphi_n$) are the AoAs of the *n-th* signal, the theoretical steering vector for each incoming AoA is calculated as:

$$\mathbf{a}(\theta_n,\ \varphi_n) = \begin{bmatrix} \exp(j\beta \mathbf{r}_{11} \cdot \mathbf{v}(\theta_n,\ \varphi_n)) \\ \exp(j\beta \mathbf{r}_{12} \cdot \mathbf{v}(\theta_n,\ \varphi_n)) \\ \vdots \\ \exp\big(j\beta \mathbf{r}_{PQ} \cdot \mathbf{v}(\theta_n,\ \varphi_n)\big) \end{bmatrix} \tag{2}$$

Using (2), the array factor of (1) can be re-written as:

$$AF(\theta_n, \varphi_n) = \mathbf{w}^{\mathrm{H}} \mathbf{a}(\theta_n,\ \varphi_n) \tag{3}$$

where

$$\mathbf{w} = \begin{bmatrix} w_{11} & w_{12} & \dots & w_{PQ} \end{bmatrix}^{\mathbf{T}} \tag{4}$$

is the excitation weight vector and superscript H denotes the Hermitian transpose operation. The total steering matrix $\mathbf{A}$ contains the steering vectors that correspond to the AoAs of all incoming signals as shown below:

$$\mathbf{A} = [\mathbf{a}(\theta_0, \varphi_0),\ \mathbf{a}(\theta_1, \varphi_1), \dots, \mathbf{a}(\theta_N, \varphi_N)], \tag{5}$$

In the same way, another matrix can be defined in the form:

$$\mathbf{A_i} = [\mathbf{a}(\theta_1, \varphi_1), \dots, \mathbf{a}(\theta_N, \varphi_N)]. \tag{6}$$

This matrix is called interference steering matrix because it is comprised by the steering vectors of the interfering incoming signals.

Thus, the beamforming weight vector produced by the NSB beamformer is calculated using the array steering matrix:

$$\mathbf{w}_{\mathrm{NSB}} = \mathbf{A}\Big(\mathbf{A}^H \mathbf{A}\Big)^{-1} \boldsymbol{\upsilon}_1, \tag{7}$$

where

$$\boldsymbol{\upsilon}_1 = [1\ 0\ \dots\ 0]^T \tag{8}$$

is a unit vector of size $N+1$.

IV. PROBLEM DEFINITION AND DATA PREPARATION

To begin with, an 8×8 UPA with its 64 elements spaced at $d=\lambda/2$ is studied. Therefore, to control the feeding of each of the 64 elements of the antenna, 128 values (64 real and 64 imaginary parts) are needed. The DOAs of incoming signals $s_n(n = 0,\ 1, \dots, N)$ are identified by their respective AoAs θ_n and $\varphi_n(n = 0,\ 1, \dots, N)$. We expect that the UPA will operate within the angular sectors [0°, 60°] for elevation and [0°, 360°] for azimuth rotation (Fig. 3). The choice of these angular sectors was based on typical operational scenarios for UPAs. These ranges were selected to cover a comprehensive field of view, ensuring that the beamforming system can effectively handle signals coming from various directions, which is crucial in urban environments and complex topologies. All incoming signals are considered to have a minimum angular distance δ (also known as angular separation) equal to 6°. Thus, for any two incoming signals n_1 and n_2 coming at respective AoAs $(\theta_{n_1}, \varphi_{n_1})$ and $(\theta_{n_2},\ \varphi_{n_2})$, δ is calculated as:

$$\delta = \cos^{-1}\big(\sin\theta_{n_1}\cos\varphi_{n_1}\sin\theta_{n_2}\cos\varphi_{n_2} + \sin\theta_{n_1}\sin\varphi_{n_1}\sin\theta_{n_2}\sin\varphi_{n_2} + \cos\theta_{n_1}\cos\theta_{n_2}\big). \tag{9}$$

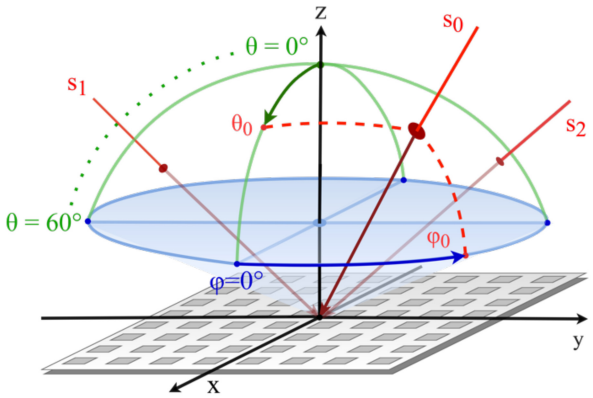

FIGURE 3. Example of 3 incoming signals where (θ_0, φ_0) correspond to the AoA of s_0. We consider the antenna operates for $\theta \in [0°, 60°]$ and $\varphi \in [0°, 360°]$.

Considering low δ values increases the difficulty of the beamforming problem since δ defines the possible distance between two adjacent nulls or between a null and the main lobe. In [3], different values of $\Delta\theta$ and $\Delta\varphi$ were tested for the same task of zero-forcing beamforming with NSB on an 8×8 UPA. The results of this work demonstrate how the decrease of minimum angular distance between main lobe and null direction degrades the main lobe accuracy of NSB. Naturally, steering nulls closer to the direction of the main lobe, can drastically affect its shape and may even shift its peak, which causes this accuracy degradation. Moreover, given the number of elements on this UPA there is also a physical limitation, as the beamwidth of an antenna array is inversely proportional to the number of elements and the array size [2]. Steering null towards interference directions near the direction of the desired user without interfering with the main lobe formation, can only be achieved when the beamwidth is significantly decreased. Thus, maintaining the desired performance while decreasing the angular separation below 6°, would necessitate increasing the physical dimensions of the antenna array. In the context of our work, a value of 6° strikes a balance between making the beamforming task for the NNs more demanding and being closer to a real-world scenario. In addition, the signal-to-noise ratio (SNR) of incoming signals is equal to 0 dB, considering high noise conditions but more variations are examined in Section VII.

To produce the initial training dataset for the NNs, we start with the simple case of one incoming SoI and two SoAs. This is beneficial for two reasons. First, the mapping task assigned to the NNs is easier, allowing us to evaluate their performance on a simple scenario. Second, the dataset produced will be smaller and NN training will be faster. This allows for more experimentation especially for the hyperparameter tuning part where multiple NN trainings are needed.

This research primarily examines how well-tuned NNs perform in associating AoAs with optimal weight vectors for theoretical zero-forcing ABF. However, to mimic real-world challenges, the study also incorporates specific factors.

These factors include the complexity added by the minimal angular separation between adjacent signals, which intensifies the beamforming challenge, and the presence of increased interference. Given the problem setup, we deem these considerations enough to demonstrate how NNs behave at an initial theoretical stage. For future exploration, we plan to generate a dataset simulating user movement within a coverage area. This will enable us to assess the NNs' effectiveness in a dynamic, path-tracking scenario.

Using the NSB algorithm explained in Section III, we produced a dataset of 5×10^6 records for the final NN training and a different one of 1.1×10^4 records to be used for the hyperparameter-tuning part. During dataset production, we put some constraints so that only the most accurate NSB results are kept. Specifically, we only store the record if NSB achieves main lobe and null placement accuracy with an error of <1° and <0.1° respectively. The dataset production showed that only 4% of the produced records were discarded, which further validates the beamforming abilities of NSB. The decision to discard 4% of records due to not meeting the accuracy standards we set does not significantly impact the robustness or generalization ability of our trained models. The excluded records constituted a minor fraction of the dataset and did not indicate any notable limitations or shortcomings of NSB in specific conditions. Their exclusion helps in preventing the model from learning from atypical or erroneous data, which could otherwise compromise its performance in standard scenarios. Our dataset, despite the exclusion of certain records, encompasses a wide range of scenarios within the operating angular sectors. It ensures that the neural networks are trained on high-quality, accurate data, leading to better performance which is later verified by the results in Section VI.

Every record of the dataset consists of $2\times(N+1)$ inputs (pairs of (θ_n, φ_n) for each AoA) and $2\times P\times Q$ outputs (the real and imaginary parts of the complex feeding weights produced by NSB). Considering the operating angular sectors mentioned earlier and the weight values that range between $[-1, 1]$, we normalize both input and output to $[0, 1]$. Most activation functions used in NNs (like sigmoid or tanh) are designed to work best with values in the range of $[0, 1]$ or $[-1, 1]$. Normalizing data to $[0, 1]$ ensures that the inputs to these activation functions are within their optimal operating range, leading to better performance of the network [31]. In addition, optimizers like Adam, utilized in our NN training greatly benefit from this normalization as normalized data lead to a more balanced gradient distribution across all input features, making Adam more effective [32].

V. SELECTION OF NEURAL NETWORK ARCHITECTURES

A. ARCHITECTURES OF NNS AND CRITERIA

In this work three different NN types are implemented. Each of these variations will be assigned with the task of mapping the incoming AoAs to proper feeding weights. The first one is the FFNN, otherwise known as the multilayer perceptron

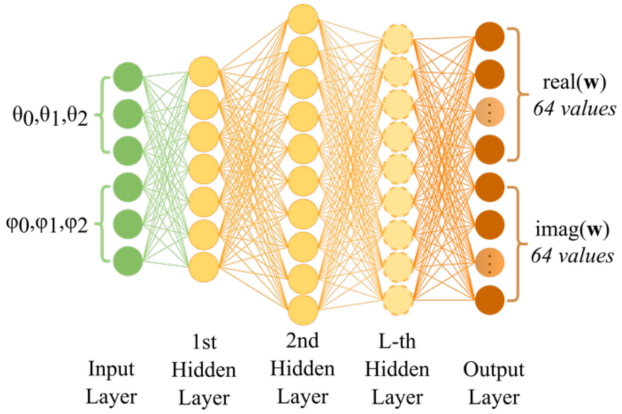

FIGURE 4. Feedforward implementation for the case of 1 SoI and 2 SoAs.

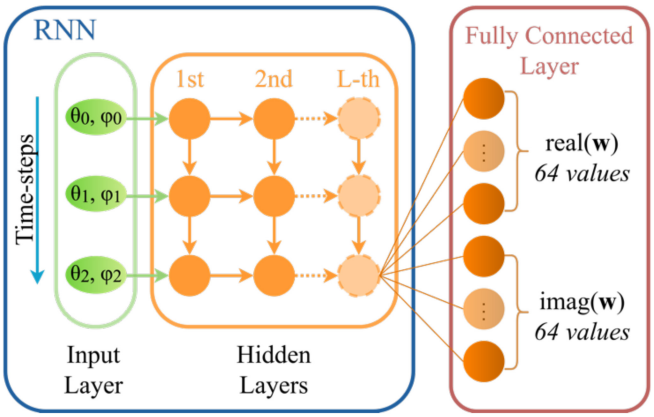

FIGURE 5. RNN implementation for the case of 1 SoI and 2 SoAs.

(shown in Fig. 4). In this case, the pairs of angles of all incoming signals enter simultaneously on the input layer of the NN. After being processed by the hidden layers, an output vector of 128 values is generated.

On the other hand, we have the recurrent neural network (RNN) configuration (Fig. 5). Here, the pair of angles of the corresponding incoming signals enter the RNN consecutively at different time-steps. This way, there is a progressive adaptation of the output of the RNN according to each new input. The processing units of each hidden layer (depicted with light orange color in Fig. 5), will be either GRUs or LSTM units. Both implementations will be tested to decide upon the most efficient configuration. The mapping task is considered as a "many-to-one" approach meaning that AoAs are fed to the RNN one after another, and only the output of the final time-step is kept. Since the inner hidden state of these processing units may differ from the desired $2\times P\times Q$ size of the output, we use an additional, trainable, fully connected layer, to transform the RNN hidden state information into a 128-value weight vector.

To train each NN, we use the root mean square error (RMSE) as cost function due to its effectiveness in capturing the average magnitude of errors in the NN predictions. RMSE is particularly suitable for our application as it

penalizes larger errors more severely, which is crucial in beamforming where precision in weight computation estimation is key. However, a different criterion is used to see how different hyperparameter configurations compare. This criterion consists of two coefficients. The first one represents the average validation error of the last 10 epochs of training, and the second calculates the average gap between training and validation error in these last 10 epochs. Thus, every configuration is evaluated both on their predictive performance by measuring the validation error, and on their overfitting risk by measuring the indication of overfitting. Since the first one is more important than the latter, we multiply the second coefficient by a factor of 0.1. This is a tunable value, but we found that 0.1 suits the needs of this evaluation better. This way, between the tunings that reach similar validation errors, we promote those that have a lower overfitting risk. As explained in [33], the dropout rate can create an unusual mismatch between training and validation error. With high dropout rates, the training error on a given epoch can be much higher than the validation error. To avoid any bias in the second coefficient (when subtracting training error from validation error gives a negative value), we use the *max* function to only keep positive values as shown below. Finally, if K is the total number of training epochs, the criterion is calculated as:

$$\text{criterion} = \frac{\sum_{\text{epoch}=K-10}^{K} \text{validation error}_{\text{epoch}}}{10} + 0.1 \cdot \frac{\sum_{\text{epoch}=K-10}^{K} \max\left(\text{validation error}_{\text{epoch}} - \text{training error}_{\text{epoch}}, 0\right)}{10} \tag{10}$$

B. HYPERPARAMETER OPTIMIZATION

As stated above, our ultimate goal is not only to identify an accurate beamforming alternative in NNs, but also to have the least response time possible. The temporal response of NNs depends on their inner complexity. This in turn, corresponds to the number of operations a single prediction step requires. As information travels from input layer to output, the number of operations is contingent upon the number of neurons on each of the hidden layers of the NN. Therefore, the prediction time of a NN is proportionate to its inner size. For this reason, we insisted upon finding the most efficient design-related tuning. Hence, we employed three different methods of hyperparameter optimization. Our intent was not only to find the most promising NN tuning, but also to compare these algorithms and contribute a comparison on their performance on hyperparameter tuning. The methods we compare are the grid search (GS), Bayesian optimization (BO) and the genetic algorithm (GA). The selection of these techniques was based on their complementary strengths. GS offers a thorough exploration of a predefined parameter space at the cost of not exploring possible solutions not included in this grid while for some cases it examines ineffective and unproductive parameter combinations that waste valuable search time. An additional limitation of GS is scalability. As the number of hyperparameters increases, the time and computational resources needed grow exponentially. GA on the other hand performs a much more effective and extensive exploration of the parameter space focusing on the most prominent solutions. GA is particularly effective in scenarios where the relationship between hyperparameters and the objective function is not well understood or is highly nonlinear. Finally, BO is particularly advantageous when each evaluation of the objective function (such as training a complex neural network) is time-consuming and resource intensive. It's about finding the best solution with the least number of evaluations. To sum up, the chosen methods cover a broad spectrum of optimization strategies and are considered to suffice for the purposes of this work. A similar comparison can be found in [34] where PSO prevails over GA for this type of optimization but a comparison with BO was considered a better contribution. All tunable hyperparameters are shown in Fig. 6. By color-coding each box, we signify which NN type it optimizes. We identify two stages of optimization.

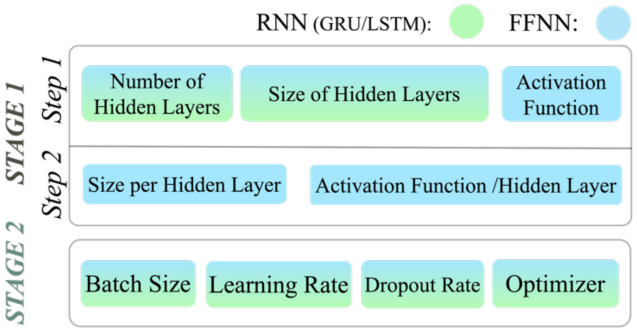

FIGURE 6. Optimization Stages Explained (the colors indicate which type of NN the hyperparameter is targeted to tune).

At the first stage we tune each design-related hyperparameter considering an initial learning rate and batch size equal to 0.001 and 256 respectively. In the case of FFNN we can fine tune each hidden layer size and its corresponding activation function separately. However, both LSTM and GRU-RNNs have a hidden state of fixed size that is exchanged between the processing units of the RNN. Therefore, Stage 1 is split into two steps. On the first step, we optimize the number of hidden layers, while keeping a constant layer size (and a constant activation function for the case of the FFNN). While on the second step, we focus solely on the FFNN, tuning the size and the activation function of each hidden layer separately. All optimization techniques participate in this stage.

On the second stage, we optimize only the training-related hyperparameters with the rest of the design-related hyperparameters fixed. Since these should not affect the NN design, we consider a good practice to tune them separately. Since we are primarily interested in the proper selection of Stage 1 parameters and given the small search space of Stage 2 parameters (described in Table 1), Stage 2 can be covered entirely with GS.

TABLE 1. Grid search parameter selection.

Hyperparameters	Possible Picks
Number of Layers	[2, 3, 4, 5]
Hidden Layer Size	[128, 256, 512, 1024]
Activation Function	[ReLU, tanh, sigmoid]
Batch Size	[64, 128, 256, 512]
Learning Rate	[0.01, 0.005, 0.001]
Dropout Rate	[0, 0.2, 0.4]
Optimizer	[Adam, Adamax, Adagrad]

TABLE 2. Stage I/ Step I of GS optimization.

Model	No. of Layers	Hidden Layer Size	Act. Fun.	Search Time (hrs)	Criterion
FFNN	5	1024	tanh	15.5	0.0407
GRU	5	512	-	6.2	0.0355
LSTM	5	512	-	6.4	0.0356

TABLE 3. Stage I/ Step II of GS optimization.

Model	Size / Hidden Layer	Activation Function / Hidden Layer	Criterion
FFNN	256,512,256,1024,1024	tanh, tanh, ReLU, ReLU, tanh	0.0395

Regarding tuning all hyperparameters in parallel, it has to be noted that a combination of small batch size with an increased number of hidden layers or hidden layer sizes may induce unnecessarily long training times. Thereby, this two-stage separation aims to improve the overall optimization efficiency and avoid timewasters. Lastly, on each NN-training a learning rate scheduler, early stopping and a 3-fold cross validation is applied to improve training time and further increase the validity of the findings.

1) GRID SEARCH (GS)

With GS, a list of possible values for each hyperparameter is predefined and then the NN models train using all possible combinations of these values. Despite this method being one of the most popular choices for hyperparameter tuning, GS limits the search space into a predefined grid, excluding configurations that could potentially provide better solutions. Furthermore, grid search scales exponentially in terms of search time, as the number of hyperparameters increases [35]. The distinct values selected according to the best exploration-exploitation tradeoff are shown in Table 1.

Observing the designs proposed by GS (Tables 2 and 3) we can easily distinguish the RNN architectures as the ones with the lowest criterion value. The increased search time for the FFNN case is due to the extra hyperparameter it needs to consider (activation function).

Having identified the proposed designs for each NN type, we can now proceed to Stage II of the optimization. After running grid search for Stage II, we observe that the Adam optimizer and a dropout rate of 0% gives the best results for all NN types so from now on these parameters are fixed. This

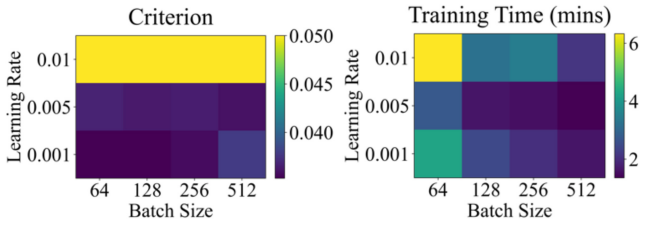

FIGURE 7. Grid search results for LSTM-RNN, using Adam as the optimization method (on the left we see the criterion values and on the right the training time of each combination respectively).

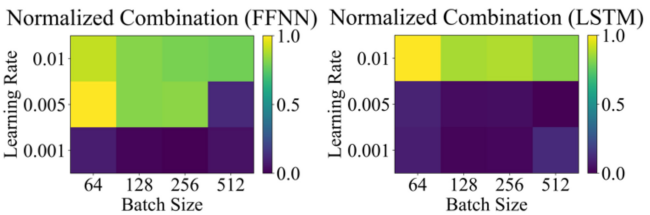

FIGURE 8. Indicative normalized combination of the grid search results for criterion and training time for the case of FFNN and LSTM-RNN.

TABLE 4. Stage II of GS optimization.

Model	Batch Size	Learning Rate	Dropout Rate	Optimizer	Search Time (hrs)	Criterion
FFNN	256	0.001	0.0	Adam	5.4	0.0393
GRU	256	0.001	0.0	Adam	15.1	0.0352
LSTM	512	0.005	0.0	Adam	15.6	0.0352

leaves us with the choice of different learning rate and batch size configurations. At this phase, we are not only interested in reaching a low criterion value, but also to train the NN in the shortest time possible. Fig. 7 shows an example of the impact of each configuration, either on the criterion value or on the training time of the LSTM-RNN model. We observe that even though some combinations obtain a low criterion value, they lead to slower training (e.g., learning rate equal to 0.001 and batch size equal to 64). Thereby, we decided to select the best configuration as the one that minimizes the following expression:

$$0.9 \cdot \text{criterion}_{normalized} + 0.1 \cdot \text{training time}_{normalized} \quad (11)$$

where subscript *normalized* indicates that each part has been normalized to [0,1] according to the corresponding max and min values reached during GS. The selection of the multipliers is such that prioritizes the criterion value over training time but promotes the tunings that achieve faster training with the lowest criterion value guaranteed. Given the results of (11) (indicative values in Fig. 8) we extract the final values proposed by grid search. These are shown in Table 4.

2) BAYESIAN OPTIMIZATION (BO)

BO is another preferred and commonly used technique [36], which has proven to be a great tool for NN hyperparameter

TABLE 5. Bayesian optimization search space.

Hyperparameters	Possible Picks
Number of Layers	2:5
Hidden Layer Size	128:16:1024
Activation Function	[ReLU, tanh, sigmoid]

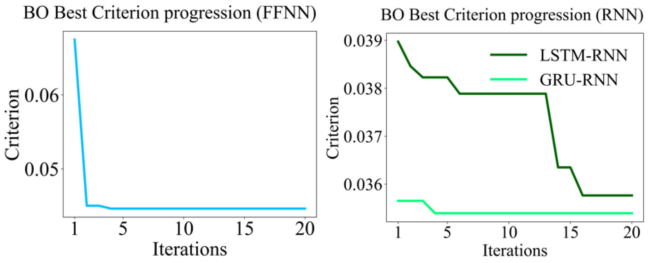

FIGURE 9. Best criterion value BO found over 20 iterations either for the case of FFNN or the GRU/LSTM-RNN.

TABLE 6. Stage I/ Step I of BO optimization.

Model	No. of Layers	Hidden Layer Size	Act. Fun.	Search Time (hrs)	Criterion
FFNN	4	528	tanh	2.6	0.0445
GRU	4	656	-	2.7	0.0354
LSTM	4	464	-	2.9	0.0357

TABLE 7. Stage I/ Step II of BO optimization.

Model	Size / Hidden Layer	Act. Fun. / Hidden Layer	Search Time (hrs)	Criterion
FFNN	496,416,176,992	tanh, ReLU, tanh, tanh	3.3	0.0402

tuning [37], [38], [39]. It refers to a sequential approximation of the maxima of a computationally expensive black-box function by strategically sampling its domain. The target of BO is to systematically search the function domain for a point $\mathbf{x}^* \in X$ that attains the global maximum value f^* by probing f as few times as possible. In this work we implemented BO using the BoTorch framework [40]. In our implementation, we begin with 5 random initial samples, and have BO approximate the global maxima within 20 iterations. Specifically, as one can later observe in Fig. 9 and Table 6, this combination can produce decent results in a very short search time. Increasing or decreasing these values could in turn increase the total search time of BO or prevent BO from reaching such results. Table 5 explains the different values examined with BO for each hyperparameter.

Comparing the results presented in Tables 6 and 7 with those produced by GS we can already observe that there is a significant drop in search time, with BO being twice as fast in the case of RNN tuning and nearly six times faster in the case of FFNN. In addition, the criterion values that BO reached are very similar to those of GS. Moreover, BO managed to reach these criterion levels with NN designs of much smaller size.

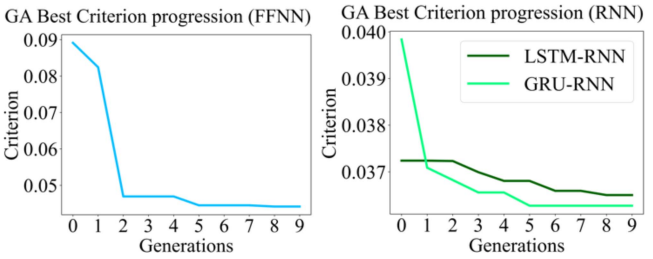

FIGURE 10. Best criterion value GA found over 10 generations for the case of FFNN or the GRU/LSTM-RNN.

TABLE 8. Stage I/ Step I of GA optimization.

Model	No. of Layers	Hidden Layer Size	Act. Fun.	Search Time (hrs)	Criterion
FFNN	5	432	tanh	3.3	0.0442
GRU	5	816	-	3.4	0.0363
LSTM	4	880	-	4.2	0.0365

TABLE 9. Stage I/ Step II of GA optimization.

Model	Size / Hidden Layer	Act. Fun. / Hidden Layer	Criterion
FFNN	336,480,416,208,496	tanh, tanh, sigmoid, tanh, tanh	0.0433

3) GENETIC ALGORITHM (GA)

A GA is an evolutionary method of solving complex optimization problems based on principals of natural selection and genetics [41]. A set of possible randomly selected sets of solutions (strings of bits, referred to as "chromosomes") is initially generated. This initial "population" represents the first "generation" of solutions which is then evaluated based on how well they perform according to a fitness function. The best-performing solutions are selected to "reproduce" and create a new generation of solutions. From the size of the initial population to the mutation probability, there are different GA parameters that have to be decided upon, before the process begins.

For our purposes we set the population size to 5, the crossover probability to 0.9, the mutation probability to 0.1, the gene length to 6 and we run the optimization for 10 generations. Just as in the case of BO these values were selected to serve the purpose of balancing exploration and exploitation of the search space to our benefit. To further elaborate, since the fitness function used in this implementation is computationally demanding, it is crucial that we spare any redundant calculations if a low criterion value can be reached within a shorter amount of time.

As Fig. 10 and Tables 8 and 9 indicate, this GA parameter configuration allows GA to reach criterion values similar to GS and at a lower search time. The proposed designs are similar to the GS tuning, at least for the cases of FFNN and GRU-RNN. However, the recommended hidden layer sizes are increased both for GRU and LSTM-RNNs. We observe that despite GAs being more preferable than GS because of

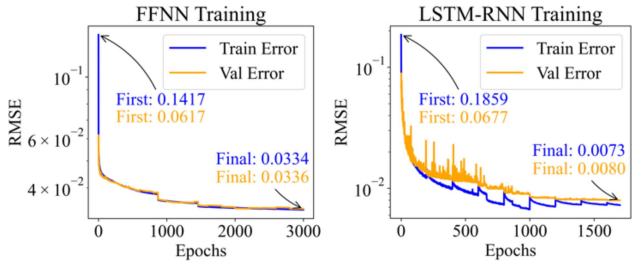

FIGURE 11. Indicative training progression of the FFNN and LSTM-RNN.

their search time, they do not reach neither the advantageous NN designs and criterion values of BO nor the search time of BO.

Given all previously presented results, the most promising settings are those proposed by BO, balancing both efficient NN designs and adequate criterion values. There is one more important detail worth noticing. Because of the 3-fold cross validation, GS requires three times all possible hyperparameter combinations, therefore $3 \times (4 \times 4 \times 3) = 144$ trainings. BO needs to train the NN three times per each probing attempt, thus $3 \times (5 + 20) = 75$ trainings. Lastly, GA requires three times the number of the population for each of the generations which gives $3 \times (5 \times 10) = 150$ trainings. Interestingly, while GS seemingly requires a smaller amount of NN trainings than GA, GS does not avoid extreme hyperparameter combinations that lead to unnecessarily time-consuming NN trainings and therefore, longer optimization duration.

C. TRAINING THE BEST CONFIGURATIONS

After comparing all proposed NN designs we come down to the ones that performed best, proposed by BO. As mentioned in Section IV, we have a dataset of 5 million records at our disposal. However, since the total size of this dataset is approximately 15GB, we split it into smaller sub-sets of 1million records. This prevents memory overload. The subsets are utilized according to the round robin principle, where each subset is used to train the NN model for a fixed number of epochs. The switch between subsets can be observed in Fig. 11 where small reoccurring spikes appear on the training error curve. After examining different values for the number of epochs we decided that 200 epochs are enough to exploit each subset without causing overfitting. Once again, we employ the learning rate scheduler. When a plateau appears on the validation error, the learning rate is reduced, which can improve training performance by a factor of 2 to 10 since it can help gradient descent-based algorithms escape sharp local minima [42]. We set the patience of the scheduler to 40 epochs, and we reduce learning rate by a factor of 0.5 each time a plateau appears. These numbers were selected to prevent the scheduler from reducing the learning rate prematurely without stalling its intervention unnecessarily. These learning rate reductions can be observed in Fig. 11 where small drops both in training and validation error can be noticed.

To save excess computational time, we set multiple termination criteria to control the training progress. Namely, training will stop if one of the following conditions is true:

1) Learning rate has decreased to the point where it no longer contributes significantly to the training process (1×10^{-6}).
2) Overfitting occurs for the last 50 epochs. We tolerate a small deviation between training and validation error as long as validation error keeps decreasing.
3) Training reaches 3000 epochs. We have observed that training beyond this number of epochs results in insignificantly better models.
4) Validation error does not progress for 60 consecutive epochs. This is to make sure that the training tools have been utilized before we proceed to an early stopping.

The FFNN trained for 10.5 hours and reached a validation RMSE of 0.033, while the GRU training lasted for 7.3 hours and the LSTM for 6.4 hours reaching a value of 0.0077 and 0.0073 correspondingly. All models were trained on an NVIDIA A100-SXM4-40GB GPU.

VI. EVALUATION OF THE BEAMFORMERS

A. PERFORMANCE EVALUATION

To evaluate the performance of each beamformer, we need to establish the metrics we intend to examine. First, each NN beamformer should be able to reach SINR levels similar to those of the NSB algorithm. The SINR is defined as

$$SINR = 10 \log_{10} \frac{P_{desired}}{P_{interference} + P_{noise}} \tag{12}$$

where $P_{desired}$ is the output power towards the desired user, $P_{interference}$ is the sum of power towards interfering signals and P_{noise} is the noise power dictated by the input SNR. High SINR levels are a great indication of good beamforming performance as they guarantee both the directivity and the adaptability required from a beamformer. Since high SINR values are achieved by rejecting the incoming interfering signals, while accurately positioning the main lobe, we considered that measuring the accuracy of main lobe and null placement is necessary. In addition, we are aware that the NSB algorithm is able to create very "deep" and accurate nulls towards the directions of SoAs therefore it is expected that the NN-beamformers will be inferior both in terms of accuracy and signal rejection (the lower the array factor, the higher the rejection). That is why we measure the array factor at two different directions. First, towards the direction the beamformer placed a null at, which we call *Predicted Null* direction, and second, towards the true direction of the incoming SoAs, where a null was expected to be placed at, which is mentioned as *Expected Null* direction (as shown in Figs. 15 and 16). Finally, since this implementation aims to improve beamforming latency, we measure the time each beamformer takes to produce the feeding weights given the AoAs as input.

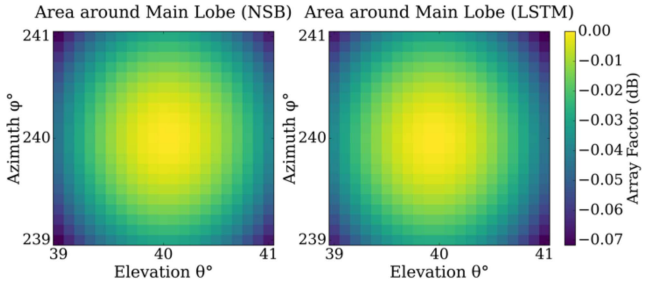

FIGURE 12. NSB-produced (left) and LSTM-RNN-produced (right) array factor for an area of 1 degree around the expected Main Lobe location.

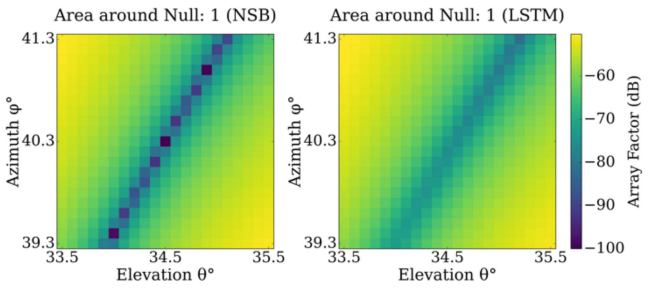

FIGURE 13. NSB-produced (left) and LSTM-RNN-produced (right) array factor for an area of 1 degree around the expected 1st Null location.

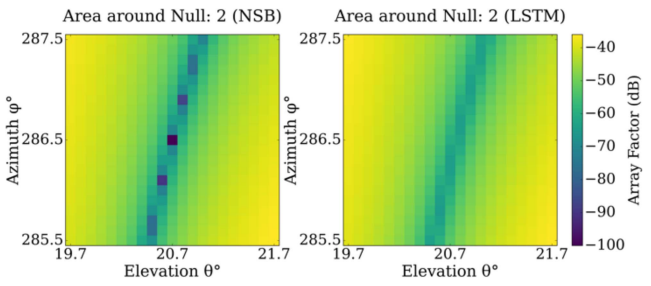

FIGURE 14. NSB-produced (left) and LSTM-RNN-produced (right) array factor for an area of 1 degree around the expected 2nd Null location.

As shown in (3), for the operating range of this UPA, the radiation pattern is composed of the array factor at each spatial direction ([0°, 60°] for elevation and [0°, 360°] for azimuth rotation with a step of 0.1°). Evidently, identifying nulls and their corresponding divergences in a grid of such increased size (3600×600 cells) is not a trivial task. To save computational time, we developed a technique that searches for local minima by focusing on the area of interest, which is around the direction of the incoming SoAs. This decreases the search space and drastically improves search time. After testing on 1×10^4 different scenarios, the average time needed to identify local minima in the entire grid is 0.7 seconds while with our technique this time is reduced to 0.01 seconds (tested on the Apple M1 Max chip). We start by focusing on an area of 1° around the direction of an SoA (Figs. 13-14). If no minima are found, we expand the search territory by a step of 0.1°, until a local minimum is found.

We measure null divergence as the angular distance δ between the expected direction of the null and the direction of the local minimum created closest to it. Measuring the main lobe divergence is a similar task, finding the angular distance between the direction of the SoI and the local maxima closest to that. Again, this is done by searching only a small interest area around the main lobe (Fig. 12) with the potential of expanding it if no local maxima are found. Examples of this approach are shown in the following figures where we start by depicting the interest areas around the 3 main points of interest (the main lobe and 2 null locations) and proceed to show the radiation pattern for the entire operational space. All figures depict the normalized array factor in a colormap plot, where the x-axis represents the elevation rotation and the y-axis the azimuth rotation. This representation is preferred from a 3D surface plot because it helps us distinguish the location of nulls easier. In each figure we see the produced array factor of either the NSB algorithm or the LSTM-RNN beamformer, concerning the same randomly generated beamforming scenario. Specifically, we have the SoI arriving at (40°, 240°) and 2 incoming SoAs at (34.5°, 40.3°) and (20.7°, 286.5°) respectively.

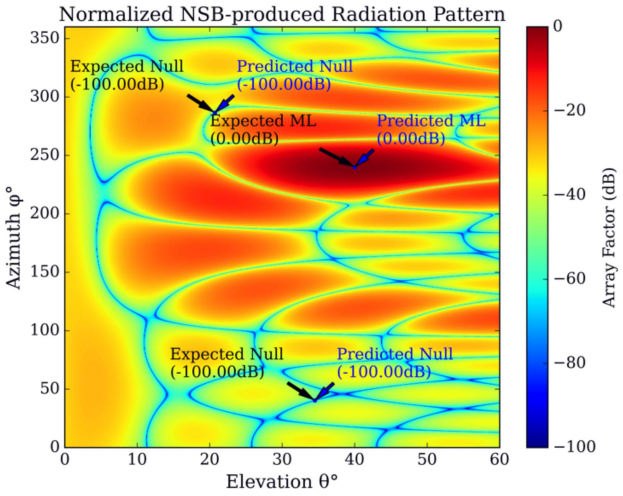

FIGURE 15. Normalized NSB-produced radiation pattern for the case of 1 SoI at (40, 240) and 2 SoAs at (34.5, 40.3) and (20.7, 286.5) respectively.

In the following figures we showcase the radiation pattern produced using the array factor of (3) using both the weights produced by NSB and the LSTM-RNN (Figs. 15 and 16 respectively). The scale of the elevation and azimuth axes is deliberately disproportional to improve presentability. In Fig. 15 we observe how in the case of the NSB algorithm the Expected and the Predicted Nulls are identical, having the same array factor, which further demonstrates the unparallel null-placement accuracy of NSB.

In Fig. 16 we can observe how the NN-based beamformer creates an overall similar radiation pattern but with a slight mismatch at the placement of nulls, with small divergences from the expected values. However, the array factor towards the expected location of the null is low enough to consider the signal rejected. For both figures, the scale of array factor was purposefully limited within the [−100dB, 0dB] sector since values lower than −100dB are considered frill to display.

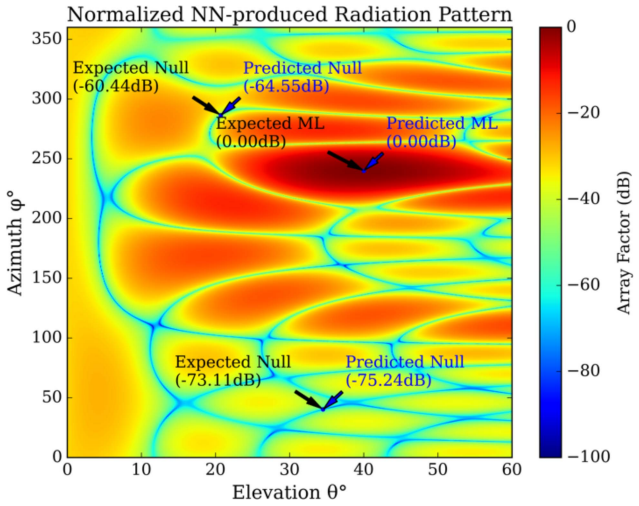

FIGURE 16. Normalized LSTM-RNN-produced radiation pattern for the case of 1 SoI at (40, 240) and 2 SoAs at (34.5, 40.3) and (20.7, 286.5) respectively.

It has to be noted that both the NSB and the NN-beamformer present excellent main lobe placement abilities since the expected and the predicted locations are identical for this scenario. Nevertheless, a more in-depth analysis must take place in order to accurately measure the performance of each beamformer, where more than one scenario is considered.

B. PERFORMANCE ANALYSIS

To analyze the performance of each model, we perform a statistical analysis using 1×10^4 new scenarios that none of the NN beamformers has prior "experience" on. For each scenario of random incoming AoAs, we employ both the NSB algorithm and the pre-trained NN-beamformers to perform beam-steering and null-steering by producing the feeding weight vector. Next, two radiation patterns are produced using (1), where each current I_{pq} takes its value from the corresponding weight of the produced beamforming vectors. For each pattern, we identify the direction the main lobe and the directions of nulls by obtaining their elevation and azimuth angle. As explained in Par. A, we designate a specific area around the expected nulls (and the main lobe) in which we attempt to search for local minima (maxima). The span of these areas is 1° (10 matrix cells) and if no minima (or maxima) are found we expand this area by 0.1° (1 matrix cell) until the null (or main lobe) is located. This way we can calculate the divergence between the directions of the desired and undesired AoAs using the angular distance as shown in (9). Using the produced weights, we can calculate the signal-to-interference ratio (SIR) with the following:

$$SIR = 10log_{10}\frac{w^H\ \mathbf{a}(\theta_0,\varphi_0)\ \mathbf{a}(\theta_0,\varphi_0)^H\ w}{w^H\ \mathbf{A}_i\ \mathbf{A}_i^H\ w} \tag{13}$$

where w are the produced weights, $\mathbf{a}$ is the steering vector as shown in (2), $\mathbf{A}_i$ is the interference steering matrix shown in (6), and $n = 0,\ 1, \ldots, N$ is the index of incoming signals with the first one being the SoI and the rest ($n = \ 1, \ldots, N$) the SoAs. To get the SINR we need to add the noise power to the denominator of SIR as in (11). Considering the input signal power equal to one, and the input SNR in dB, the noise power at the input can be derived from:

$$P_{noise} = 10^{-\frac{SNR}{10}} \tag{14}$$

Therefore, SINR is calculated as:

$$SINR = 10log_{10}\frac{w^H\ \mathbf{a}(\theta_0,\varphi_0)\ \mathbf{a}(\theta_0,\varphi_0)^H\ w}{w^H\ \mathbf{A}_i\ \mathbf{A}_i^H\ w + P_{noise}\ w^H\ w} \tag{15}$$

It is evident that the array factor towards the SoI and SoAs is directly correlated to the main lobe and null divergence that was derived earlier. The higher the divergence, the less optimal the array factor is going to be. To demonstrate the impact of beamforming accuracy, we also present the array factor measured at the directions of SoI and SoAs. This gives us a more direct representation of the performance of our beamformers. Finally, we measure the time needed for each beamformer to extract these weight vectors. Considering that for n incoming signals the computational complexity of NSB is $O(n^3)$ while for both FFNN and RNNs it is $O(n)$, we expect that the NN-based beamformers will be much faster than the deterministic NSB algorithm. The mean response time of each beamformer was measured when operating on the same GPU that was used for training (Section V Par. C). This is the time a beamformer needs to produce the beamforming weight vector for the case of 1 SoI and 2 SoAs with the AoAs known. Since in this implementation, DOA estimation is needed for both NSB and NN-based beamforming, the time needed for DOA estimation would be the same for both beamformers and can be omitted when comparing the response times below. The results of the performance of each model are shown in Tables 10-12.

TABLE 10. Statistical analysis of the FFNN model performance.

	Mean	Std
NSB Main Lobe Divergence (°)	0.23	1.42
NN Main Lobe Divergence (°)	0.12	1.08
NSB Null Divergence (°)	0.00	0.00
NN Null Divergence (°)	1.75	2.46
NSB SINR (dB)	17.87	0.57
NN SINR (dB)	17.20	1.66
Array Factor towards SoI [NSB/**NN**] (dB)	-0.09/**-0.04**	0.3/**0.1**
Array Factor towards SoA [NSB/**NN**] (dB)	-130/ **-33.16**	0.0/ **10.6**
Response Time [NSB/**NN**] (ms)	1.14/ **0.24**	-

TABLE 11. Statistical analysis of the GRU-RNN model performance.

	Mean	Std
NSB Main Lobe Divergence (°)	0.25	1.66
NN Main Lobe Divergence (°)	0.24	1.55
NSB Null Divergence (°)	0.00	0.00
NN Null Divergence (°)	0.28	0.97
NSB SINR (dB)	17.87	0.58
NN SINR (dB)	17.80	0.85
Array Factor towards SoI [NSB/**NN**] (dB)	-0.1/**-0.1**	0.3/ **0.4**
Array Factor towards SoA [NSB/**NN**] (dB)	-130/ **-44.40**	0.0/ **9.0**
Response Time [NSB/**NN**] (ms)	1.14/ **0.36**	-

TABLE 12. Statistical analysis of the LSTM-RNN model performance.

	Mean	Std
NSB Main Lobe Divergence (°)	0.25	1.73
NN Main Lobe Divergence (°)	0.24	1.73
NSB Null Divergence (°)	0.00	0.00
NN Null Divergence (°)	0.23	0.74
NSB SINR (dB)	17.87	0.57
NN SINR (dB)	17.83	0.76
Array Factor towards SoI [NSB/**NN**] (dB)	-0.09/**-0.08**	0.3/ **0.3**
Array Factor towards SoA [NSB/**NN**] (dB)	-130/ **-46.24**	0.0/ **9.5**
Response Time [NSB/**NN**] (ms)	1.14/ **0.40**	-

The FFNN model demonstrates impeccable main lobe placement accuracy with an angular divergence smaller than that of NSB by 0.1 degrees. However, null divergence is increased. This has an impact on the mean SINR levels where a 0.65dB degradation is observed. Despite that, the array factor towards the SoAs is adequate, which guarantees satisfactory signal rejection. In addition, the FFNN model is 5 times faster than NSB as a beamformer.

Both GRU and LSTM based RNN models present an outstanding beamforming performance. These architectures achieve near optimal SINR, insignificantly lower than the NSB algorithm. Similarly, these RNN beamformers can guarantee an array factor of lower than -30dB towards unwanted signals with unprecedented main lobe and null placement accuracy. Out of the two, the GRU-RNN model is faster, being nearly 4 times faster than NSB while the LSTM is at least 3 times faster. These results are more than satisfactory as they indicate that these RNN structures are well designed and able to perform fast and accurate beamforming for a wide variety of scenarios. Since the accuracy of the LSTM model is slightly higher without sacrificing much of the response time, we choose to continue with this as the proposed model. In the following section, the robustness of this model in various noise and interference scenarios is presented.

C. COMPARISON WITH CODEBOOK APPROACH

Let us assume that a codebook contains k predefined beam patterns. The complexity of the beamforming process is primarily dictated by how these patterns are searched and selected. In the simplest case, if a linear search is used to find the best beam pattern, the complexity would be O(k), where you potentially have to compare each beam pattern with the current channel conditions or user requirements. If the codebook is structured or indexed in a way that allows for more efficient searching, the complexity can be lower [43]. However, in systems with large codebooks the complexity can become significant as the number of weight vectors stored in the codebook is increased. To accommodate all conceivable scenarios determined by the operational angular sectors and an angular accuracy of our study (as detailed in Section IV), a codebook would require an immense amount of storage space. This size escalates exponentially with the inclusion of additional interferences and when considering antennas with a greater number of elements. The subsequent closed-form expression provides an estimated calculation of the required storage size in Terabytes:

$$Total\ Storage\ (TB) \approx \frac{\left(\frac{60}{s} \times \frac{360}{s}\right)^n \times 2PQ \times b}{1024^4} \quad (16)$$

where s is the step size in degrees, n is the number of incoming AoAs, P and Q correspond to the number of elements on the horizontal and vertical axes of a UPA, and b is equal to 4 (bytes per floating point number). Taking the simplest scenario of $n = 3$ incoming AoAs (1 SoI and 2 SoAs), and $s = 0.3$ degrees (the precision level achieved with the proposed LSTM-based beamformer for 3 AoAs as shown in Table 12), yields 6.4×10^6 TB of storage. Thus, it is worth considering that although the codebook approach may offer lower complexity compared to conventional beamformers and possibly some NN-beamformers, it fails to achieve the same level of precision as the NN-beamformer without requiring an impractically large amount of storage space.

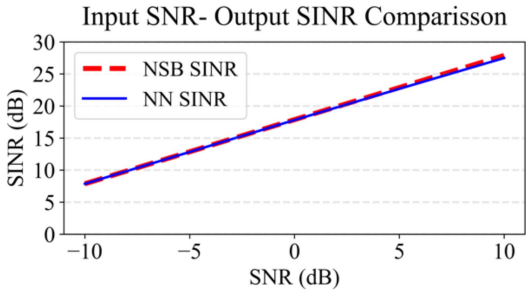

FIGURE 17. Input SNR - output SINR correlation.

VII. ROBUSTNESS ANALYSIS

A. SINR ROBUSTNESS WITH VARYING INPUT SNR

An important performance indicator of beamformers is their ability to maintain high SINR in varying incoming noise conditions. As shown in Fig. 1, the part of the beamforming process we examine is not directly influenced by incoming noise. Specifically, this noise might influence the ability of a DOA estimation algorithm to accurately predict the AoAs of incoming signals. The beamformers we examine simply receive several incoming angles, without direct dependance on noise-carrying information. For this reason, it could be said that our models are noise indifferent. Therefore, a more accurate performance metric for our scenario would be to simply measure the SIR. Evidently, improving the SIR of a radio frequency link, also improves the SINR while P_{noise} remains unaffected. The reason we use the SINR metric is because it is a more commonly used beamforming performance indicator. As shown in Fig. 17, both NSB and the LSTM model are independent of the input noise, and they maintain optimal SINR levels regardless of the incoming SNR.

B. CASES WITH INCREASING SOAS

Up to this point we have examined the case of three incoming signals (1 SoI and 2 SoAs). Having found a promising

TABLE 13. Statistical analysis of the LSTM-RNN model for 11 AoAs.

	Mean	Std
NSB Main Lobe Divergence (°)	1.20	2.26
NN Main Lobe Divergence (°)	1.19	2.24
NSB Null Divergence (°)	0.00	0.00
NN Null Divergence (°)	0.40	1.08
NSB SINR (dB)	16.97	1.54
NN SINR (dB)	16.85	1.69
Array Factor towards SoI [NSB/**NN**] (dB)	-0.51/**-0.50**	0.92/ **0.93**
Array Factor towards SoA [NSB/**NN**] (dB)	-100/ **-37.52**	0.0/ **8.84**
Response Time [NSB/**NN**] (ms)	1.23/ **0.73**	-

NN structure that performs adequately on the problem in hand, we are left with exploring the limitations this method may present. Since producing datasets for multiple scenarios of ranging number of incoming signals is a highly time-consuming process, we deemed it adequate to demonstrate the proposed models' performance for the case of 10 SoAs and 20 SoAs. For these scenarios we consider that the SNR is 0dB.

1) 1 SOI AND 10 SOAS

After training the LSTM-RNN design with a dataset of 3×10^6 records for the case of 1 SoI and 10 SoAs, we perform a statistical analysis on 1×10^4 new records that the NN has not been trained upon and we compare our model with NSB. The results in Table 13 show that once again our model manages to achieve similar main lobe placement abilities with the NSB algorithm while the null placement mean divergence is at a higher but still very promising level. This is also reflected by the 0.12dB degradation on the SINR levels which stands to show how adequately the LSTM-RNN beamformer still performs. Even though the increased time-steps have an impact on the response time of the RNN architecture, it manages to still be twice as fast as NSB. In Fig. 18 we demonstrate indicative radiation pattern produced by the LSTM-RNN for a randomly generated scenario. It is evident that the proposed model manages to accurately place the main lobe while maintaining the ability to adequately decrease the array factor around the areas of expected interference.

2) 1 SOI AND 20 SOAS

Moving on to the case of 20 SoAs, the model is trained once more on a new 3×10^6 record dataset, and we perform a new analysis on 1×10^4 records. The results shown in Table 14 have some interesting insights. First, we notice that the proposed model is becoming better than NSB at the task of steering the main lobe, since the angular divergence becomes less than that of NSB. Despite the increased null placement divergence, it is interesting to observe that the mean SINR level of the NN is now higher than that of NSB. This positive outcome can be explained as a result of how nulls, may affect the main lobe. When many "deep" nulls are placed close to the desired main lobe direction,

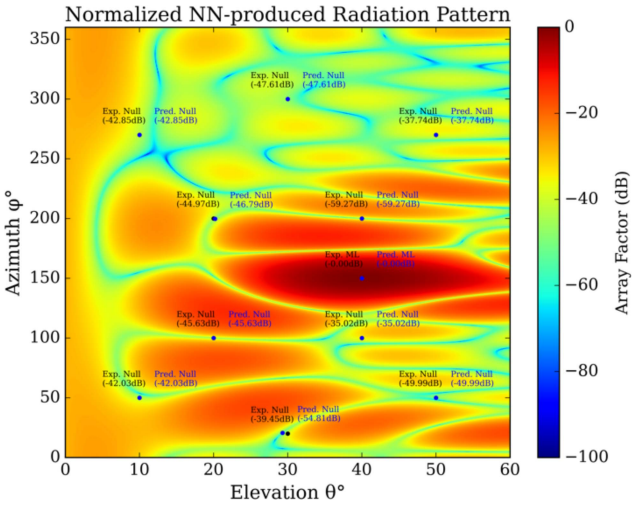

FIGURE 18. Normalized LSTM-RNN-produced radiation pattern for the case of 1 SoI at (40, 150) and 10 random SoAs.

TABLE 14. Statistical analysis of the LSTM-RNN model performance for 21 AoAs.

	Mean	Std
NSB Main Lobe Divergence (°)	3.51	5.51
NN Main Lobe Divergence (°)	3.21	4.89
NSB Null Divergence (°)	0.00	0.00
NN Null Divergence (°)	1.00	1.84
NSB SINR (dB)	15.04	2.98
NN SINR (dB)	15.12	2.97
Array Factor towards SoI [NSB/**NN**] (dB)	-1.41/**-1.31**	1.85/ **0.93**
Array Factor towards SoA [NSB/**NN**] (dB)	-100/ **-33.24**	0.0/ **9.69**
Response Time [NSB/**NN**] (ms)	1.4/ **1.12**	-

they have an effect on its array factor. Consequently, the maximum value of array factor (main lobe peak) can be slightly shifted from the desired direction. This explains the increase in main lobe divergence. Since NSB produces almost infinitely negative array factor towards desired null directions, when nulls are neighboring the main lobe, the power towards the desired direction is decreased [44]. This decrease in desired output power, can cause the SINR to drop (as shown in (12) and (15)). This is not the case of the LSTM-RNN. As Tables 12, 13 and 14 demonstrate, the mean array factor towards the expected null direction is not as "deep" as NSB, but low enough to guarantee good signal rejection.

VIII. SELF-IMPROVING ALGORITHM

In a real-world beamforming scenario, where the electromagnetic environment is constantly changing, supervised DL models would greatly benefit from a method that allows them to adapt to new conditions. In an urban environment these changes might include construction sites, addition or removal of trees, traffic signs and other obstacles. From a DL perspective these new conditions can be defined as two types of cases. First, those which our model has never "seen" before, or in other words, has not been trained

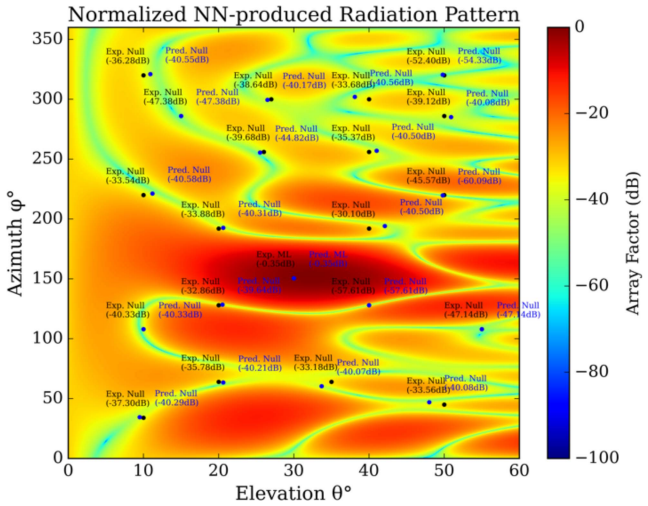

FIGURE 19. Normalized LSTM-RNN-produced radiation pattern for the case of 1 SoI at (30, 150) and 20 random SoAs.

upon yet. This pertains to combinations of SoI and SoAs that were not included in the training dataset, leading to a subpar performance of the model. Secondly, there may be instances where the model, having been trained on certain combinations and achieving satisfactory results, experiences a decline in performance due to unforeseen environmental changes. In these situations, the NN-based beamformer must adapt to maintain the required performance levels. To do so, it might need to take advantage of multipath propagation and signal reflections to enhance the signal coverage and even reach non-line-of-sight areas. Alternatively, it can also take advantage of neighboring intelligent reflecting surfaces (IRSs). Our model learns to map incoming AoAs to the weight vector that better serves the predefined beamforming purpose. In the context of null-steering, the purpose is to correlate the AoAs of desired and undesired signals with weight vectors that steer the main lobe and the nulls towards the optimal directions. In the context of IRS, the NNs can be trained to perform a different kind of correlation. For example, the NN can be trained to steer the main lobe towards an IRS for certain AoAs of the desired incoming signal. This way, the NN-beamformer can learn to selectively utilize a nearby IRS in scenarios where steering the beam directly towards the desired AoA would lead to suboptimal performance. All the aforementioned concepts are predicated on the ability of the NN model to adapt and form new correlations between the direction of incoming AoAs and the appropriate direction in which the produced beam should be steered. Our approach on this adaptation involves two key steps. Pinpointing the scenarios causing difficulties and retraining the NN to yield the required output. Notably, it is not necessary to completely retrain the weights within the neurons; a slight modification (fine-tuning) is often sufficient [45], [46]. An important aspect of the proposed continual-learning approach is its ability to retain previously learned information while incorporating new knowledge.

Recently, deep transfer learning and meta-learning algorithms have been proposed for ABF in dynamic wireless environments, particularly for MISO downlink systems. The transfer learning technique followed in [47] is similar to the one utilized in our self-improvement method in the sense that a pre-trained NN is used and fine-tuned to new data. The difference in our approach is that instead of freezing most of the pre-trained network and using only the final layer for adaptation, we fine-tune all individual neurons of the NN. To do this without degrading the performance of the pre-trained model, we use large, and well-distributed datasets and re-train the models with a very small learning rate. The main reason for this fine-tuning approach is that there is no concrete internal representation of the RNN models that can be detached as a transferable "feature extraction" part. The contribution of each hidden state to the output of the RNN model is equally important while the final fully connected layer merely operates as a transformation layer, to reduce the hidden state size to the desired output size. Thus, we consider fine-tuning all hidden layers a better practice. Coming to the meta-learning approach, it has to be noted that it appears to be superior to transfer-learning as an adaptation technique at the cost of longer execution times due to the need of two backward passes in the training stage [47]. Even though the meta-learning approach is more promising for the task of adaptation compared to transfer learning, our work mainly focuses on investigating which NN-based scheme can perform fast and accurate 3D beamforming utilising the most compact and efficient tuning, training and adaptation methods. The purpose of our proposed self-improvement technique is to provide the means to adapt this supervised learning approach offline, so that it does not require additional operational overhead. Since this fine-tuning is an offline procedure, it does not intervene with the operation of the NN-beamformer. In a hypothetical implementation, one instance of the LSTM-RNN model can actively execute ABF while a cloned instance learns from the first one's errors and undergoes updates as necessary. The updated version of the model can later be deployed, and the fine-tuning can be applied anew, to a new copy of the recently updated model.

The models proposed so far (Tables 10-14) are trained on a very large dataset, comprised of millions of records. Producing such datasets is a highly time-consuming process. A side contribution of the proposed self-improvement method is that it also works as an efficient dataset collection technique. Instead of producing multiple records of random incoming AoAs (as shown in Section VI) to cover as many scenarios as possible, we actively search for the most challenging scenarios. This targeted strategy greatly benefits the NN training. By focusing on difficult cases, the NN learns faster as each record encompasses important information. The more challenging the scenario, the greater the error in the cost function, leading to a more significant contribution of the record to the training process. Moreover, this technique aids at preventing overfitting to common or simple scenarios,

Algorithm 1 Self-Improvement Algorithm

Input: A pre-trained NN-beamformer
Output: An improved, fine-tuned version of the NN-beamformer
model ← Load pretrained model
while *model mean SINR* ≤ *satisfactory SINR* **do**
  Collect 1×10^4 records
  Get the SINR for each record
  Create two empty lists: *badCases, goodCases*
  for record **in** collected records **do**
    if *record SINR* < *model mean SINR* **then**
      Append this record to the *badCases* list
    Else
      Append this record to the *goodCases* list
  End for
  new dataset ← apply NSB to *badCases* to create the new dataset
  new dataset ← apply NSB to *goodCases* (keep less than half the number of *badCases*), mix with previous records and shuffle the rows of the new dataset
  Fine-tune the model on the new dataset using ADAM
  Evaluate the model performance on a different, unbiased and well-distributed dataset and get new *model mean SINR*
End while

ensuring that the trained model generalizes well across a broad spectrum of challenging conditions. We expect that the LSTM-RNN design proposed in Section VI will train much faster and reach similar levels of SINR with that of Table 12 while utilizing less data overall.

This "self-improvement" method employs a pre-trained NN, subjects it to diverse beamforming scenarios, and focuses on progressively enhancing its mean SINR level. We start by exposing the NN to random scenarios of incoming AoAs and keep track of the SINR achieved in each scenario. For SINR values that fall below the model's average SINR, the relevant combination of AoAs is stored in a list of "underperforming" cases. During this process, some of the "well-performing" records are also collected. When enough challenging records are collected, the NSB algorithm is employed to obtain the optimal weight vector that the NN was unable to produce. This way, a new training dataset can be created. To establish a balanced and unbiased dataset, we strategically blend a selection of "well-performing" records into this dataset. Importantly, the number of these records is kept to less than half of the "underperforming" records. This methodical approach ensures the stability of the pre-trained NN model by providing a well-rounded and representative range of data scenarios. We then retrain the pre-trained model using a very small learning rate. This algorithm is better explained below in Algorithm 1.

To realistically emulate demanding conditions and showcase the efficacy of our proposed algorithm, it's necessary to deliberately present the operating NN with beamforming tasks that are challenging. Therefore, instead of deploying a highly trained network for which challenging cases are rare to find, we use an undertrained model that is more likely to underperform. For this reason, an LSTM-RNN of the same design as in Section VI is deployed, trained using only 4×10^4 records. Once again, we refer to the simple case of 1 SoI and 2 SoAs. Since the mean SINR value the NSB achieves for this scenario is 17.87dB (as seen in Tables 10-12), we set it as the maximum expected value (satisfactory SINR value). Each iteration of the algorithm starts with the collection of 1×10^4 records and proceeds with the fine-tuning of the model. Training lasts for 200 epochs, using Adam as the optimizer and a learning rate of 1×10^{-5}. After running this algorithm for 100 iterations (approximately 22 hours of running time on an Apple M1 Max chip with 32GB LPDDR5 SDRAM) we observe the SINR and RMSE improvement shown in Fig. 20.

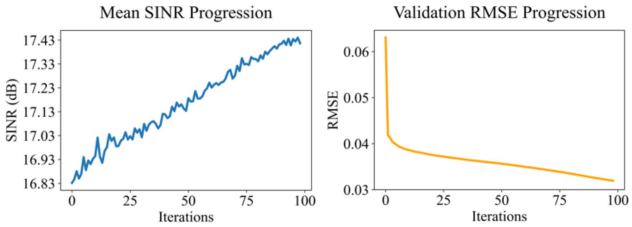

FIGURE 20. SINR and Validation RMSE progression after 100 iterations of the proposed algorithm.

IX. DISCUSSION

Observing the results shown in Tables 13 and 14, and the indicative radiation patterns shown in Figs. 17-18, it is evident that the increase in SoAs causes a small degradation in the accuracy of the model. It should be emphasized that although the proposed design was initially configured based on the simple scenario of 3 incoming AoAs, it is able to provide adequate beamforming performance for up to 21 AoAs. In Section V, we compare different optimization techniques to finally identify Bayesian optimization as the most prevalent to tune an NN-based beamformer for this task. Therefore, a major side-contribution of this paper is that it indicates the most efficient method to configure the NN design according to any specific scenario. If, for example, the purpose of the antenna is to cover a very busy area, then a model trained to cope with a large number of AoAs will be required. That model may be of a bigger size than the one proposed here, in order to better imitate the complexity of the task in hand. We aim not only to demonstrate the performance of NNs on the selected scenarios but also to provide a method by which these NN architectures can easily be redesigned to adapt to any particular scenario.

Finally, it is worth noting that this more targeted record collection technique allows the LSTM-RNN to reach similar performance with that of Table 12 by utilizing only one fifth of the total amount of records used in that training. Creating the dataset of 5×10^6 records was a process that lasted almost a week while with this proposed self-improving technique, both data collection and NN-training can be completed in less than a day.

X. CONCLUSION

This work presents a comprehensive examination of three different NN architectures used as adaptive beamformers on an 8×8 planar antenna array. The objective is to discover a promising NN-based beamformer achieving high SINR with main lobe and null-placement accuracies comparable to the NSB, with the advantage of faster temporal response. We utilize three optimization techniques for hyperparameter tuning, identifying the most effective tuning method and the most promising design for each type of NN. The LSTM-RNN architecture, obtained through Bayesian optimization, emerges as the strongest candidate, displaying excellent accuracy with only 0.04 dB SINR degradation compared to NSB and a much faster response. Its robustness is further verified under varying noise and interference conditions. Testing the same NN structure with increased AoAs shows it maintains excellent beamforming ability for up to 20 interfering signals. Lastly, an iterative self-improving method is proposed to continuously improve the mean SINR level of the NN-based beamformers which also functions as an alternative cost-effective training method.

As future work, a more hands-on implementation incorporating the proposed models on a simulated network to observe their behavior under dynamic real-time conditions would be highly insightful. Given the unprecedent performance of the proposed NN-based beamformers, their robustness when applied to realistic planar antenna arrays should also be examined in the future.

ACKNOWLEDGMENT

Results presented in this work have been produced using the Aristotle University of Thessaloniki (AUTh) High Performance Computing Infrastructure and Resources.

REFERENCES

[1] A. M. G. Guerreiro, A. D. D. Neto, and F. A. Lisboa, "Beamforming applied to an adaptive planar array," in *Proc. IEEE Radio Wireless Conf.*, Aug. 1998, pp. 209–212, doi: 10.1109/RAWCON.1998.709173.

[2] C. A. Balanis, *Antenna Theory: Analysis and Design*, 4th ed. Newark, NJ, USA: Wiley, 2016.

[3] Z. D. Zaharis, I. P. Gravas, P. I. Lazaridis, T. V. Yioultsis, and T. D. Xenos, "Improved beamforming in 3D space applied to realistic planar antenna arrays by using the embedded element patterns," *IEEE Trans. Veh. Technol.*, vol. 71, no. 6, pp. 6145–6157, Jun. 2022, doi: 10.1109/TVT.2022.3155966.

[4] J. F. Cardoso and A. Souloumiac, "Blind beamforming for non-gaussian signals," *IEEE Proc. F Radar Signal Process.*, vol. 140, no. 6, pp. 362–370, Dec. 1993, doi: 10.1049/ip-f-2.1993.0054.

[5] T. S. Kiong, S. B. Salem, J. K. S. Paw, K. P. Sankar, and S. Darzi, "Minimum variance distortionless response beamformer with enhanced nulling level control via dynamic mutated artificial immune system," *Sci. World J.*, vol. 2014, Jun. 2014, Art. no. e164053, doi: 10.1155/2014/164053.

[6] M. Abualhayja'a and M. Hussein, "Comparative study of adaptive beamforming algorithms for smart antenna applications," in *Proc. Int. Conf. Commun., Signal Process. Appl. (ICCSPA)*, Mar. 2021, pp. 1–5, doi: 10.1109/ICCSPA49915.2021.9385725.

[7] Y. Lv, F. Cao, X. Feng, and H. Li, "Improved binary particle swarm optimization and its application to beamforming of planar antenna arrays," *Progr. Electromagn. Res. C*, vol. 114, pp. 217–231, Aug. 2021, doi: 10.2528/PIERC21062002.

[8] M. Sarevska and A.-B. M. Salem, "Antenna array beamforming using neural network," World Acad. Sci. Eng. Technol., Dec. 2008, doi: 10.5281/zenodo.1059687.

[9] H. A. Kassir, Z. D. Zaharis, P. I. Lazaridis, N. V. Kantartzis, T. V. Yioultsis, and T. D. Xenos, "A review of the state of the art and future challenges of deep learning-based beamforming," *IEEE Access*, vol. 10, pp. 80869–80882, 2022, doi: 10.1109/ACCESS.2022.3195299.

[10] G. L. Santos, P. T. Endo, D. Sadok, and J. Kelner, "When 5G meets deep learning: A systematic review," *Algorithms*, vol. 13, no. 9, p. 9, Sep. 2020, doi: 10.3390/a13090208.

[11] Y. Li, Y. Huang, G. F. Pedersen, and M. Shen, "Recurrent NEAT assisted 2D-DOA estimation with reduced complexity for satellite communication systems," *IEEE Access*, vol. 10, pp. 11551–11563, 2022, doi: 10.1109/ACCESS.2022.3145583.

[12] Y. Ma, Y. Zeng, and S. Sun, "A deep learning based super resolution DoA estimator with single snapshot MIMO radar data," *IEEE Trans. Veh. Technol.*, vol. 71, no. 4, pp. 4142–4155, Apr. 2022, doi: 10.1109/TVT.2022.3151674.

[13] Z.-M. Liu, C. Zhang, and P. S. Yu, "Direction-of-arrival estimation based on deep neural networks with robustness to array imperfections," *IEEE Trans. Antennas Propag.*, vol. 66, no. 12, pp. 7315–7327, Dec. 2018, doi: 10.1109/TAP.2018.2874430.

[14] W. Fang et al., "A deep learning based mutual coupling correction and DOA estimation algorithm," in *Proc. 13th Int. Conf. Wireless Commun. Signal Process. (WCSP)*, Oct. 2021, pp. 1–5, doi: 10.1109/WCSP52459.2021.9613199.

[15] S. Xu, B. Chen, H. Lian, and Z. Guo, "Deep learning based direction of arrival estimation of multiple targets," in *Proc. IEEE 5th Int. Conf. Electron. Inf. Commun. Technol. (ICEICT)*, Aug. 2022, pp. 138–143, doi: 10.1109/ICEICT55736.2022.9908986.

[16] I. Mallioras, Z. D. Zaharis, P. I. Lazaridis, and S. Pantelopoulos, "A novel realistic approach of adaptive beamforming based on deep neural networks," *IEEE Trans. Antennas Propag.*, vol. 70, no. 10, pp. 8833–8848, Oct. 2022, doi: 10.1109/TAP.2022.3168708.

[17] H. P. Z. Cano, Z. D. Zaharis, T. V. Yioultsis, N. V. Kantartzis, and P. I. Lazaridis, "Pattern reconfigurable antennas at millimeter-wave frequencies: A comprehensive survey," *IEEE Access*, vol. 10, pp. 83029–83042, 2022, doi: 10.1109/ACCESS.2022.3196456.

[18] I. Chafaa, R. Negrel, E. V. Belmega, and M. Debbah, "Self-supervised deep learning for mmWave beam steering exploiting sub-6 GHz channels," *IEEE Trans. Wireless Commun.*, vol. 21, no. 10, pp. 8803–8816, Oct. 2022, doi: 10.1109/TWC.2022.3170104.

[19] M. Chu, A. Liu, V. K. N. Lau, C. Jiang, and T. Yang, "Deep reinforcement learning based end-to-end multiuser channel prediction and beamforming," *IEEE Trans. Wireless Commun.*, vol. 21, no. 12, pp. 10271–10285, Dec. 2022, doi: 10.1109/TWC.2022.3183255.

[20] J. M. J. Huttunen, D. Korpi, and M. Honkala, "DeepTx: Deep learning beamforming with channel prediction," 2022, *arXiv:2202.07998*.

[21] I. Mallioras et al., "A novel utilization of NARX for antenna array adaptive beamforming," in *Proc. 3rd URSI Atlantic Asia–Pacific Radio Sci. Meeting (AT-AP-RASC)*, May 2022, pp. 1–4, doi: 10.23919/AT-AP-RASC54737.2022.9814406.

[22] Z. D. Zaharis, C. Skeberis, T. D. Xenos, P. I. Lazaridis, and J. Cosmas, "Design of a novel antenna array beamformer using neural networks trained by modified adaptive dispersion invasive weed optimization based data," *IEEE Trans. Broadcast.*, vol. 59, no. 3, pp. 455–460, Sep. 2013, doi: 10.1109/TBC.2013.2244793.

[23] S. I. Orakwue, R. Ngah, T. A. Rahman, and S. Z. M. Hashim, "Neural network based switch beam smart antenna," in *Proc. IEEE Asia–Pacific Conf. Wireless Mobile*, Aug. 2014, pp. 292–296, doi: 10.1109/APWiMob.2014.6920300.

[24] P. Ramezanpour, M. J. Rezaei, and M. R. Mosavi, "Deep-learning-based beamforming for rejecting interferences," *IET Signal Process.*, vol. 14, no. 7, pp. 467–473, 2020, doi: 10.1049/iet-spr.2019.0495.

[25] I. Mallioras, Z. D. Zaharis, P. I. Lazaridis, V. Poulkov, N. V. Kantartzis, and T. V. Yioultsis, "An adaptive beamforming approach applied to planar antenna arrays using neural networks," in *Proc. IEEE Int. Black Sea Conf. Commun. Netw. (BlackSeaCom)*, Jun. 2022, pp. 293–297, doi: 10.1109/BlackSeaCom54372.2022.9858302.

[26] R. Lovato and X. Gong, "Phased antenna array beamforming using convolutional neural networks," in *Proc. IEEE Int. Symp. Antennas Propag. USNC-URSI Radio Sci. Meeting*, Jul. 2019, pp. 1247–1248, doi: 10.1109/APUSNCURSINRSM.2019.8888573.

[27] R. Dong, B. Wang, and K. Cao, "Deep learning driven 3D robust beamforming for secure communication of UAV systems," *IEEE Wireless Commun. Lett.*, vol. 10, no. 8, pp. 1643–1647, Aug. 2021, doi: 10.1109/LWC.2021.3075996.

[28] A. Alkhateeb, S. Alex, P. Varkey, Y. Li, Q. Qu, and D. Tujkovic, "Deep learning coordinated beamforming for highly-mobile millimeter wave systems," *IEEE Access*, vol. 6, pp. 37328–37348, 2018, doi: 10.1109/ACCESS.2018.2850226.

[29] G. Eappen, J. Cosmas, T. Shankar, A. Rajesh, R. Nilavalan, and J. Thomas, "Deep learning integrated reinforcement learning for adaptive beamforming in B5G networks," *IET Commun.*, vol. 16, no. 20, pp. 2454–2466, 2022, doi: 10.1049/cmu2.12501.

[30] Z. D. Zaharis, I. P. Gravas, P. I. Lazaridis, T. V. Yioultsis, C. S. Antonopoulos, and T. D. Xenos, "An effective modification of conventional beamforming methods suitable for realistic linear antenna arrays," *IEEE Trans. Antennas Propag.*, vol. 68, no. 7, pp. 5269–5279, Jul. 2020, doi: 10.1109/TAP.2020.2977822.

[31] J. Sola and J. Sevilla, "Importance of input data normalization for the application of neural networks to complex industrial problems," *IEEE Trans. Nucl. Sci.*, vol. 44, no. 3, pp. 1464–1468, Jun. 1997, doi: 10.1109/23.589532.

[32] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, vol. 1. Cambridge, MA, USA: MIT Press, 2016.

[33] N. Bakas, G. Markou, D. Charmpis, and K. Hadjiyiannakou, "Performance and scalability of deep learning models trained on a hybrid supercomputer: Application in the prediction of the shear strength of slender RC beams," presented at the 8th Int. Conf. Comput. Methods Struct. Dyn. Earthquake Eng. Methods Struct. Dyn. Earthquake Eng., Athens, Greece, 2021, pp. 3878–3893, doi: 10.7712/120121.8754.19593.

[34] E. Pellegrino et al., "Deep learning architecture optimization with metaheuristic algorithms for predicting BRCA1/BRCA2 pathogenicity NGS analysis," *BioMedInformatics*, vol. 2, no. 2, p. 2, Jun. 2022, doi: 10.3390/biomedinformatics2020016.

[35] A. Agnihotri and N. Batra, "Exploring Bayesian optimization," *Distill*, vol. 5, no. 5, p. e26, May 2020, doi: 10.23915/distill.00026.

[36] R. Garnett, *Bayesian Optimization*. Cambridge, U.K.: Cambridge Univ. Press, 2023. doi: 10.1017/9781108348973.

[37] J. Bergstra, D. Yamins, and D. Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," in *Proc. 30th Int. Conf. Mach. Learn.*, Feb. 2013, pp. 115–123.

[38] J. Snoek, H. Larochelle, and R. Adams, "Practical Bayesian optimization of machine learning algorithms," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 4, Jun. 2012, pp. 1–12.

[39] R. Turner et al., "Bayesian Optimization is superior to random search for machine learning hyperparameter tuning: Analysis of the black-box optimization challenge 2020," in *Proc. Competition Demonstration Track*, Aug. 2021, pp. 3–26.

[40] M. Balandat et al., "BOTORCH: A framework for efficient Monte-Carlo Bayesian optimization," in *Proc. Adv. Neural Inf. Process. Syst.*, Dec. 2020, pp. 21524–21538.

[41] K. Sastry, D. Goldberg, and G. Kendall, "Genetic Algorithms," in *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, E. K. Burke and G. Kendall, Eds. Boston, MA, USA: Springer, 2005, pp. 97–125, doi: 10.1007/0-387-28356-0_4.

[42] Y. N. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio, "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization," in *Proc. 27th Int. Conf. Neural Inf. Process. Syst.*, vol. 2, Dec. 2014, pp. 2933–2941.

[43] S. Mabrouki, I. Dayoub, Q. Li, and M. Berbineau, "Codebook designs for millimeter-wave communication systems in both low- and high-mobility: Achievements and challenges," *IEEE Access*, vol. 10, pp. 25786–25810, 2022, doi: 10.1109/ACCESS.2022.3154016.

[44] T. Nguyen, "Null depth trade-off for output power reduction in a downlink adaptive antenna array," M.S. thesis, School Eng. Sci., Victoria Univ., Melbourne, VIC, Australia, 2006. [Online]. Available: https://vuir.vu.edu.au/522/

[45] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?," in *Proc. 27th Int. Conf. Neural Inf. Process. Syst.*, vol. 2, Dec. 2014, pp. 3320–3328.

[46] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. 3rd Int. Conf. Learn. Represent.*, 2015, pp. 1–14.

[47] Y. Yuan, G. Zheng, K.-K. Wong, B. Ottersten, and Z.-Q. Luo, "Transfer learning and meta learning-based fast downlink beamforming adaptation," *IEEE Trans. Wireless Commun.*, vol. 20, no. 3, pp. 1742–1755, Mar. 2021, doi: 10.1109/TWC.2020.3035843.