A Low-complexity Structured Neural Network Approach to Intelligently Realize Wideband Multi-beam Beamformers

Hansaka Aluvihare, Sivakumar Sivasankar, Xianqi Li, Arjuna Madanayake, and Sirani M. Perera.

***Abstract*—True-time-delay (TTD) beamformers can produce wideband squint-free beams in both analog and digital signal domains, unlike frequency-dependent FFT beams. Our previous work showed that TTD beamformers can be efficiently realized using the elements of the delay Vandermonde matrix (DVM), answering the longstanding beam-squint problem. Thus, building on our work on DVM algorithms, we propose a structured neural network (StNN) to realize wideband multi-beam beamformers using structure-imposed weight matrices and submatrices. The structure and sparsity of the weight matrices and submatrices are shown to reduce the computational complexity of the NN significantly. The proposed StNN architecture has $\mathcal{O}(pLM \log M)$ complexity compared to a conventional fully connected L-layers network with $\mathcal{O}(M^2 L)$ complexity, where M is the number of nodes in each layer of the network, p is the number of sub-weight matrices per layer, and $M >> p$.**

We show numerical simulations in the 24 to 32 GHz range to demonstrate the numerical feasibility of realizing wideband multi-beam beamformers using the proposed StNN architecture. We also show the complexity reduction of the proposed NN and compare that with fully connected NNs, to show the efficiency of the proposed architecture without sacrificing accuracy. The accuracy of the proposed NN architecture was shown in terms of the mean squared error, which is based on an objective function of the weight matrices and beamformed signals of antenna arrays, while also normalizing nodes. The proposed StNN's robustness was tested against channel impairments by simulating with Rayleigh fading at different signal-to-noise ratios (SNRs). We show that the proposed StNN architecture leads to a low-complexity NN to realize wideband multi-beam beamformers, enabling a path for reconfigurable intelligent systems.

***Index Terms*—Intelligent Systems, Structured Neural Networks, Wideband Multi-beam Beamformers, Structured Weight Matrices, Delay Vandermonde Matrix, Low-complexity Neural Networks, Wireless Communication Systems**

H. Aluvihare is with the Department of Mathematics, Embry-Riddle Aeronautical University, Daytona Beach, FL, 32703 USA e-mail:aluvihah@my.erau.edu

S. Sivasankar is with the Department of Electrical and Computer Engineering, Florida International University, Miami, FL, 33174 USA e-mail:ssiva011@fiu.edu

X. Li is with the Department of Mathematics & Systems Engineering, Florida Institute of Technology, Melbourne, FL 32901, USA e-mail: xli@fit.edu (see https://www.fit.edu/faculty-profiles/l/li-xianqi/).

A. Madanayake is with the Department of Electrical & Computer Engineering, Florida International University, Miami, FL 33174 USA e-mail: amadanay@fiu.edu (see https://ece.fiu.edu/people/faculty/profiles/madanayake-arjuna/).

S. M. Perera is with the Department of Mathematics, Embry-Riddle Aeronautical University, Daytona Beach, FL, 32703 USA e-mail: pereras2@erau.edu (see https://faculty.erau.edu/Sirani.Perera).

This work was supported by the National Science Foundation award numbers 2229473 and 2229471.

I. Introduction

Beamforming has been widely explored for its diverse applications across fields, such as radar, communication, and imaging. The transmission of beamforming overcomes the path loss by concentrating energy in a specific direction, while the reception of beamforming progressively improves the propagation of planar waves based on the desired direction of arrival [1]. When the signal of interest is wideband, multi-beam beamforming based on the spatial Fast Fourier Transform (FFT) suffers from the beam-squint problem [2].

A. Realize TTD-based Beamformers via DVM

The FFT beams are frequency-dependent and thus cause poor beam orientations for wideband signals. Fortunately, the true-time-delay (TDD) beamformers have significantly mitigated the beam-squint problem associated with spatial FFT beams [3]. On the other hand, the delay Vandermonde matrix (DVM) elements can be utilized to determine the TTD beams [2]–[5]. This amounts to incorporating the DVM between antennas and source/sink channels and implementing via frequency-dependent phase shifts at each antenna to achieve TDD beamformers leading to wideband multi-beam architecture. Thus, utilizing TDD, at time $t \in \mathbb{R}$, the N-beam beamformer $\tilde{y} \in \mathbb{C}^N$ can be expressed as a product of the input vector $\tilde{x} \in \mathbb{C}^N$ and the DVM, i.e., $A_N \in \mathbb{C}^{N \times N}$, s.t., $\tilde{y} = A_N \tilde{x}$. In this context, each row of the DVM symbolizes the progressive wideband phase shift associated with a specific beam. However, computational cost plays a crucial role in computing the matrix-vector product associated with wideband multi-beam beamformers. Each TTD is typically realized in the digital domain using a finite impulse response (FIR) digital filter - sometimes known as a Frost Structure. Thus, in order to reduce the delays of N beams from $\mathcal{O}(N^2)$ to $\mathcal{O}(N \log N)$, we proposed sparse factorization to realize narrowband multi-beam beamformers in [4] and wideband multibeam beamformers in [2], [3], [5]. The necessity of retaining intermediate values in memory can result in increased memory demands. Such circumstances may pose a disadvantage in real-time or low-latency applications. Hence, a critical requirement emerges for the development of a real-time training and prediction algorithm to effectively realize wideband multi-beam beamforming. Thus, we propose a structured neural network (StNN) to efficiently realize wideband multi-beam beamformers by training and learning weight matrices with structure, enabling us to develop lightweight and low-complexity NNs. We show

the efficiency and effectiveness of the proposed StNN through numerical simulations for beamformers in the 24 to 32 GHz range.

B. Neural Networks Approaches for Beamformers

Several methods have been proposed for the application of both shallow and deep neural networks or multi-layer perceptrons in the context of adaptive beamforming as applied to phased arrays [6]–[9]. In [10], a radial basis function neural network (RBFNN) was employed to approximate the beamformers derived through the application of a minimum mean-squared error (MSE) beamforming criterion while adhering to a specified gain constraint. In [11], a NN was trained to create adaptive transmit and receive narrowband digital beamformers for a fully digital phased array. Many adaptive algorithms based on convolutional neural networks (CNN) have been proposed, such as [12]–[16]. In [12], an approach known as frequency constraint wideband beamforming prediction network (WBPNet) is introduced (without a delay structure) based on a CNN method to tackle the limitations associated with insufficient received signal snapshots while reducing computational complexity. This CNN-based method focused on predicting the direction of arrival (DOA) of interference. Next, authors in [16] introduce a CNN-based neural beamformer to predict the interference from received signals and an LSTM model to predict the samples of desired signals for a low number of receiving snapshots. In [17], a CNN is trained based on the data obtained from the optimum Wiener solution, and results are compared with 8×8 antenna arrays. Moreover, in [18], a scheme is introduced to predict a power allocation vector before determining the beamforming matrix with CNN. This method addresses the challenge of overly complex networks and power minimization problems in the context of wideband beamforming for synthetic aperture radar (SAR). The above methods include training a CNN model to design the beamformer for specific sizes of antenna arrays. However, as the number of elements in the array increases (which is expected for mmWave communications), there is a lack of research that evaluates the relative performance of the above methods. The authors of [19] proposed a multilayer neural network model to design a beamformer for 64-element arrays to tackle the challenges in imperfect CSI and hardware challenges by maximizing the spectral efficiency. Besides, [20] proposed a CNN-based beamformer to estimate the phase values for beamforming. Furthermore, [21] and [22] explored the recurrent neural network-based algorithm to estimate the weights in the antenna array. Authors in [21] proposed GRU-based ML algorithms for adaptive beamforming.

C. Structured Weight Matrices in Neural Networks

As modern NN architectures grow in size and complexity, the demand for computational resources is significantly increasing. Structured weight matrices present a solution to mitigate this increased resource consumption by simplifying computational tasks [23]. These matrices, by leveraging inherent structures, can reduce the computational complexity for propagating information through the network [24]. However, selecting the appropriate structure within the diverse array of matrix structures and classes is not a trivial task. To address this challenge, numerous methods [25]–[32] have been developed to minimize the computational costs and memory requirements of neural networks. Those existing efforts generally fall into two categories: reduction techniques focused on fully-connected NN including weight pruning/clustering [26], [27], which prune and cluster the weights via scalar quantization, product quantization, and residual quantization, to reduce the NN model size, and reduction strategies aimed at convolutional layers, such as low-rank approximation [30], [33], [34] and sparsity regularization [25], [28]. These approaches are critical for enhancing the efficiency of neural networks, making them more practical for a variety of applications.

D. Objective of the Paper

Our goal is to introduce a structure-imposed NN, i.e. StNN, to realize multi-beam beamformers while dynamically updating the StNN with low-complexity and lightweight NN. We have shown that the sparse factorization of the DVM is an efficient strategy to reduce the complexity in computing the DVM-vector product from $\mathcal{O}(N^2)$ to $\mathcal{O}(N \log N)$. Nevertheless, it is crucial to dynamically update delays and sums based on the DVM-vector product so that we can intelligently realize multi-beam beamformers. In order to do so, we regularize weight matrices within network while adopting the sparse factorization of the DVM in [3]. This adoption leads us to train, learn, and dynamically update the TTD beamformers while imposing the structure of the DVM, followed by its sparse factors based on structured matrices. Hence, we propose a hybrid of classical and ML algorithms to dynamically realize wideband multi-beam beamforming, in contrast to weight-pruning techniques that result in irregular pruned networks [35]. Since the DVM can be determined by the $\mathcal{O}(N)$ parameters, utilizing its structure and factorization in NN weight matrices should reduce the computational complexity. Thus, the proposed StNN architecture leads to

1) intelligently realizes wideband multi-beam beamformers while reducing TTD blocks,
2) ensures a robust structure for a trained network while reducing computational complexities incurred by complex indexing processes,
3) reduce computational complexities due to the usage of structured and sparse weight matrices, i.e., 70% complexity reduction compared to our previous paper [36], and
4) obtain a lightweight and robust NN while intelligently realizing wideband multi-beam beamformers.

We note here that the DVM is a low-displacement rank (LDR) matrix, and LDR-based neural networks have gained attention due to their potential to reduce complexity when the structure is imposed for the neural network [29]–[31]. Thus, the utilization of the DVM structure followed by factorization of the low-rank DVM in [3], without the need for retraining (due to utilization of frequencies, i.e., 24, 27, & 28 GHz) lead to propose a low-complexity StNN that can be utilized to intelligently realize wideband multi-beam beamformers.

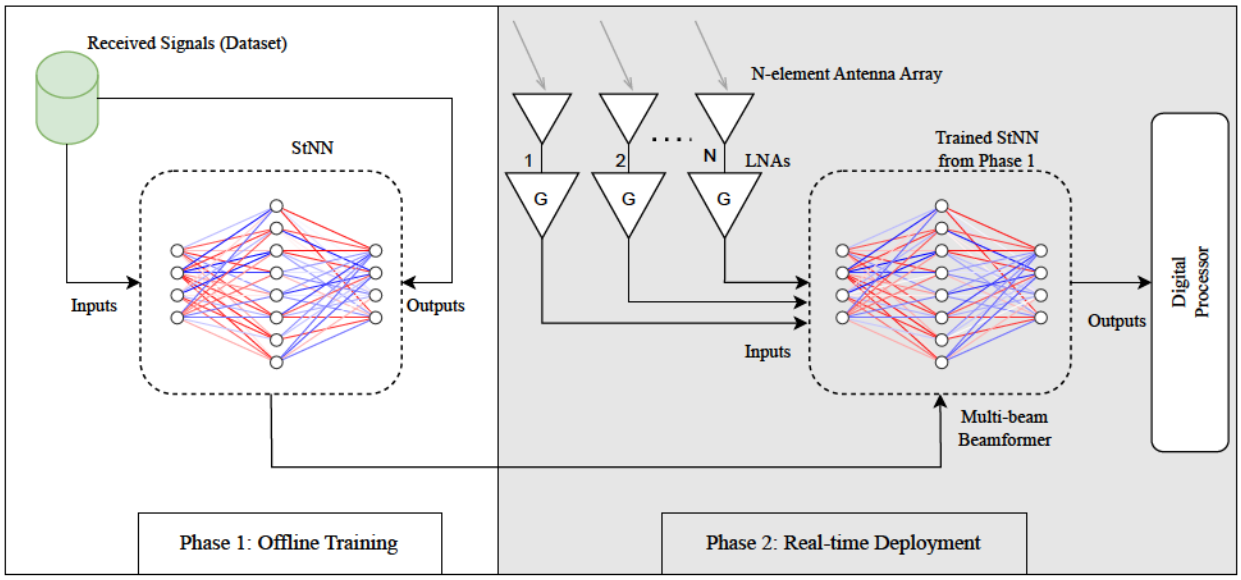


Fig. 1. ML-based architecture of multi-beam beamforming: In the offline training, we train the neural network to align the input data by weight matrices to the desired output data. In real-time deployment: RF signals from the antennas and low noise amplifiers (LNAs) are beamformed utilizing the structure-imposed neural network, i.e., StNN. Once the multibeams are formed, they will be sent to the digital processor.

The authors in [29] introduced the concept of leveraging LDR matrices in NNs to reduce both storage and computational overhead. This was achieved by factorization via displacement equations of structured matrices [37]. In contrast, our approach employs the StNN, leveraging the DVM factorization instead of displacement equations. This strategic choice effectively minimizes the number of trainable weights, inference time, and floating-point operations (FLOPs), leading to a more efficient and computationally lightweight neural network architecture for multi-beam beamforming applications.

E. Structure of the Paper

The remainder of the paper is organized as follows. Section II introduces the theory of the structure-imposed neural network, i.e., StNN, to realize wideband multi-beam beamformers. Section III shows the arithmetic complexity of the StNN, showing the reduction of the complexity. Section IV shows the numerical simulations showing the efficiency and accuracy of the StNN as opposed to the fully connected neural networks in realizing wideband multibeam beamformers in the 24 GHz to 32 GHZ range. Finally, the Section V concludes the paper.

II. Methodology

We first present the background of the DVM factorization in [3], enabling us to theoretically formulate the Structured Neural Network to realize wideband multi-beam beamformers. The StNN utilizes custom weight matrices to effectively incorporate the structure of the DVM, followed by a sparse factorization in [3]. This approach enables efficient computation of DVM-vector multiplications, enabling the realization of wideband multi-beam beamformers, and ensuring model stability. Unlike conventional feed-forward neural networks(FFNN), StNN significantly reduces the computational complexity. It optimizes space and storage requirements, handling large-scale systems involving high values of N, making it a more scalable, low-complexity learning algorithm compared to conventional multi-beam beamformers.

The high-level design of the proposed framework is shown in Figure 1, illustrating a two-phase process comprising offline training and real-time deployment. During real-time operation, the StNN processes input data from each antenna element to estimate the multibeam beamformer output. Prior to deployment, the StNN model undergoes offline training on a pre-collected dataset of received RF signals. After training, the StNN predicts N beamformer output signals based on the true time-delay Vandermonde beamformer, which the digital processor then uses for further processing.

A. DVM Factorization in [3]

The DVM is defined using the nodes $\{1, \alpha, \alpha^2, \cdots, \alpha^{N-1}\}$ via the matrix $A_N = \{\alpha^{kl}\}_{k=1,l=0}^{N,N-1}$, where $\alpha = e^{-j\omega\tau} \in \mathbb{C}$, $N = 2^r (r \geq 1)$, ω represents the temporal frequency, τ denotes the time delay, and $j^2 = -1$. In [3], we introduced a scaled version of the DVM, denoted as $\widetilde{A}_N$, followed by a sparse matrix factorization. This factorization enables efficient computation of the scaled DVM-vector product using an optimized algorithm with a computational complexity of $O(N \log(N))$, as expressed in the following equation:

$$\widetilde{A}_N = \hat{D}_N [J_{M\times N}]^T F_M^* \breve{D}_M F_M J_{M\times N} \hat{D}_N, \qquad (1)$$

where $M = 2N$, $\hat{D}_N = \text{diag}[\alpha^{\frac{k^2}{2}}]_{k=0}^{N-1}$ is a diagonal scaling matrix, $\breve{D}_M = \text{diag}\left[\tilde{F}_M \mathbf{c}\right]$ is a diagonal matrix and the column vector $\mathbf{c}$ s.t. $\mathbf{c} = \left[1, \alpha^{-\frac{1}{2}}, \cdots, \alpha^{-\frac{(N-1)^2}{2}}, 1, \alpha^{-\frac{(N-1)^2}{2}}, \alpha^{-\frac{(N-2)^2}{2}}, \cdots, \alpha^{-\frac{1}{2}}\right]^T$,

$\tilde{F}_M = \sqrt{M}F_M$, $J_{M\times N} = \begin{bmatrix} I_N \\ 0_N \end{bmatrix}$ is a sparse matrix, I_N denotes the identity matrix, 0_N represents the zero matrix, $F_N = \frac{1}{\sqrt{N}}[\omega_N^{kl}]_{k,l=0}^{N-1}$ is the discrete Fourier transform (DFT) matrix defined via the nodes $\omega_N = e^{-\frac{2\pi j}{N}}$, and F_M^* represents the conjugate transpose of the DFT matrix F_M. This factorization not only enhances computational efficiency but also reduces storage and processing complexity, making it a viable approach for large-scale implementations.

B. DVM Structure-imposed Neural Networks (StNN)

To efficiently compute wideband multi-beam beamformers, we use the DVM-vector product based on the factorization equation (1) to impose structure for the weight matrices of the StNN. The proposed StNN follows an L-layer feedforward architecture, consisting of an input layer, output layer, and l hidden layers, where $l = L - 2$. Notably, this framework is adaptable, allowing for the addition of more hidden layers and units to accommodate the accuracy of the predictions.

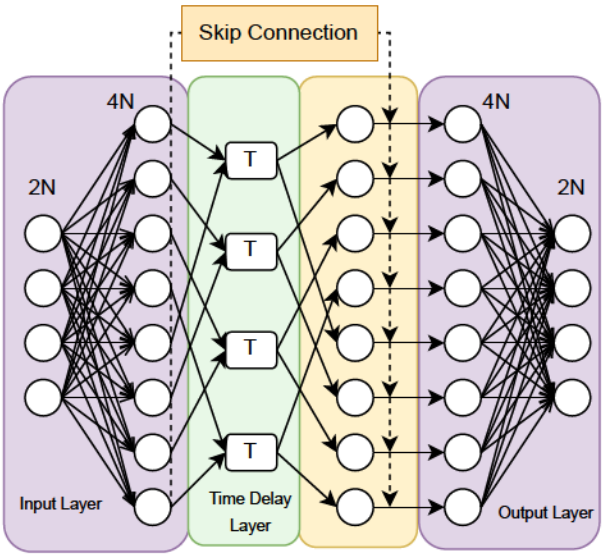


Fig. 2. StNN architecture for predicting the output of the TTD beamformers. $2N$ neurons in the input layer (separating real-vales and imaginary parts of the received vector $\tilde{x} \in \mathbb{C}^N$, giving $2N$ neurons representing the input vector $\underline{x} \in \mathbb{R}^{2N}$), $4pN$ neurons in the hidden layer, where $p, N \in \mathbb{Z}^+$, $4pN$ neurons in each hidden layer, and $2N$ neurons in the output vector $\underline{y} \in \mathbb{R}^{2N}$ resulting the beamformed vector $\tilde{y} \in \mathbb{C}^N$.

The neural network architecture of the StNN model is illustrated in Fig. 2. Given that the received RF signal from N-element antenna array consists of complex-valued signal $\tilde{x} \in \mathbb{C}^N$, we separate the real and imaginary components of each signal to pass that to the StNN. This transformation ensures that only real-valued inputs $\underline{x} \in \mathbb{R}^{2N}$ are processed within the StNN. Each complex number $a + ib$ is mapped to a real-valued pair $(a, b) \in \mathbb{R}^2$. Consequently, for N complex-valued inputs, a corresponding real-valued input vector of size $2N$ is generated for the StNN.

1) Forward Propagation of the StNN: First, we obtain the forward propagation equations for the StNN. Here, we consider the input layer consisting $2N$ neurons and a fully connected hidden layer with $4pN$ neurons having a weight matrix $W \in \mathbb{R}^{4pN\times 2N}$, where p is the number of submatrices. When a $2N$ sized input vector $\underline{x}$ is given to the StNN, the general forward propagation equation for the first hidden layer through can be expressed as

$$\underline{y}^{(1)} = \sigma(W^{(1)}\underline{x} + \underline{\theta}^{(1)}), \tag{2}$$

where $\underline{y}^{(1)} \in \mathbb{R}^{4pN}$ is the output of the first hidden layer, $W^{(1)} \in \mathbb{R}^{4pN\times 2N}$ is the weight matrix between the input layer and the first hidden layer (defined by the sparse matrices aligned with the factorization equation 1 (as described next), θ is the bias vector, and $\sigma(.)$ denotes the activation function of the current layer.

In general, the forward propagation equation for the output vector $\underline{y}^{(l+1)}$ at the $(l+1)$-th hidden layer can be expressed as,

$$\underline{y}^{(l+1)} = \sigma(W^{(l+1)}\underline{y}^{(l)} + \underline{\theta}^{(l+1)}),$$

where, $\underline{y}^{(l)}$ is the output vector of the previous hidden layer. Next, we redesign the weight matrices between the layers of the StNN. More precisely, weight matrices, $W^{(1)}$ (i.e. weight matrix between the input layer and the first hidden layer) and $W^{(4)}$(i.e. weight matrix between the last hidden layer and the output layer) in StNN shown in the Fig. 2, is decomposed into p smaller sub-weight matrices. For instance, the weight matrix between the input layer and the first hidden layer, i.e. $W^{(1)}$, is structured as shown below.

$$W^{(1)} = \begin{bmatrix} w_1^{(1)} & w_2^{(1)} & \cdots & w_p^{(1)} \end{bmatrix}^T, \tag{3}$$

where $w_i^{(1)}$ for $i = 1, 2, \cdots, p$ are p sub-weight matrices defined via 9. Similarly, the weight matrix between the last hidden layer and the output layer, i.e. $W^{(4)}$, follows the structure presented below.

$$W^{(4)} = [w_1^{(4)}, w_2^{(4)}, w_3^{(4)}, ..., w_p^{(4)}], \tag{4}$$

where $w_i^{(4)}$ for $i = 1, 2, \cdots, p$ are p sub-weight matrices defined via 10. Moreover, the weight matrix between the first hidden layer and the second hidden layer, i.e., $W^{(2)}$ is not fully connected and it acts as a physical delay to the signals, meaning it does not contain any trainable weights. Instead, it consists solely of $2pN$ time delay elements. The primary purpose of this layer is to introduce delays to the signal transformed by the first layer. Each delay element applies a fixed delay to the output signal from the first hidden layer. With the introduction of a physical delay layer inside the network, StNN evolves to better fit for wideband multi-beam beamformers, where beamformers are realized with TTDs. We note here that the output of the first hidden layer, i.e. $\underline{y}^{(1)} \in \mathbb{R}^{4pN}$, is a real-valued vector. Therefore, before applying the delay, this real-valued vector is converted into a complex-valued signal $\tilde{y}^{(1)}$ as follows.

$$\tilde{\underline{y}}^{(1)} = \underline{y}_{1:2pN}^{(1)} + j\underline{y}_{2pN+1:4pN}^{(1)}, \tag{5}$$

where $\underline{y}^{(1)}_{1:2pN}$ represents the first half of the vector $\underline{y}^{(1)}$, i.e., representing the 1^{st} element to the $2pN^{\text{th}}$ element of $\underline{y}^{(1)}$ to gather the real part, and $\underline{y}^{(1)}_{2pN+1:4pN}$ represents the second half of the vector $\underline{y}^{(1)}$, i.e., representing the $(2pN+1)^{\text{st}}$ element to the $4pN^{\text{th}}$ element of $\underline{y}^{(1)}$ to gather the imaginary part. Next, the delay is applied to the reconstructed complex signal $\underline{\tilde{y}}^{(1)} \in \mathbb{C}^{2pN}$ producing $\underline{\tilde{y}}^{(2)} \in \mathbb{C}^{2pN}$ s.t.

$$\underline{\tilde{y}}^{(2)} = W^{(2)}.\underline{\tilde{y}}^{(1)}, \tag{6}$$

where $W^{(2)} = \text{diag}\left[\alpha^k\right]_{k=0}^{2pN-1}$. Afterwards, $\underline{\tilde{y}}^{(2)}$ is converted back into a real-valued signal $\underline{y}^{(2)}$ by separating the real and imaginary components s.t.

$$\underline{y}^{(2)}_{1:2pN} := \underline{\tilde{y}}^{(2)}_{Re}, \qquad \underline{y}^{(2)}_{2pN+1:4pN} := \underline{\tilde{y}}^{(2)}_{Im}, \tag{7}$$

where, $\underline{\tilde{y}}^{(2)}_{Re}$ represents the real components of $\underline{\tilde{y}}^{(2)}$ and $\underline{\tilde{y}}^{(2)}_{Im}$ represents the imaginary components of the $\underline{\tilde{y}}^{(2)}$. The resulting output of the second hidden layer, i.e. $\underline{y}^{(2)} \in \mathbb{R}^{4pN}$, is thus a real-valued vector. In summary, the second hidden layer is a non-trainable layer that does not contain any weights but applies a time delay to the complex signal.

In the third hidden layer, we apply the weighted skip connection to $\underline{y}^{(1)}$, which is then added to $\underline{y}^{(2)}$ to produce $\underline{y}^{(3)}$, and hence results the forward propagation equation s.t.

$$\underline{y}^{(3)} = \underline{y}^{(2)} + W^{(3)}.\underline{y}^{(1)}, \tag{8}$$

where $W^{(3)} = \text{diag}\left[w_k\right]_{k=0}^{4pN-1} \in \mathbb{R}^{4pN\times 4pN}$ is a diagonal matrix in which the weights along the diagonal are trained, while all the other weights that are not on the main diagonal remain zero.

2) Structure Imposed Sub-weight Matrices: The StNN features trainable weights in $W^{(1)}$, $W^{(3)}$, and $W^{(4)}$, while the weights in $W^{(2)}$ remain frozen. However, the StNN still exhibits computational complexity of $\mathcal{O}(N^2)$ in generating the beamformer output (i.e. this complexity arises due to fully connected weight matrix-vector multiplications involving $W^{(1)}$ and $W^{(4)}$ with the input vectors in the corresponding layer). To mitigate this and achieve a reduced complexity NN, we impose matrix factorization on the weight matrices, as expressed in Equation (1). For each submatrix i, i.e $w_i^{(1)}$, in $W^{(1)}$, we employ a split factorization based on Equation (1), where the factorization is defined as,

$$w_i^{(1)} = [\check{D}_i]_{2M}[F_i]_{2M}J_{2M\times 2N}[\hat{D}_i]_{2N} \tag{9}$$

for p submatrices. This p submatrices approach is necessary because, when transitioning from the input layer to the first hidden layer, the number of nodes must increase to effectively capture patterns among the input features. The increased number of parameters introduced during the DVM factorization facilitates the input features into a higher-dimensional space within the first hidden layer. A similar factorization strategy is applied to the final layer, utilizing the remaining split DVM factorization from Equation (1). Specifically, we employ p submatrices of

$$w_i^{(4)} = [\hat{D}_i]_{2N}[J_{2M\times 2N}]^T[F_i^*]_{2M}. \tag{10}$$

In summary, we implement the DVM factorization for each p submatrix within the weight matrices, where each submatrix is a product of sparse matrices. The structured p submatrices appear between the input layer and the first hidden layer, having each submatrix with a size of $2M \times M$. Furthermore, there are p submatrices, and each submatrix with a size of $M \times 2M$ appears between the last hidden layer and the output layer. Moreover, the training process of submatrices involves learning parameters that are only located along the diagonals of $\hat{D}_i$ and $\check{D}_i$ matrices, while keeping other values fixed at zero and without updates during backpropagation. Additionally, matrix J within the StNN remains frozen, exempt from training adjustments during backpropagation.

3) Recursive Algorithm for Weight Matrices: In this section, we incorporate the FFT recursion embedded into the DVM for further reduction of complexity. The main objective of this approach is to reduce the adders and gain-delay block counts [3], which leads to the reduction of the complexity of AI-based circuits. For example when $M = 2N$, the matrix F_i appears within the submatrices can be factored as follows:

$$[\tilde{F}_i]_M = P_M \begin{bmatrix} [\tilde{F}_i]_N & 0_N \\ 0_N & [\tilde{F}_i]_N \end{bmatrix} [H_i]_M \tag{11}$$

and

$$[H_i]_M = \begin{bmatrix} I_N & I_N \\ [\widetilde{D}_i]_N & [-\widetilde{D}_i]_N \end{bmatrix}, \tag{12}$$

where $[\tilde{F}_i]_M \approx [F_i]_M$, $\widetilde{D}_i$ is a diagonal weight matrix and P_M is an $M \times M$ even-odd permutation matrix.

Thus, utilizing equations (11) and (12), we can recursively factorize $[\tilde{F}_i]_{2N}$ matrices. Through this recursion, the matrix factorization can be performed up to $[\tilde{F}_i]_2$, resulting in $log(M)$ factorization steps. The determination of factorization steps is based on the performance. Opting for higher steps significantly reduces the weight of the network. However, it may also increase the error of the predicted output due to the reduced number of weights in the NN. Hence, there is a trade-off when tuning the number of recursive factorization steps in the StNN model based on the results. Thus, we need to tune the hyperparameters in the StNN model. In the training process, P_{2N}, O_N, and I_N are fixed matrices with ones and zeros, and those matrices are not updated through backpropagation. During the recursive factorization, we are only updating the matrices $\widetilde{D}_i$ and $[\tilde{F}_i]_2$, at each recursive step. Here, $\widetilde{D}_i$ diagonal matrix at every step is different and independent of each other, and we allowed the neural network to learn weights. The $[\tilde{F}_i]_2$ matrix is trained as a full matrix to yield $[\tilde{F}_i]_{2N}$ In summary, during backpropagation of the recursive factorization, we only need to train a set of diagonal and $[\tilde{F}_i]_2$ matrices. When we update weights through gradient descent, we only update weights that are along the diagonal elements in each diagonal matrix, while leaving the rest of the values as zero.

The overall graphical interpretation of the proposed forward propagation along with the recursion is illustrated in Fig. 3.

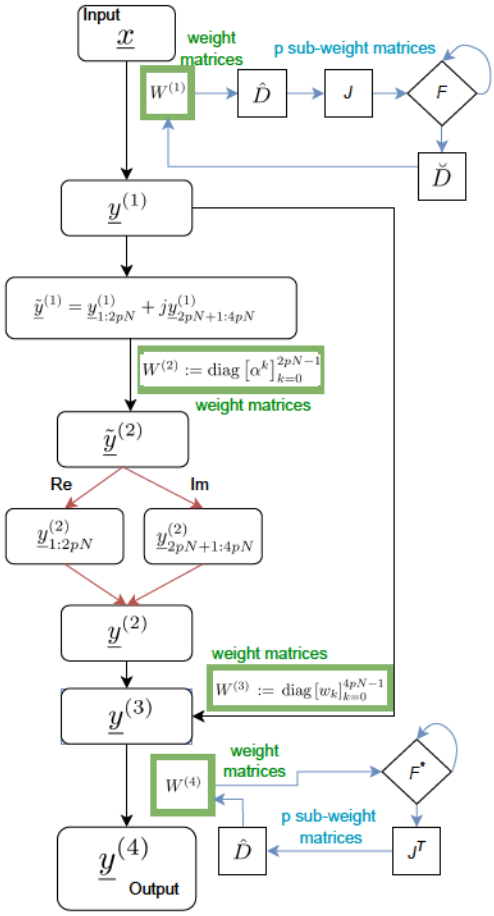


Fig. 3. The forward propagation of the StNN architecture features input $\underline{x}$, output $\underline{y}^{(4)}$, the weight matrices $W^{(1)}$,$W^{(2)}$, $W^{(3)}$, and $W^{(4)}$, the sub-weight matrices $\breve{D}_i$, J, $\hat{D}_i$ complemented by the recursion applied to F_i and F_i^* as shown in equations (11) and (12).

Remark II.1 We note here that another approach to designing a StNN can be found in [38], i.e., a classical algorithm utilized to design layers of NNs while imposing structured weight matrices, to realize states of dynamical systems.

4) Backpropagation of the StNN: The backpropagation process in the StNN follows the standard gradient-based optimization framework, i.e., PyTorch's automatic differentiation engine - Autograd, to compute gradients efficiently. The StNN architecture is implemented in Python using the PyTorch library, where the gradients of all trainable parameters are automatically computed using the framework in sections II-B1, II-B2, II-B3, followed by the backpropagation.

Throughout the training process of the diagonal matrices, only the weights along the diagonals are updated, while all off-diagonal elements remain zero and are frozen. This results in highly sparse weight matrices, significantly reducing the number of trainable parameters while preserving the model's ability to capture essential transformations. We use the Mean Squared Error (MSE) as the loss function (13) to update weights via

$$O(W^{(1)},\cdots,W^{(4)}) = \frac{1}{NM_b}\sum_{s=1}^{M_b}\sum_{k=1}^{N}(y_s^{(k)}-\hat{y}_s^{(k)})^2, \quad (13)$$

where $W^{(1)},\cdots,W^{(4)}$ are defined via (3) and (4 respectively, M_b is the mini-batch size, $y_s^{(k)}$ and $\hat{y}_s^{(k)}$ denote the actual and predicted values at k^{th} antenna index for the s^{th} data sample, respectively.

III. Arithmetic Complexity Analysis

In this section, we present an analysis of the arithmetic complexity of the StNN having an arbitrary input vector $\underline{x} \in \mathbb{R}^{2N}$, where $M := 2N$, which is constructed by extracting real and imaginary parts of the vector $\tilde{\underline{x}} \in \mathbb{R}^N$. In this calculation, we assume that the number of additions $(\#a)$ and multiplications $(\#m)$ required to compute $[F_i]_N$ by an N dimensional vector as Nr and $\frac{1}{2}Nr+\frac{3}{2}N$ [3], respectively, where, $N=2^r (r\geq 1)$.

Proposition III.1 Let StNN be constructed using 5 layers with weight matrices and sub-weight matrices that have the input layer with M nodes, 3 hidden layers with $2pM$ nodes consisting p submatrices per hidden layer, and an output layer with M nodes. Then, the number of additions $(\#a)$ and multiplications $(\#m)$ of the StNN having the input vector $\underline{x} \in \mathbb{R}^M$ to produce the output vector $\underline{y} \in \mathbb{R}^M$ is given via

$$\begin{aligned} \#a(StNN) &= 4prM + 17pM \\ \#m(StNN) &= 2prM + 23pM \end{aligned} \quad (14)$$

where $N=2^r (r\geq 1)$, $M=2N$, and $M >> p$.

Proof. Using the number of additions and multiplication counts in computing the $[F_i]_N$ by an N dimensional vector followed by equations (11) and (12), we can calculate the addition and multiplication counts of the StNN as follows (assuming $\log(M)$ recursive factorization steps), for each sub-weight matrix i,

$$\begin{aligned} \#a(\hat{D}_i) &= 0, & \#m(\hat{D}_i) &= M \\ \#a(J) &= 0, & \#m(J) &= 0 \\ \#a([F_i]_{2M}) &= 2Mr+4M, & \#m([F_i]_{2M}) &= Mr+5M \\ \#a(\breve{D}_i) &= 0, & \#m(\breve{D}_i) &= 2M. \end{aligned}$$

Using the above counts, the arithmetic complexity for each sub-weight matrix $w_i^{(1)}$ and $w_i^{(4)}$ can be computed via

$$\begin{aligned} \#a(w_i^{(1)}) &= 2Mr+4M, & \#a(w_i^{(4)}) &= 2Mr+4M \\ \#m(w_i^{(1)}) &= Mr+8M & \#m(w_i^{(4)}) &= Mr+6M. \end{aligned}$$

We recall that $W^{(1)}$ and $W^{(4)}$ contain the p number of w_i sub-weight matrices. Thus, with p sub-weight matrices, we have $p(2Mr+4M)$ additions and $p(Mr+8M)$ multiplications for $W^{(1)}$ by a vector and $p(2Mr+4M)$ additions and $p(Mr+6M)$ multiplications for $W^{(4)}$ by a vector, respectively. Moreover, arithmetic complexities for computing

diagonal weight matrices $W^{(2)}$ and $W^{(3)}$ by vectors can be computed as follows

$$\begin{aligned} \#a(W^{(2)}) &= 0, & \#a(W^{(3)}) &= 0 \\ \#m(W^{(2)}) &= pM, & \#m(W^{(3)}) &= 2pM. \end{aligned}$$

We note here that, due to the skip connection through Eq. (8), the layer 3 needs an extra addition count of $2pM$. Next, incorporating bias vectors and computing activation introduces $2pM$ additions and multiplications per each hidden layer, and M bias additions and 0 multiplications for the output layer. Additionally, in the output layer, adding the resultant p number of M-sized vectors introduces $(p-1)M$ additions. Therefore, the total number of additions and multiplication counts for the StNN is given via (14). □

Remark III.2 Let a StNN be constructed using L layers, i.e., the input layer with M nodes, $L-2$ hidden layers with $2pM$ nodes consisting p submatrices per hidden layer, and an output layer with M nodes. Then, the number of additions $(\#a)$ and multiplications $(\#m)$ of the StNN having the input vector $\underline{x} \in \mathbb{R}^M$ to produce the output vector $\underline{y} \in \mathbb{R}^M$ is given via

$$\begin{aligned} \#a(StNN) =& prML + \frac{17}{4}pML - prM - \frac{17}{4}pM \\ \#m(StNN) =& \frac{prML}{2} + \frac{23}{4}pML - \frac{prM}{2} - \frac{23}{4}pM \end{aligned} \quad (15)$$

where $N = 2^r (r \geq 1)$, $M = 2N$, and $M >> p$.

Thus, considering the recursive factorization strategy applied to each sub-weight matrix, the computational complexity per single hidden layer is reduced from $\mathcal{O}(M^2)$ to at least $\mathcal{O}(pMlogM)$, having p sub-weight matrices. Thus, given L layers, the overall computational complexity of StNN becomes $\mathcal{O}(pLMlogM)$, offering a significant reduction compared to conventional $\mathcal{O}(M^2L)$ in fully connected networks.

Remark III.3 The MSE results in Section IV show that there is a need to adjust and potentially reduce the number of recursive factorization steps into $\ell (< r) \in \mathbb{N}$ to reduce the MSE values to the order of 10^{-4}. Although the recursive factorization can be used to reduce the number of learnable weight matrices in the StNN, utilizing this can result in an under-parameterized model, especially when the number of weights becomes insufficient to reduce the MSE. To overcome this challenge, we reduce the number of factorization steps (ℓ) as N increases. Additionally, when the factorization runs up to $\log M$ recursive steps, we may encounter a vanishing gradient problem. This issue arises as the last weights in the factorization step (i.e., $[F_i]_2, [F_i]_4$) may not be updated during backpropagation due to very small gradients. However, this can be partially overcome with proper weight initialization techniques [39]. Therefore, reducing factorization steps to ℓ steps allows for improving the overall performance of the model. Hence, computational complexity of the StNN with ℓ recursive steps can be derived from (15) via

$$\#a_\ell(StNN) = p\ell 2^\ell L + \frac{17}{4}pML - p\ell 2^\ell - \frac{17}{4}pM \quad (16)$$

$$\#m_\ell(StNN) = p\ell 2^{\ell-1} L + \frac{23}{4}pML - p\ell 2^{\ell-1} - \frac{23}{4}pM \quad (17)$$

We note here that the optimal number of steps ℓ is determined through empirical evaluation and tuning based on the specific characteristics of the MSE requirement and dataset, as shown in the numerical simulation followed by the Table I and II values in the next Section.

IV. Simulation Results

In this section, we present numerical simulations based on the StNN to realize wideband multi-beam beamformers. The scaled DVM $\tilde{A}_N$ by the input vector $\tilde{x} \in \mathbb{C}^N$ results in the output vector $\tilde{y} \in \mathbb{C}^N$ in the Fourier domain. Thus, we show numerical simulations to assess the accuracy and performance, which are in terms of MSE followed by FLOPs and weight reduction, training time, and inference time, of the StNN model as opposed to FFNNs in realizing wideband multi-beam beamformers.

A. Numerical Setup for Wideband Multi-beam Beamformers

Using an N-element uniform linear array (ULA), we could obtain received signals based on the direction of arrival θ, measured counter-clockwise from the broadside direction. The received signals $u_k(t); k = 1, 2, ...N$ are defined in the complex exponential form, s.t.

$$u_k(t) = e^{-2j\pi f(t-\Delta\tau_k)} + n_s(t), \quad (18)$$

where f is the temporal frequency of the signal, t is the time at which the signal is received, $\Delta\tau_k$ denotes the time delay at the k^{th} element of the antenna array, $n_s(t)$ is complex-valued additive white Gaussian noise (AWGN) with mean 0 and standard deviation of 0.1. Moreover, the time delay $\Delta\tau_k$ is expressed as follows:

$$\Delta\tau_k = \frac{(k-1)d\sin\theta}{c}, \quad (19)$$

where $d = 0.5\lambda$ represents the antenna spacing, λ is the wavelength, c is the speed of light, and θ stands for the angle of arrival. To train the StNN model, we utilized a dataset consisting of the size of $S := 10000$, time-discretized values from $t = 0$ to $t = 1$ for each antenna array. At time t_s, the input vector is determined by the values of $\tilde{x} \in \mathbb{C}^N$, and $k = 1, 2, ..., N$ corresponding to the k^{th} element of the antenna array. Consequently, the data set can be represented as follows.

$$X_{S,N} = \begin{bmatrix} x_1(t_1) & x_2(t_1) & x_3(t_1) & ... & x_N(t_1) \\ x_1(t_2) & x_2(t_2) & x_3(t_2) & ... & x_N(t_2) \\ . & . & . & & . \\ . & . & . & & . \\ . & . & . & & . \\ x_1(t_S) & x_2(t_S) & x_3(t_S) & ... & x_N(t_S) \end{bmatrix}$$

The input vector at time t_s for the StNN can be extracted as $\underline{x}(t_s)$ from each row of $X_{S,N}$, where s is a sample size extracted from S.

$$\underline{x}(t_s) = \begin{bmatrix} x_1(t_s) & x_2(t_s) & x_3(t_s) & ... & x_N(t_s) \end{bmatrix}$$

The StNN is then trained with the values of $\underline{x}(t_s)$ to predict the output $\underline{y}(t_s)$. The output vector $\underline{y}(t_s)$ is computed by multiplying $\underline{x}(t_s)$ by the scaled DVM $\widetilde{A}_N$. The StNN is trained to predict the beamformed vector $\underline{y}(t_s)$ resulting from the multiplication of the input vector $\underline{x}(t_s)$ by the DVM, s.t.

$$\underline{y}(t_s)^T = \widetilde{A}_N \times \underline{x}(t_s)^T, \tag{20}$$

where

$$\underline{y}(t_s) = \begin{bmatrix} y_1(t_s) & y_2(t_s) & y_3(t_s) & \dots & y_N(t_s) \end{bmatrix}.$$

Each element in $\widetilde{A}_N$ can be defined using α's, where $\alpha = e^{-2j\pi f\tau}$. The frequency f is taken as 24 GHz, 27 GHz, and 32 GHz, and the value of τ can be approximately calculated using [3], s.t.

$$\tau = \frac{2\Delta x}{cN} \approx \frac{1}{f_{max}.N}$$

where $\Delta x := d$ is the antenna spacing. In our scenario, f_{max}:=32 GHz represents the maximum frequency of the received signal.

B. Numerical Simulations in Realizing Wideband Multi-beam Beamformers

Here, we discuss the numerical simulations of the StNN to realize wideband multi-beam beamformers. To demonstrate that the StNN model has lower computational complexity compared to FFNNs. We conducted numerical simulations based on the execution of StNN and FFNN while taking FLOPs and weights as quantitative measurements. We standardized the parameters and metrics for both models to ensure a fair comparison. The input layer of both networks comprises $2N$ neurons, representing the real and imaginary parts of elements corresponding to the received signal at each element of the antenna array. Similarly, the output layer has the same number of nodes as the input layer. The compared FFNN includes both a delay layer and a skip connection layer. However, the weight matrices connecting the input layer to the first hidden layer and the last hidden layer to the output layer are both fully connected weight matrices without any structure imposed. We first examine the performance of the StNN for 3 frequencies, i.e., 24GHz, 27GHz, and 32GHz in the range of 24GHz to 32GHz with the receiving signals at 3 different angles (i.e., $\theta = 30, 40$ and 50). We generate 1,000 data samples for each angle, resulting in a total of 3,000 data samples for each frequency. Before training the StNN, we split the dataset into 80% for training and 20% for validation. For each frequency, we train separate StNN models to evaluate their performance. We conducted simulations for three antenna array element sizes: $N = 8$, 16, and 32. Furthermore, since the relationship between the input features and the target variable is relatively straightforward, we have taken a three-hidden-layer architecture as discussed in Section II to capture sufficient underlying patterns. Adding more layers introduces unnecessary complexity, leading the model to struggle with generalization. Moreover, training deeper networks requires more computational resources and time [39]. Therefore, we adhered to the discussed hidden layer architecture while increasing p in each hidden layer for enhanced convergence. All subsequent simulations for StNN and FFNN use the Leaky-Relu activation [40] function with 0.2 scaling factor. During training, we used the MSE as the loss function and the Levenberg-Marquardt algorithm [41] as an optimization function to learn and update the weights. All the numerical simulations were done in Python (version - 3.10) and Pytorch (version - 2.5) framework to design and train the neural networks.

Remark IV.1 To improve readers' comprehension of the theoretical foundation and its relation to the proposed StNN architecture, we encourage readers to access the codes at *Intelligent Wideband Beamforming using StNN*.

The MSE values for the NN predictions in the training and validation sets for the StNN model are shown in Fig. 4. We trained both FFNN and StNN models to reach a minimum MSE value between 10^{-4} and 10^{-3}. Next, we list and compare the accuracy and performance results of both models w.r.t. the quantitative measurements, s.t. MSE, weights, FLOPs, training time, inference time, and validations in Tables I, II, and III. In Table I, the FFNN and StNN models are conceptualized by the model representing numbers, say-(A, B_1, B_2, B_3, C) s.t. A for the number of input nodes, B_i's for the number of hidden nodes, and C for the number of output nodes. Here, p and ℓ denote the number of sub-weight matrices and recursive steps, respectively. The final column of Table I provides values based on the percentage of weight reduction i.e., $Pr(weight)$ Reduction of the StNN compared to the FFNN. The $Pr(weight)$ Reduction is calculated using the formula

$$Pr(weight)\ Reduction = \frac{W_{FFNN} - W_{StNN}}{W_{FFNN}} \times 100\%,$$

where W_{FFNN} and W_{StNN} denote the total trainable weights of FFNN and StNN, respectively. However, as shown in Fig. 4, when training FFNN and StNN models for 1900 steps (i.e., 380 steps per one epoch and training over 5 epochs), they converge to the MSE values of 10^{-10} and 10^{-4}, respectively. This shows that there is a challenge in maintaining complexity and accuracy simultaneously. Thus, to obtain the MSE with an accuracy of 10^{-4}, we trained the StNN for 1900 epochs. The main reason is that FFNN models have more weights, which allows for more flexibility during backpropagation, whereas StNN models have fewer weights with the imposed structure. Table I shows that for smaller sizes of N (i.e., $N = 8, 16$), performing the recursive factorization steps is feasible without compromising accuracy. However, with an increase in N, performing multiple recursive factorization steps results in a significant reduction of weights in the network. Unfortunately, this reduction leads to an increase in the MSE, indicating that the StNN model struggles to capture the patterns between input and output. Consequently, for larger N values, it becomes crucial to decrease the recursive steps (ℓ) to achieve a lower MSE. In summary, both ℓ and p act as hyperparameters that need to be tuned based on accuracy requirements.

TABLE I

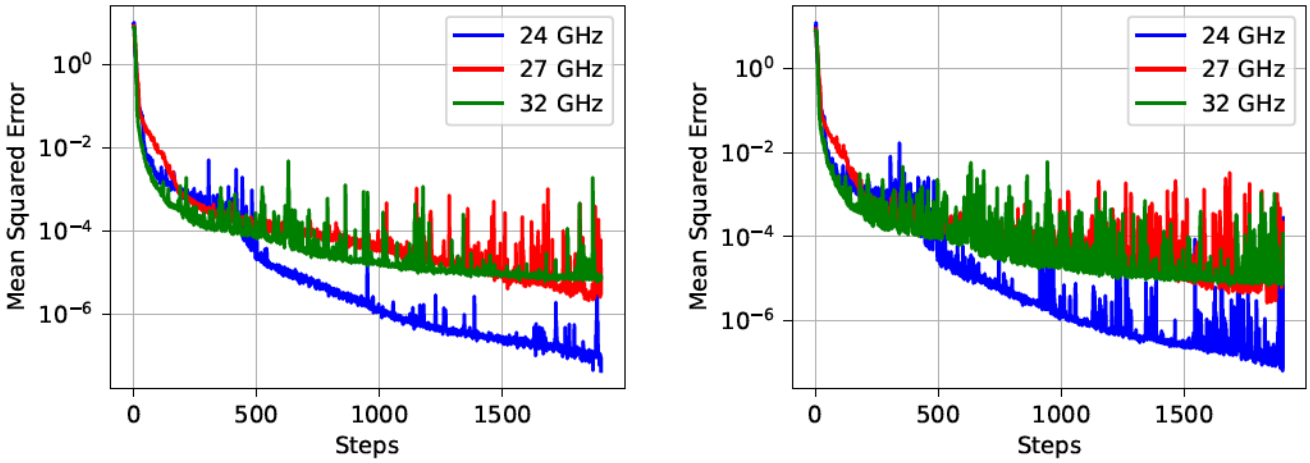
THIS SHOWS MSE VALUES OF STNN AND FFNN HAVING DIFFERENT ANTENNA ARRAY ELEMENTS. THESE VALUES ARE OBTAINED USING CODES WRITTEN IN *Python (Version-3.10)* ALONG WITH THE *Pytorch (version-2.5.1)* FRAMEWORK. THE TERM "MODEL" CONSISTS OF FIVE NUMBERS REPRESENTING NODES IN INPUT, 3-HIDDEN, AND OUTPUT LAYERS(THE FIRST HIDDEN LAYER IS A FULLY CONNECTED LAYER, THE SECOND HIDDEN LAYER IS A DELAY LAYER, THE THIRD IS A SKIP CONNECTION LAYER, AND THE LAST IS ANOTHER FULLY CONNECTED LAYER.). THE NOTATIONS p AND ℓ DENOTE THE NUMBER OF SUB-WEIGHT MATRICES AND RECURSIVE STEPS, RESPECTIVELY. THE LAST COLUMN SHOWS THE PERCENTAGE OF SAVINGS ON UTILIZING STNN OVER FFNN, LEADING TO A LIGHTWEIGHT NN.

N	Model/ Weights(FFNN)	MSE (FFNN)	Model/ p/ ℓ/ Weights(StNN)	MSE (StNN)	Pr(weight) Reduction
8	$(16, 32, 32, 32, 16)/1104$	$(2.8 \pm 0.8) \times 10^{-13}$	$(16, 32, 32, 32, 16)/1/4/220$	$(5.6 \pm 0.2) \times 10^{-8}$	83%
16	$(32, 64, 64, 64, 32)/4256$	$(2.8 \pm 4.1) \times 10^{-12}$	$(32, 64, 64, 64, 32)/1/5/428$	$(2.0 \pm 0.8) \times 10^{-4}$	90%
32	$(64, 128, 128, 128, 64)/16704$	$(3.2 \pm 0.9) \times 10^{-12}$	$(64, 128, 128, 128, 64)/1/6/716$	$(1.0 \pm 3.4) \times 10^{-4}$	96%

TABLE II

ADDITION AND MULTIPLICATION COUNTS(FLOPS) OF THE STNN AND FFNN, I.E., FLOPS: = $\#a(StNN) + \#m(StNN)$, THE PERCENTAGE OF THE SAVINGS ON UTILIZING STNN (EXECUTING $\ell < r$ RECURSIVE STEPS) OVER FFNN, TRAINING AND INFERENCE TIME OF THE STNN AND FFNN.

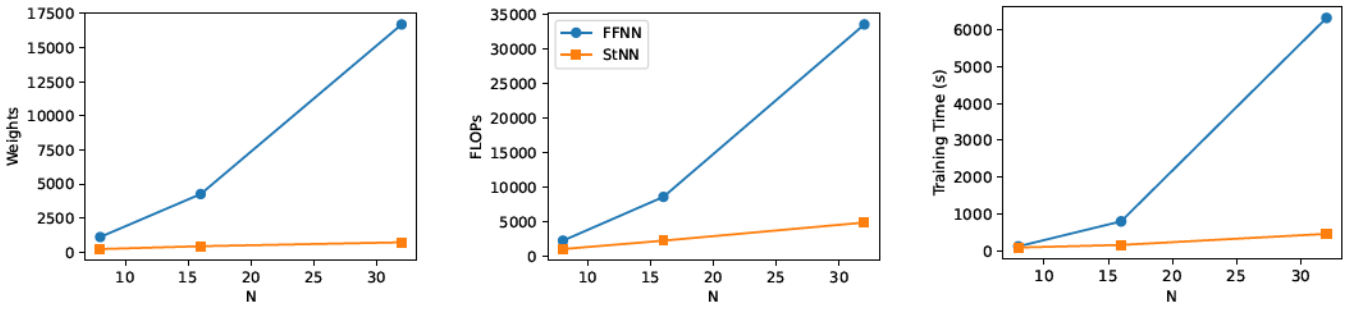
N	FLOPs(FFNN)	FLOPs(StNN) (Eq.(16)+Eq.(17))	Pr(FLOPs) Reduction	Training time(s)(FFNN)	Training time(s)(StNN)	Inference time(ms)(FFNN)	Inference time(ms)(StNN)
8	2240	1024	54%	121	89	0.1984	1.2141
16	8576	2240	74%	799	160	0.2944	2.4063
32	33536	4864	86%	6315	460	0.7621	4.7590



(a) Training performance (b) Validation performance

Fig. 4. The figures (a) and (b) show training and validation results of the StNNs based on the different frequencies (i.e. 24GHz, 27GHz, and 32GHz) for $N = 16$. These graphs are obtained referencing the "Models" listed in Table I. When StNN is executed for 1900 epochs, it converges to MSE values of 10^{-5}. These graphs are obtained using *Python (Version-3.10)* along with the *Pytorch (version-2.5)* framework, and compiled with *Levenberg-Marquardt* optimizer.

The FFNN achieves low MSE values, i.e., order of 10^{-12}, compared with the StNN models having MSE values around 10^{-4}, especially as the depth of recursive factorization steps (ℓ) increases. This is mainly due to the reduced number of trainable parameters and the imposed structure on the weight matrices, which may limit the expressiveness of the model. Despite the MSE to approximately 10^{-4}, the MSE remains adequate for practically realizable wideband mutibeam beamformers. The Table I also highlights a significant trend: as N increases, the StNN model demonstrates a substantial reduction in weights, leading to a significant decrease in FLOPs as shown in Table II. For larger N values, such as 32, the StNN model achieves a 96% reduction in weights with an MSE of 10^{-4} compared to the FFNN model. As N increases, it becomes crucial to adjust the value of p based on the recursive steps ℓ. Increasing more nodes in the hidden layer becomes necessary to reduce MSE and enable the StNN to capture more features [39]. However, the primary advantage of StNN over FFNN is that it has lower FLOPs, lower training time, and lighter weights compared to FFNN, leading to reduced adders and gain-delay block counts for intelligent beamformer realizations. To provide a clearer comparison between the FFNN and StNN models, Fig. 5 summarizes the key performance metrics presented in Tables I and II. As shown, the StNN model requires significantly fewer parameters and computations than the FFNN model, particularly as N increases. *The findings suggest that incorporating more hidden units can further reduce the MSE. However, it is crucial to note*



(a) Training weights vs N (b) Inference FLOPs vs N (c) Training time vs N

Fig. 5. Summary of performance metrics comparing FFNN and StNN models across varying values of N. (a) total number of trainable weights, (b) inference FLOPs, and (c) total training time. StNN demonstrates lower model complexity and computational cost compared to FFNN, especially at higher values of N.

that larger p values may lead to overfitting, highlighting the importance of selecting the optimal p value based on the given N. In summary, the MSE trade-off is acceptable, particularly in scenarios where minimizing computational complexity and intelligent hardware complexity is a major concern.

Furthermore, Table II shows the percentage FLOP reduction of the StNN, which is at least 54% to at most 86%. More importantly, the FLOPs will be further reduced as the number of antenna array elements increases, showing the feasibility of adopting the proposed StNN in reducing the computational demand. In addition to FLOP reductions, StNN also significantly reduces training time, showing a nearly 14-fold improvement or 93% reduction in training time for 32-element antenna arrays due to the smaller number of training parameters and structure-imposed weight and sub-weight matrices within the DVM factorization imposed for the design of StNN.

Although the current inference observed for the StNN appears to be higher than that of the FFNN, despite the lower FLOPs, training time, and weight counts. This difference is largely due to implementation factors; the FFNN inference was performed using PyTorch's highly optimized modules that leverage parallel computation and hardware acceleration. In contrast, the current implementation of the StNN does not incorporate parallelism or hardware-specific optimizations. However, the theoretical reduction of FLOPs, training time, and weight counts remains consistently lower than that of the FFNN. With further optimization, including GPU-level parallel processing or FPGA deployment, the inference time of StNN is expected to align more closely with the FFNN. To ensure a fair comparison, a real-world deployment and hardware-specific benchmarking of inference time, such as using an FPGA or a GPU, would be essential. Nevertheless, the theoretical FLOPs, inference time, and weight reductions indicate that StNN's low-power intelligence chips could be an efficient solution for real-time applications.

We note here that our previous work on the S-LSTM network for multi-beam beamformers [36], saved 30% of training weights to achieve an MSE of $8\times10^{(-2)}$ for $N = 16$ elements antenna array. Although the S-LSTM approach outperformed conventional LSTM beamforming algorithms, the complexity of the S-LSTM remained relatively high due to the large number of parameters as opposed to the StNN. Thus, in this paper, we showed that the StNN reduces 90% of training parameters compared to FFNN, achieving a significantly lower MSE of $2 \times 10^{(-4)}$. These results demonstrate that the proposed approach is generalizable to intelligent wideband multibeam beamformers with lower computational overhead. In particular, the FLOPs presented in Table I and II are closely tied to the logic resource requirements in hardware such as FPGAs. Since FPGAs are resource-constrained platforms, especially in terms of DSP slices and logic utilization, models with significantly fewer FLOPs, such as the proposed StNN, are inherently more suitable for efficient implementation. The reduction in FLOPs directly translates to a reduced number of adders and gain-delay block counts, leading to lower power consumption, less resource usage, and potentially higher throughput in FPGA-based intelligent beamforming systems.

Furthermore, the ability to achieve such performance gains with reduced weights and FLoP counts opens pathways for deploying AI-driven wideband multi-beam beamformers in resource-constrained environments. Future work will explore the applicability of this approach to larger antenna array elements, i.e., 128, 256, 512, as well as its adaptability to intelligent signal delaying in nonlinear and time-varying beamforming scenarios.

C. Evaluation Under Fading and SNR Variations

To assess the robustness of the proposed StNN architecture under practical channel impairments, we augmented the simulation to include Rayleigh fading while varying signal-to-noise ratios (SNRs). More specifically, for a ULA with $N = 16$ antenna elements and carrier frequency $f_c = 27\,\text{GHz}$, we modeled the baseband received signal at the k-th antenna as:

$$u_k(t) = h_k \cdot e^{j(2\pi f_c t + \phi_k)} + n_k(t),$$

where $h_k \sim \mathcal{CN}(0,1)$ represents a single-tap complex Rayleigh fading coefficient for the k-th antenna, $\phi_k = 2\pi \cdot \frac{d_x}{\lambda} \cdot k \cdot \sin(\theta)$ denotes the deterministic phase offset due to angle of arrival θ, $n_k(t) \sim \mathcal{CN}(0, \sigma^2)$ is AWGN, scaled according to the desired SNR level.

The inter-element spacing was set to half the minimum wavelength in the operating band, i.e., $d_x = \lambda_{\min}/2$ with $\lambda_{\min} = c/f_{\max}$. A time vector of 1000 samples over 1 microsecond duration was used to simulate signals, which

were passed through a Rayleigh fading channel and corrupted with noise corresponding to SNR levels ranging from 10 dB to 30 dB. The composite noisy received signal vector $\tilde{\mathbf{x}}(t) \in \mathbb{C}^N$ was then processed through the TTD DVM beamformer to store the input and output signals to train the StNN model.

TABLE III
THE VALIDATION MSE OF THE STNN IN RAYLEIGH FADING CONDITIONS ACROSS VARIOUS SNR LEVELS IS PRESENTED. THIS ALSO CONSIDERS THE WAVELENGTH λ_{min} FOR A 16-ELEMENT ANTENNA ARRAY, AS WELL AS THE PERCENTAGE REDUCTION IN BOTH FLOPS AND WEIGHTS FOR THE SUB-WEIGHT MATRICES SETTING OF $p = 1$.

SNR (dB)	ℓ	Validation MSE StNN	Pr(weights) Reduction	Pr(FLOPs) Reduction
30	5	7.24×10^{-3}	90%	86%
30	3	3.79×10^{-3}	85%	63%
30	2	9.20×10^{-8}	67%	55%
20	5	4.03×10^{-2}	90%	86%
20	3	1.04×10^{-2}	85%	63%
20	2	7.92×10^{-7}	67%	55%
10	5	7.40×10^{-1}	90%	86%
10	3	3.13×10^{-2}	85%	63%
10	2	3.99×10^{-7}	67%	55%

Table III presents the validation MSE results of StNN models trained with distinct values of the recursive factorization parameter ℓ, all in the configuration $p = 1$. The results indicate that at higher SNR levels, larger values of ℓ yield acceptable performance. However, under lower SNR conditions, the StNN requires a higher number of trainable parameters within the weight matrices to effectively capture environmental variations and produce accurate beamformer outputs. Therefore, ℓ must be reduced, which eventually leads to the reduction of the number of recursive factorization steps. This leads to a lesser reduction in weight and FLOP savings compared to the FFNN baseline. However, StNN is still able to produce accurate MSE, saving 67% fewer weights and almost 50% fewer FLOPs compared to FFNN. This confirms that the proposed StNN maintains robust performance even in the presence of multipath fading and degraded SNR scenarios.

V. CONCLUSION

We introduced a novel structured neural network (StNN) to intelligently realize wideband multi-beam beamformers utilizing structured weight matrices and submatrices. The proposed StNN leverages the factorization of the DVM in our previous work to reduce the computational complexities of matrix-vector computations in the layers of neural networks. Numerical simulations show that StNN-based wideband multi-beam beamformers can be implemented with fewer FLOPs, lower training time, and a lightweight neural network, achieving an accuracy of 10^{-4} to intelligently realize beamformers. Even as the number of elements in antenna arrays increases, the FLOPs, weights, and inference time are significantly reduced while maintaining accuracy order of 10^{-4}. This makes a compelling argument for the feasibility of adopting the StNN for power-hungry intelligent chip design. Simulation within the range of 24 GHz to 32 GHz shows that the StNN can be utilized to accurately realize wideband multi-beam beamformers as opposed to the conventional fully connected neural network with the complexity reduction from $\mathcal{O}(M^2L)$ to $\mathcal{O}(pL\ M \log M)$, where M is the number of nodes in each layer of the network, p is the number of submatrices per layer, and $M >> p$. Numerical simulations conducted within the 24 GHz to 32 GHz range have shown that the proposed structured neural architecture can be efficiently, accurately, and intelligently utilized to realize wideband multi-beam beamformers in a robust environment.

REFERENCES

[1] H. L. Van Trees, *Optimum array processing: Part IV of detection, estimation, and modulation theory*. John Wiley & Sons, 2002.

[2] S. M. Perera, V. Ariyarathna, N. Udayanga, A. Madanayake, G. Wu, L. Belostotski, Y. Wang, S. Mandal, R. J. Cintra, and T. S. Rappaport, "Wideband n-beam arrays using low-complexity algorithms and mixed-signal integrated circuits," *IEEE Journal of Selected Topics in Signal Processing*, vol. 12, no. 2, pp. 368–382, 2018.

[3] S. M. Perera, L. Lingsch, A. Madanayake, S. Mandal, and N. Mastronardi, "A fast dvm algorithm for wideband time-delay multi-beam beamformers," *IEEE Transactions on Signal Processing*, vol. 70, pp. 5913–5925, 2022.

[4] S. M. Perera, A. Madanayake, and R. J. Cintra, "Radix-2 self-recursive sparse factorizations of delay vandermonde matrices for wideband multi-beam antenna arrays," *IEEE Access*, vol. 8, pp. 25 498–25 508, 2020.

[5] ——, "Efficient and self-recursive delay vandermonde algorithm for multi-beam antenna arrays," *IEEE Open Journal of Signal Processing*, vol. 1, pp. 64–76, 2020.

[6] H. Al Kassir, Z. D. Zaharis, P. I. Lazaridis, N. V. Kantartzis, T. V. Yioultsis, I. P. Chochliouros, A. Mihovska, and T. D. Xenos, "Antenna array beamforming based on deep learning neural network architectures," in *2022 3rd URSI Atlantic and Asia Pacific Radio Science Meeting (AT-AP-RASC)*, 2022, pp. 1–4.

[7] 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 Transactions on Broadcasting*, vol. 59, no. 3, pp. 455–460, 2013.

[8] Z. D. Zaharis, T. V. Yioultsis, C. Skeberis, T. D. Xenos, P. I. Lazaridis, G. Mastorakis, and C. X. Mavromoustakis, "Implementation of antenna array beamforming by using a novel neural network structure," in *2016 International Conference on Telecommunications and Multimedia (TEMU)*. IEEE, 2016, pp. 1–5.

[9] A. H. Sallomi and S. Ahmed, "Multi-layer feed forward neural network application in adaptive beamforming of smart antenna system," in *2016 Al-Sadeq International Conference on Multidisciplinary in IT and Communication Science and Applications (AIC-MITCSA)*. IEEE, 2016, pp. 1–6.

[10] G. Castaldi, V. Galdi, and G. Gerini, "Evaluation of a neural-network-based adaptive beamforming scheme with magnitude-only constraints," *Progress In Electromagnetics Research B*, vol. 11, pp. 1–14, 2009.

[11] I. T. Cummings, T. J. Schulz, T. C. Havens, and J. P. Doane, "Neural networks for real-time adaptive beamforming in simultaneous transmit and receive digital phased arrays: Student submission," in *2019 IEEE International Symposium on Phased Array System & Technology (PAST)*. IEEE, 2019, pp. 1–8.

[12] X. Wu, J. Luo, G. Li, S. Zhang, and W. Sheng, "Fast wideband beamforming using convolutional neural network," *Remote Sensing*, vol. 15, no. 3, 2023. [Online]. Available: https://www.mdpi.com/2072-4292/15/3/712

[13] Z. Liao, K. Duan, J. He, Z. Qiu, and B. Li, "Robust adaptive beamforming based on a convolutional neural network," *Electronics*, vol. 12, no. 12, p. 2751, 2023.

[14] S. Bianco, P. Napoletano, A. Raimondi, M. Feo, G. Petraglia, and P. Vinetti, "Aesa adaptive beamforming using deep learning," in *2020 IEEE Radar Conference (RadarConf20)*, 2020, pp. 1–6.

[15] H. Huang, Y. Peng, J. Yang, W. Xia, and G. Gui, "Fast beamforming design via deep learning," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 1, pp. 1065–1069, 2019.

[16] P. Ramezanpour and M.-R. Mosavi, "Two-stage beamforming for rejecting interferences using deep neural networks," *IEEE Systems Journal*, vol. 15, no. 3, pp. 4439–4447, 2021.

[17] T. Sallam and A. M. Attiya, "Convolutional neural network for 2d adaptive beamforming of phased array antennas with robustness to array imperfections," *International Journal of Microwave and Wireless Technologies*, vol. 13, no. 10, pp. 1096–1102, 2021.

[18] W. Xia, G. Zheng, Y. Zhu, J. Zhang, J. Wang, and A. P. Petropulu, "Deep learning based beamforming neural networks in downlink miso systems," in *2019 IEEE International Conference on Communications Workshops (ICC Workshops)*, 2019, pp. 1–5.

[19] T. Lin and Y. Zhu, "Beamforming design for large-scale antenna arrays using deep learning," *IEEE Wireless Communications Letters*, vol. 9, no. 1, pp. 103–107, 2020.

[20] R. Lovato and X. Gong, "Phased antenna array beamforming using convolutional neural networks," in *2019 IEEE International Symposium on Antennas and Propagation and USNC-URSI Radio Science Meeting*, 2019, pp. 1247–1248.

[21] I. Mallioras, Z. D. Zaharis, P. I. Lazaridis, and S. Pantelopoulos, "A novel realistic approach of adaptive beamforming based on deep neural networks," *IEEE Transactions on Antennas and Propagation*, vol. 70, no. 10, pp. 8833–8848, 2022.

[22] H. Che, C. Li, X. He, and T. Huang, "A recurrent neural network for adaptive beamforming and array correction," *Neural Networks*, vol. 80, pp. 110–117, 2016.

[23] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[24] M. Kissel and K. Diepold, "Structured matrices and their application in neural networks: A survey," *New Generation Computing*, vol. 41, no. 3, pp. 697–722, 2023.

[25] J. Feng and T. Darrell, "Learning the structure of deep convolutional networks," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 2749–2757.

[26] Y. Gong, L. Liu, M. Yang, and L. Bourdev, "Compressing deep convolutional networks using vector quantization," *arXiv preprint arXiv:1412.6115*, 2014.

[27] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[28] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," *Advances in neural information processing systems*, vol. 29, 2016.

[29] L. Zhao, S. Liao, Y. Wang, Z. Li, J. Tang, and B. Yuan, "Theoretical properties for neural networks with weight matrices of low displacement rank," in *international conference on machine learning*. PMLR, 2017, pp. 4082–4090.

[30] S. R. Kamalakara, A. Locatelli, B. Venkitesh, J. Ba, Y. Gal, and A. N. Gomez, "Exploring low rank training of deep neural networks," *arXiv preprint arXiv:2209.13569*, 2022.

[31] L. Lingsch, M. Michelis, E. de Bezenac, S. M. Perera, R. K. Katzschmann, and S. Mishra, "Beyond regular grids: Fourier-based neural operators on arbitrary domains," in *the International Conference on Machine Learning, PMLR 235*, 2024, pp. 30 610–30 629.

[32] S. Liao and B. Yuan, "Circconv: A structured convolution with low complexity," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 4287–4294.

[33] T. N. Sainath, B. Kingsbury, V. Sindhwani, E. Arisoy, and B. Ramabhadran, "Low-rank matrix factorization for deep neural network training with high-dimensional output targets," in *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE, 2013, pp. 6655–6659.

[34] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up convolutional neural networks with low rank expansions," *arXiv preprint arXiv:1405.3866*, 2014.

[35] S. Anwar, K. Hwang, and W. Sung, "Structured pruning of deep convolutional neural networks," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 13, no. 3, pp. 1–18, 2017.

[36] H. Aluvihare, C. Shanahan, S. M. Perera, S. Sivasankar, U. Kumarasiri, A. Madanayake, and X. Li, "A low-complexity lstm network to realize multibeam beamforming," in *2024 IEEE International Conference on Wireless for Space and Extreme Environments (WiSEE)*, 2024, pp. 11–16.

[37] T. Kailath and V. V. Olshevsky, "Displacement structure approach to polynomial vandermonde and related matrices," *Linear Algebra Appl.*, vol. 261, pp. 49–90, 1997.

[38] H. Aluvihare, L. Lingsch, X. Li, and S. M. Perera, "A low-complexity structured neural network to realize states of dynamical systems," *in review, SIAM Journal on Applied Dynamical Systems*, 2025.

[39] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[40] A. L. Maas, A. Y. Hannun, A. Y. Ng *et al.*, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. icml*, vol. 30, no. 1. Atlanta, GA, 2013, p. 3.

[41] J. J. Moré, "The levenberg-marquardt algorithm: implementation and theory," in *Numerical analysis: proceedings of the biennial Conference held at Dundee, June 28–July 1, 1977*. Springer, 2006, pp. 105–116.