A Review of Millimeter Wave Phased Array Antenna Synthesis Using Gradient Boosted and Random Forest Techniques

Alex Hostick

Department of Electrical and Computer Engineering, University of New Mexico

ECE551: Problems in Machine Learning

Dr. Christos Christodoulou

23 July 2024

Abstract

Over the last decade, international telecommunications providers have evolved to bring data to exceptional speed and bandwidth capabilities. Individuals are not the only market that relies on telecom services for personal internet and phone access. Businesses and industries are shifting to smart factories, large broadcast events, gaming hubs, and IoT services requiring a non-cabled solution. This paper aims to provide insight into the current and future state of telecom services using phased array antennas to broadcast urban or aerial solutions with K-Band signals, improving bandwidth and data transfer with additional dynamic reconfiguration options based on the requirement.

This paper reviews the 2020 research paper "Millimeter Wave Phased Array Antenna Synthesis Using a Machine Learning Technique for Different 5G Applications" written at the Institute for Communication System (ICS), Home of the 5G Innovation Center (5GIC) University of Surrey, Guildford, GU2 7XH, UK by Danesh, S., et al. The research team created a machine-learning model using a phase synthesis-only approach. Additionally, the approach creates a far-field pattern based on the combination of array factor and element pattern while using the machine learning model to synthesize and optimize the result, all while using low cost and low complexity. MATLAB's similar machine-learning functions were tested against the paper's model complexity and accuracy for comparison. Finally, this research endeavor enhanced my knowledge of machine learning models and helped me understand how the industry uses phased array antennas within telecom services.

A Review of Millimeter Wave Phased Array Antenna Synthesis Using Multiple Machine Learning Techniques

Phased array antennas offer dynamic radio frequency transmission with multiple individual antenna elements that can transform or coalesce a beam pattern. One can control the individual elements in the array to electronically steer the direction and the beam formation by applying mathematical calculations. According to Guney and Onay (2007), "one efficient method based on bees algorithm (BA) for the pattern, which is an amplitude-only synthesis for antenna array optimization." A team from the Institute for Communication System (ICS), Home of the 5G Innovation Center (5GIC), University of Surrey, Guildford, created a machine learning technique to estimate the desired radiation patterns by applying phase-only synthesis. The team's concern is that genetic algorithm (GA) approaches have been used for phase-only synthesis problems and have no guarantee to converge or find the optimum solution while adding additional complexity. This paper reviews the "Millimeter Wave Phased Array Antenna Synthesis Using a Machine Learning Technique for Different 5G Applications" technique by Shadi Danesh, et al. The paper also includes a high-level review of phased array antennas. It compares the gradient boosted tree (GBT) method the ICS's team used against the accuracy and complexity of MATLAB GBT and Random Forest algorithms.

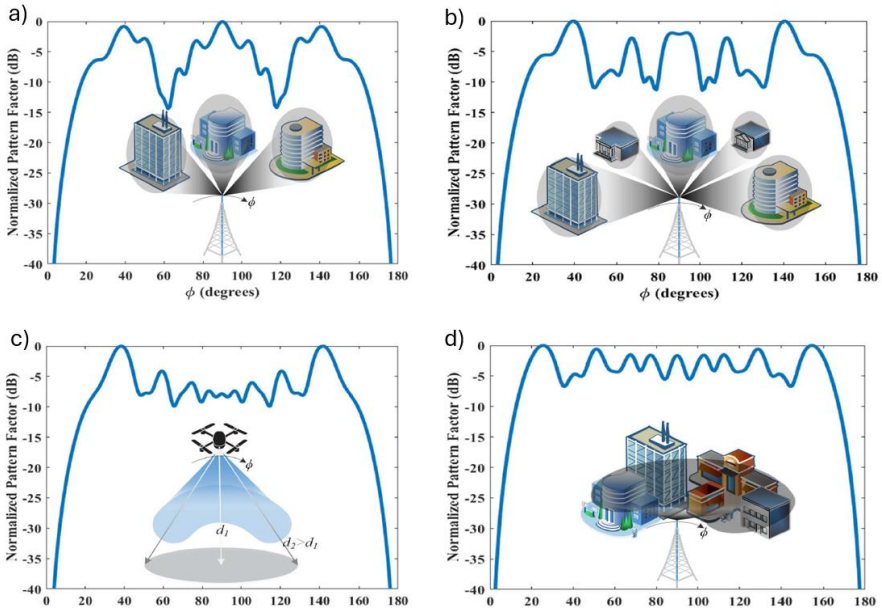
Within the paper, the authors detail the need for millimeter wave (mmWave) techniques for 5G networks. The 5G network, still under completion in 2020, proposed a 26 GHz band for the European telecommunication band and a 28 GHz band for the 5G network in the US. The network bands, stated by the paper from 2016 to 2017, reference the national 5G deployments in Europe's LTE 5G network has K-Band frequency ranges of 24.25-27.5 GHz and the US

frequency range of 27.5-28.35 GHz (5gmmwave.com). Specifically, Europe's K-Band 5G network frequencies are labeled the n258 and the n257 band for the US.

High-frequency bands are not employed without risk. Frequencies in the K-Band cannot travel a considerable distance compared to the sub 6 GHz but offer greater data throughput, lower latencies, and greater significance to first responders and municipalities. Telecom K-band frequencies are typically integrated into urban areas, enhancing mobile telecom services within a short broadcast range. Manufacturers of K-Band-enabled microchips are promoting the band to target mobile broadband, fixed wireless access, industrial IoT, and smart factories for Industry 4.0 applications (Ericsson.com, n.d.). In the new era of broadband services, industry can enjoy the speed of a K-band signal network relative to a standard wired solution. The author's paper has various use cases for the waveforms, as noted in Fig. 1.

Fig. 1

Paper's example use cases of (a) three-beam patterns in Unicast, (b) five-beam patterns in Unicast, (c) saddle-shaped beam for drones in Multicast, (d) widebeam usage for Multicast



Phase Only Synthesis Approach

The authors of the paper detail the use of phase-only synthesis for deploying unicast and multicast applications for service delivery. Unicast can create unique streams required for each user, with high throughput and low latency. Multicast can be used for broadcast events, such as live TV, live events, and radio broadcasts. Therefore, unicast is typically used for one-to-one communication networks while multicast gives your network the ability to broadcast messages on a wide scale (cbtnuggets.com, n.d.). The phase-only synthesis approach helps determine the optimal set of phase values, creating a multibeam pattern. Phased array antennas can also reconfigure their steering angle or pattern based on regional requirements.

An array response is based on the array structure, separation distance between elements, element weights, and the excitation phases (Rosu, n.d, pp. 9-11). Together, these features are called the array factor (AF). For the referenced linear array with M radiator elements, the AF calculated by

$$AF = \sum_{n=1}^{M} w_n e^{j(n-1)\psi_n}$$

where w_n is the element weight and ψ_n is the excitation phase due to element position and observation direction. The author states the far-field pattern can be synthesized by using the pattern factor (PF) combination. Pattern factors are calculated by taking the array factor (AF) by the far-field radiation pattern of a single radiator element as

$$Pattern\ Factor\ (PF) = Element\ Pattern\ (EP)\ \times AF$$

The author created several different beam patterns based on the phases in Table I. The phases were made from characteristics of a known 16-element phased array. Additionally, normalized 3D Radiation patterns were made by the research team and noted in Fig. 2.

Fig. 2

Author's 3D normalized radiation pattern factor in dB of (a) three-beam, (b) five-beam, (c) saddle-shaped beam, (d) widebeam, (e) boresight pattern for $i = 0$

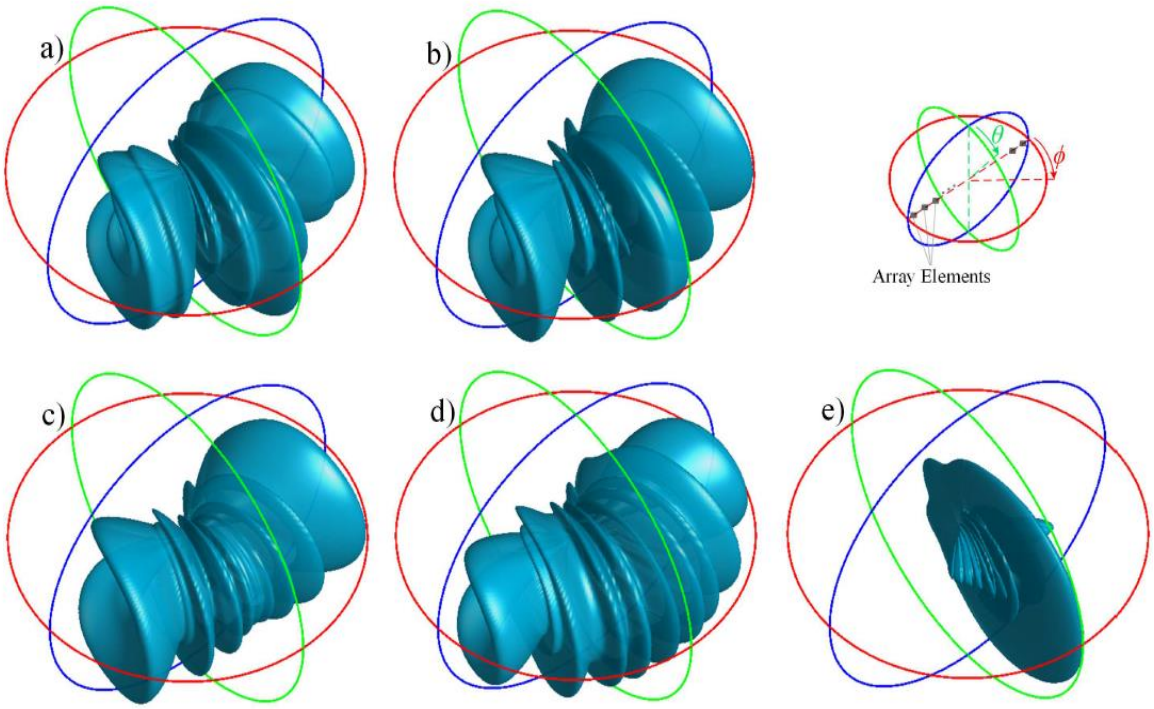


Table I

Author's Predicted 16 Phase Values for Different Radiation Patterns

Estimated Phases	Three-Beam Pattern	Five-Beam Pattern	Saddle-Shaped Pattern	WideBeam Array Pattern
ψ_1	0	0	0	0
ψ_2	142.32847184	143.33452399	136.77605927	184.76658412
ψ_3	281.99939286	282.62453362	273.55211855	309.52055325
ψ_4	435.69848803	475.69848803	411.77337706	500.89647463
ψ_5	562.63272243	561.77552221	547.10423709	645.04110651
ψ_6	719.28436561	720.10224530	686.28896177	803.8512703
ψ_7	801.60312986	891.60312986	883.54675412	1120.85281386
ψ_8	436.89813568	496.89813568	439.64520644	666.78612754

ψ_9	436.99478245	496.99513568	439.64520644	666.78612754
ψ_{10}	801.74963177	891.74312986	883.54675412	1120.85281386
ψ_{11}	719.32898648	720.11392745	686.28896177	803.8512703
ψ_{12}	562.87651446	561.63986522	547.10423709	645.04110651
ψ_{13}	435.37910201	475.37910201	547.10423709	500.89647463
ψ_{14}	281.42939179	282.91291166	273.55211855	309.52055325
ψ_{15}	142.52491657	143.84732378	136.77605927	309.52055325
ψ_{16}	0	0	0	0

Table I is the base pattern for comparing the GBT model in the paper. From Table I, we can calculate the AF, and therefore the PF, for each of the proposed far field patterns. The PF for each pattern is noted in Table II.

Table II

PF calculated for each beam pattern in Table I

Estimated Phases	Three-Beam Pattern	Five-Beam Pattern	Saddle-Shaped Pattern	WideBeam Array Pattern
PF	4.86225803	3.862624768	2.121360053	3.244233629

The paper did not mention details of the phased array antenna other than its linear configuration, 16 elements, and the 26 GHz operating frequency. It was assumed that the elements were spaced half wavelength apart (λ/2), and no tapering or filtering was applied when modeling in MATLAB. The generalized MATLAB equivalent, taking the frequency, element spacing, and linear configuration, generated the 3D radiation patterns noted in Fig. 3. Setup of the antenna also had all elements using identical amplitudes, or $\forall(n \in M)[w_n = 1]$, with only the phase varying per element. A general view of how the elements were spaced and the excitation phase is illustrated in Fig. 4.

Fig. 3

MATLAB 3D Comparison of author's 3D normalized radiation pattern factor in dB of (a) three-beam, (b) five-beam, (c) saddle-shaped beam, (d) widebeam, (e) boresight pattern for $i = 0$

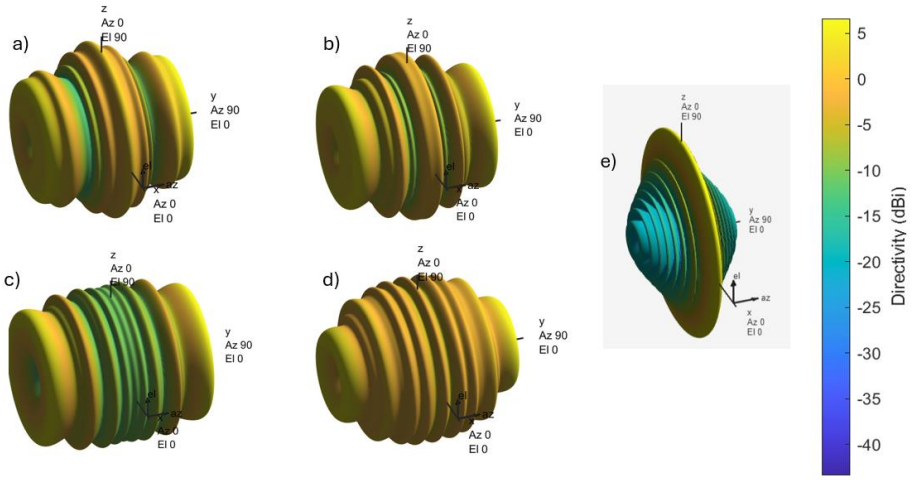
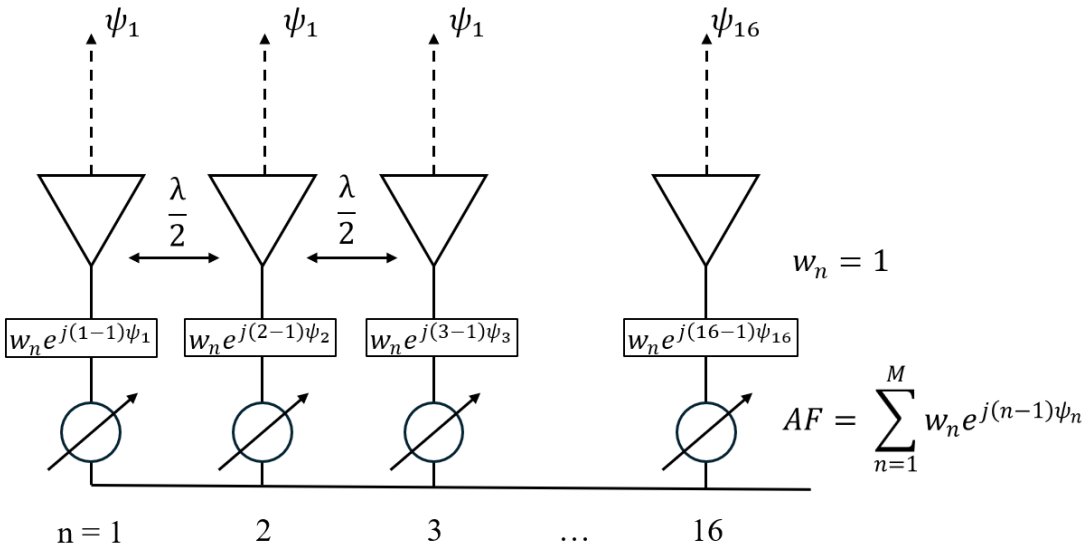


Fig. 4

Setup of MATLAB Linear Antenna Array



Pre-Processing of Data

Data emulating the paper's pre-processing became an essential step in optimizing the machine-learning models within the paper. Antennas typically have RF propagation properties

noted on azimuth (θ) and elevation (φ) sheets provided by the manufacturer. Five thousand progressive pattern factors and their subsequent phases were generated with five thousand random pattern factors and their subsequent phases to parallel the research team's data set. Then, the power factors were randomly shuffled into an overarching data set to make 10,000 samples. As noted in Fig. 5, the data separated into the first two PCA components shows how complex a support linear regression model may need to be for the data. Fig. 6 notes the pattern factors in the data series and how the random pattern factors add noise to the data set.

Fig. 5

First and Second Principal Components of the Three-Beam Data Set

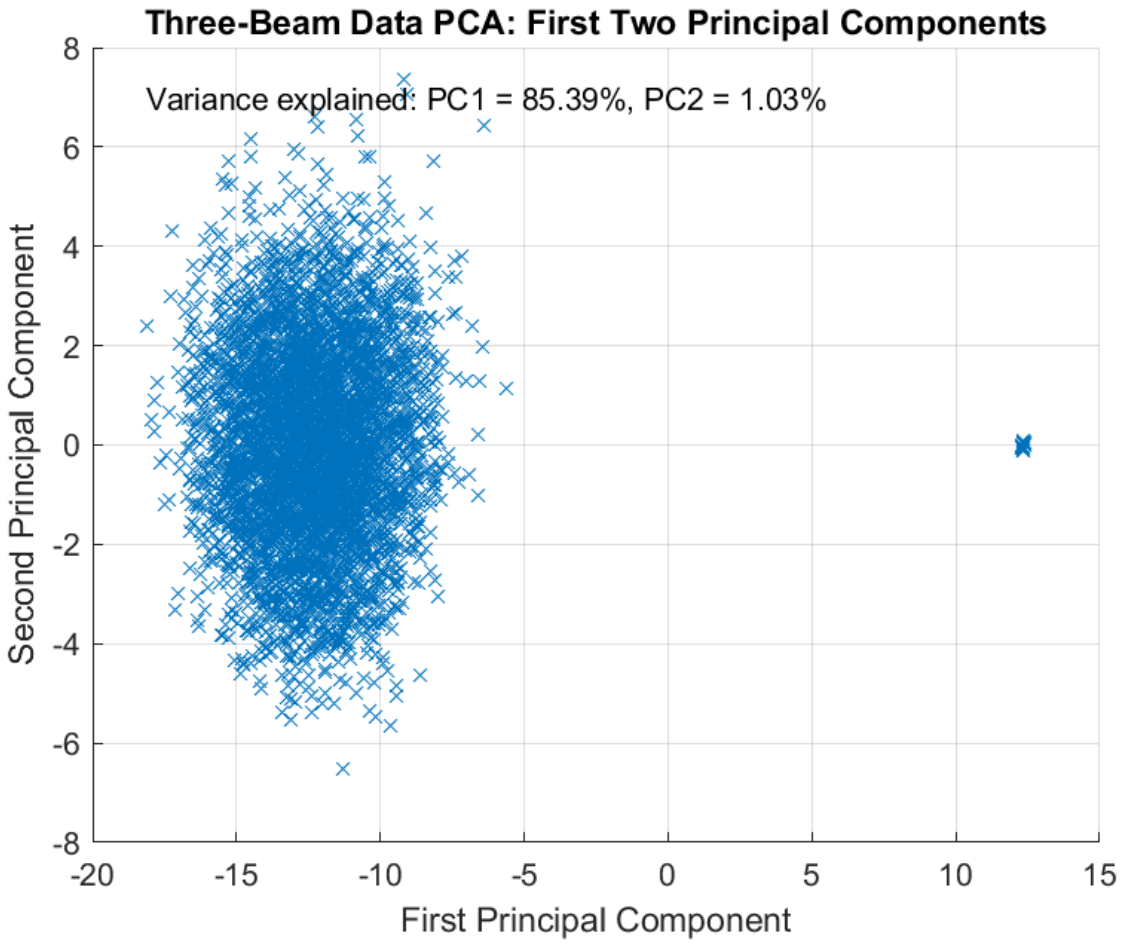
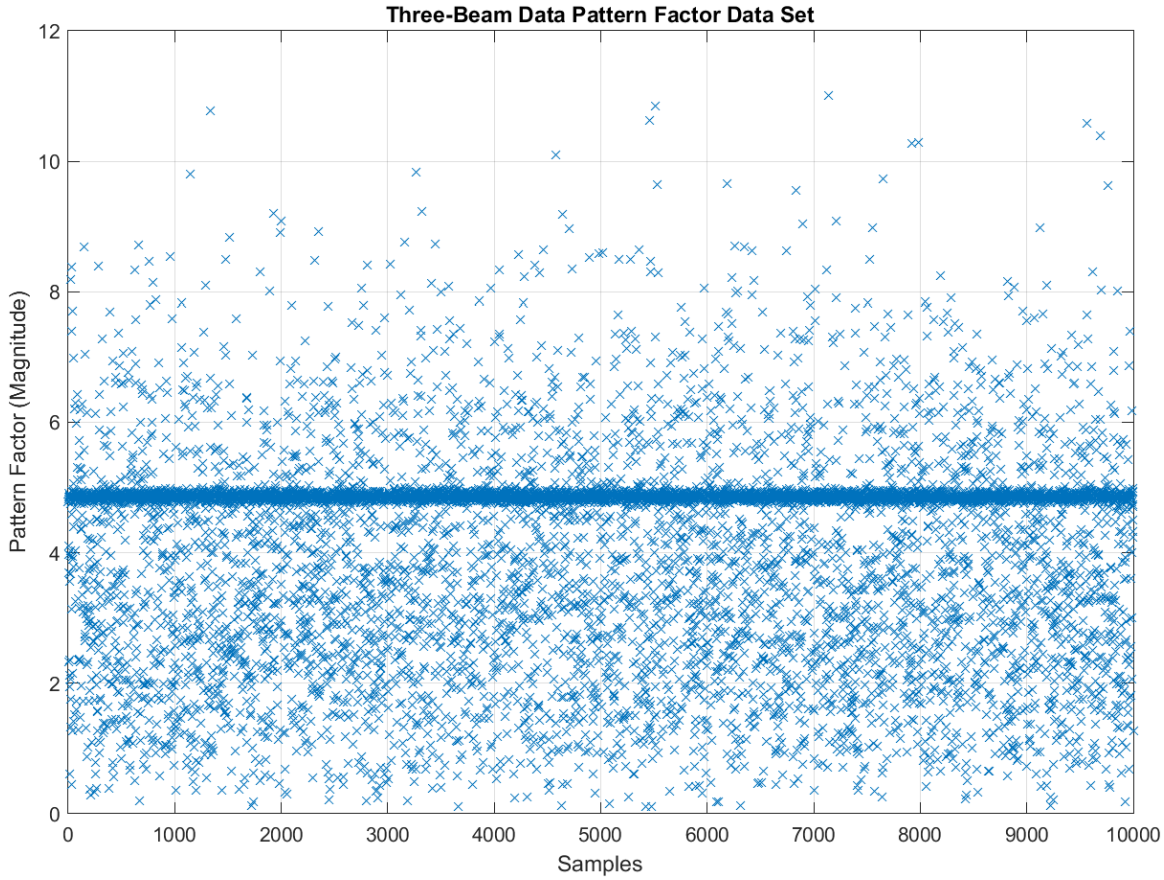


Fig. 6

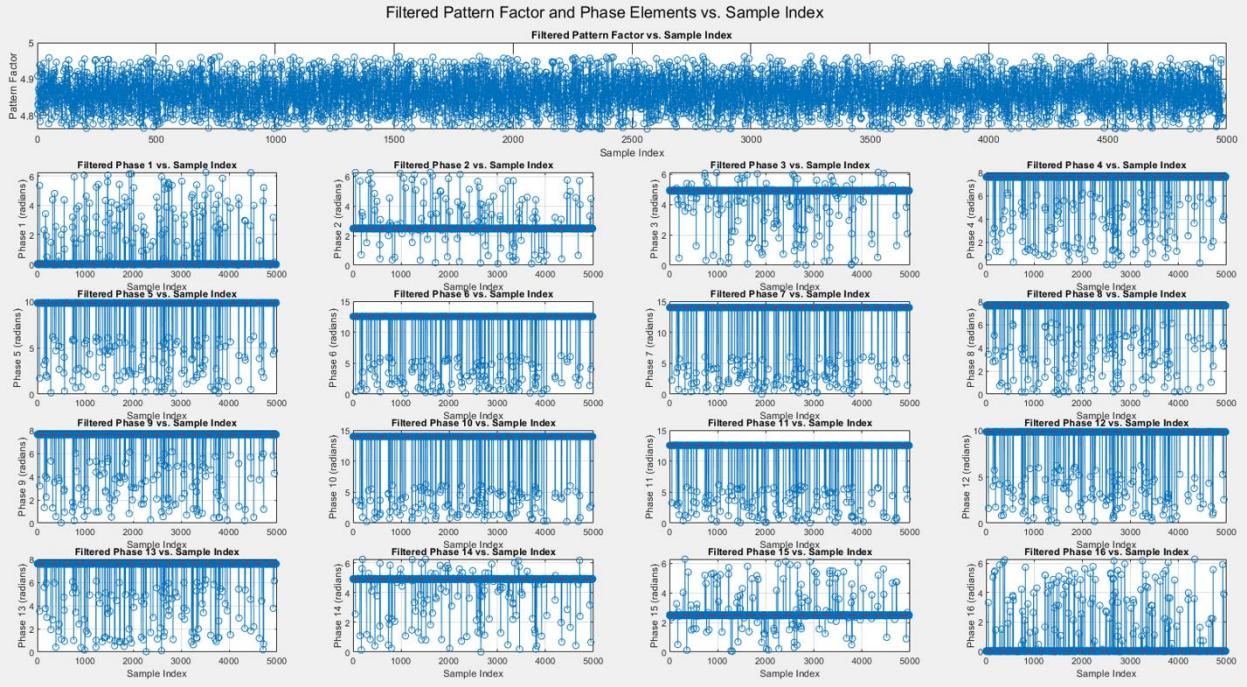
Three-Beam Pattern Factor Data After Creation and Shuffling with Random Data



After creating the data, the authors noted that the model should learn from relevant data by correlating pattern factors and element phase data sets with a 90% or higher similarity. Correlating and filtering non-matched data decreased the noise dramatically, allowing the model to train on pattern factors closest to the values in Table II. Fig. 7 notes the data after correlation, denoting the sample amount near the expected phase and the random noise. A dashed line notes the actual phase, in radians, for the three-beam pattern data set.

Fig. 7

Pattern Factor and Phases Post Filtering for the Three-Beam data set

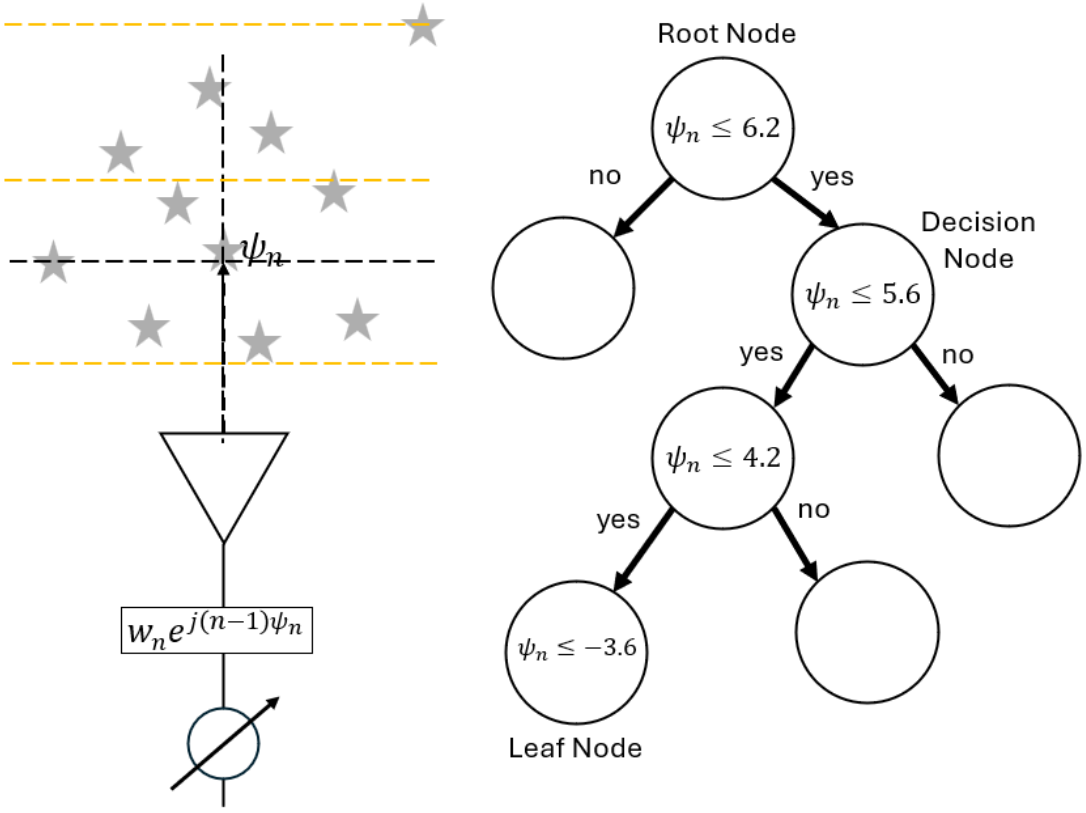


Decision Tree Models

Decision trees can be thought of as flowcharts, with a starting "root node" that leads to branches with potential answers. Branches from the tree, usually dubbed the "decision" nodes, lead the model into more iterations that have more outcomes. When the model finishes, the iterations stop at the "leaf" node and end the sequence. Fig. 8 illustrates a relevant decision tree model to show how the phases for the elements are best chosen based on a classic decision tree. Each decision node recursively decides to find the phase that best fits within its threshold per decision node until optimally finding the leaves that hold the best solution in the purest possible outcome. Is this different from an "if else" statement? Yes, as the model will learn the best features to use and the best corresponding threshold values to optimally split the data.

Fig. 8

Example Decision Tree Sequence



Gradient Boosted Trees

The paper's research team described their machine learning model as a "gradient boosting tree," explicitly using a regression algorithm. In machine learning, gradient-boosted decision trees build simple prediction models sequentially, where each model tries to predict the error left over by the previous model (machinelearningplus.com). Fig. 9 is a high-level illustration of how decision trees use the previous errors to find the best solution. In contrast to a decision tree model, gradient trees may use one root node with one decision node per iteration. This ensemble of "weak" learners predicts the solution by minimizing the error from each

previous solution. Using weak learners lowers the complexity of the model and results in a solution close to the accuracy of a decision tree.

Fig. 9

Gradient boosting trees using the previous error with a loss function to enhance the next tree

The research paper's regression gradient boosted tree method is noted in the paper by

$$p_k = w_k \,/\, \sum_{i=1}^{N} w_{ki}$$

$$e_k = \sum_{i=1}^{N} p_k |h_k(x_i) - y_i|$$

$$b_k = \frac{e_k}{1 - e_k}$$

$$w_{(k+1)i} = w_{ki} b_k^{1-|h_k(x_i - y_i|}$$

where k is the iteration index, e_k is the error at step k, and w_{k+1} represents sample weights for the estimator at the next step. The paper states the procedure focuses on improving performance for miss-predicted samples by increasing their weights with all estimators combined by a weighted majority vote for the final prediction. To explain further on the algorithm, (x_i, y_i) are the data samples (x) and the actual value (y), w_i is the weight with the *i*-th sample, $h_k(x)$ is the weak learner at iteration k, p_k is the normalized weight, b_k is the weighting factor for updating sample weights, and N is the number of samples.

MATLAB uses "fit**r**ensemble()" for its gradient boosting decision trees, with the 'r' designating the function as the regression over classification. The fitrensemble() creates an

aggregation, or ensemble, of learners. MATLAB describes its boosting algorithm, using "LSBoost", as a least-squares boosting regression ensemble that fits a new learner to the difference between the observed response and the aggregated prediction of all learners grown previously (ensemble-algorithms, n.d.). The ensemble minimizes the mean-squared error by using the algorithm

$$y_n - nf(x_n)$$

where y_n is the observed response, $f(x_n)$ is the aggregated prediction from all weak learners grown so far for observation x_n, and n is the learning rate. In contrast to the paper's model, Table III notes how MATLAB uses the following method for its iterative boosting algorithms.

Table III

MATLAB GBT LSBoost Sequence

Step	Algorithm
1. Initialization	$f_0(x) = \frac{1}{N}\sum_{i=1}^{N} y_i$
2. Pseudo-residuals	$r_{im} = y_i - f_{m-1}(x_i)$
3. Fit Weak Learners	$h_m(x) = \arg\min_h \sum_{i=1}^{N} \left(r_{im} - h(x_i)\right)^2$
4. Update the Model	$f_m(x) = f_{m-1}(x) + \eta h_m(x)$

Note that the $y_n - nf(x_n)$ relates to the residual of $y_n - f_{m-1}(x_n)$ when the model makes iterative predictions for each weak learner. With fitrensemble(), one can choose the weight, from 0 to 1, as a percentage of how much each learner will impact the next iteration. The percentage of impact to the overall model is called the "learning rate".

Random Forest

The third model during the research was the TreeBagger() function MATLAB uses for random forest models. Random forest ensembles will use several learners (trees) and take a majority vote on the prediction. The prediction with the most votes becomes the outcome of the random forest. With regression, an average of each tree's prediction becomes the random forest output. With several models making decisions, the outcome will likely have higher accuracy than any model alone.

Random forest models have each tree make its own assumption of the splits (decisions) and the predictions (leaves). The model does this by "bootstrapping" the data, randomly sampling a subset, and creating different training data sets. Then, the model aggregates the results, called "bootstrap aggregating", which is combined into the phrase "bagging." MATLAB's algorithm for the TreeBagger() function with regression is done by

$$\hat{f}(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$$

where B is the bootstrap samples from the training data set, and Tb is the decision tree. The MATLAB model comparison research also used an out-of-Bag (OOB) error estimator. OOB errors provide an unbiased estimate of the prediction error with data not included in each bootstrap sub-sample set. The OOB error calculation is done by

$$OOB\ Error = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - \hat{f}_{OOB}(x_i)\right)^2$$

Data Results

Data results from each model were generally 95%+ accurate at predicting each phase associated with their respective beam pattern. Both the paper's emulated GBT model and the MATLAB random forest model could not correctly predict the 0 phase at $\psi_1$and ψ_{16}, leading to an overall prediction accuracy score of less than 90%. Accuracy is a great metric to measure a model's prediction effectiveness, but cost and complexity are the most significant discriminators between each model. Cross-validation is used to gauge the best learning rate and number of learners for the paper's GBT model and MATLAB's fitrensemble() function. For the random forest TreeBagger() model, the parameters of number of trees and the maximum number of splits (decision nodes) is used. The model uses mean absolute error (MAE) after cross-validation for the best combination of trees, learning rates, and splits to use. The best MAE results from cross-validation are noted in Table IV, with MAE heatmaps noted in Fig. 10 through Fig. 12.

Fig. 10

Cross-correlation plots from the paper's GBT model to find lowest MAE of (a) three-beam pattern, (b) five-beam pattern, (c) saddle-beam pattern, and (d) widebeam pattern

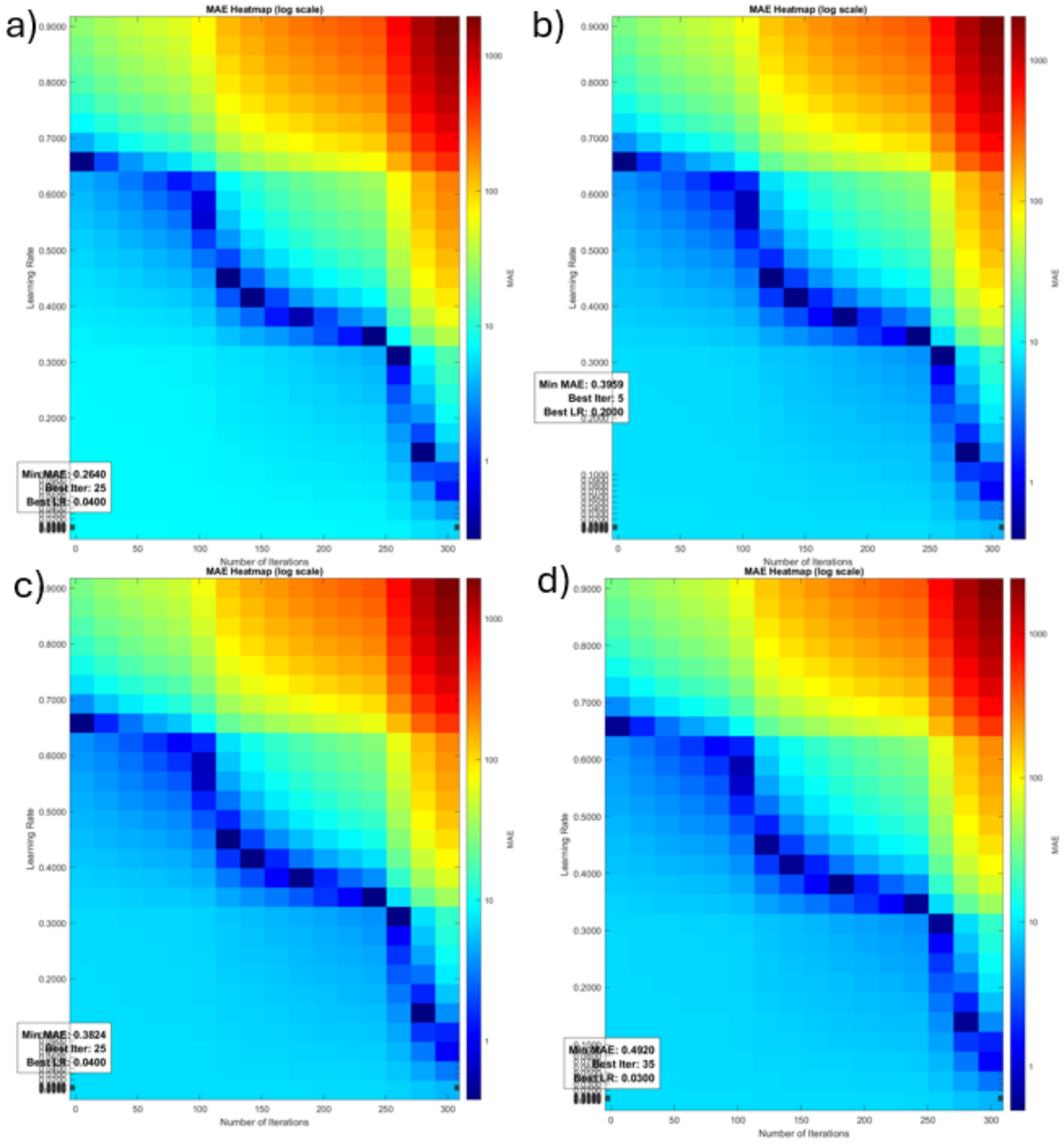


Fig. 11

Cross-correlation plots from the MATLAB's LSBoost GBT model to find lowest MAE of (a) three-beam pattern, (b) five-beam pattern, (c) saddle-beam pattern, and (d) widebeam pattern

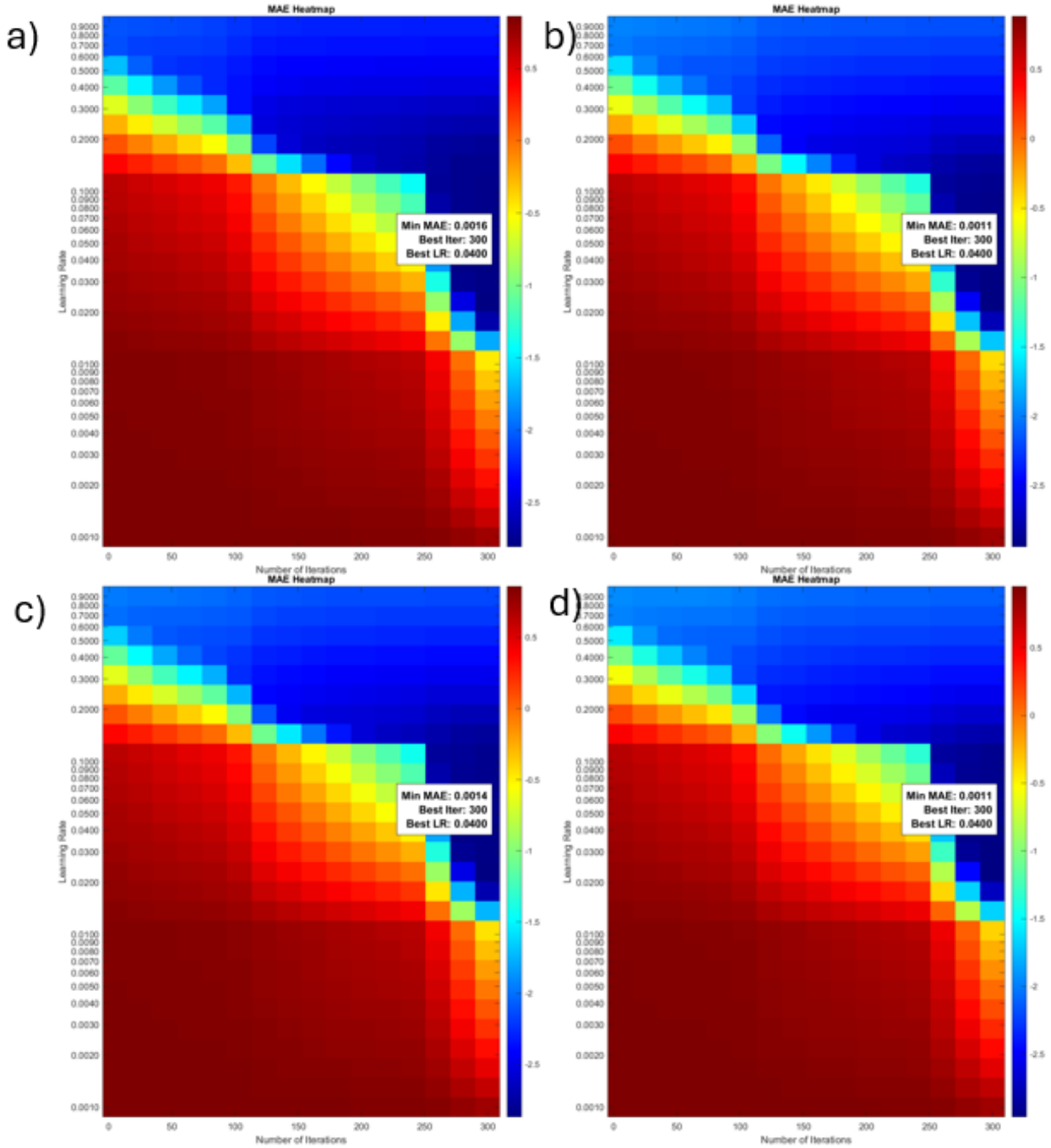


Fig. 12

Cross-correlation plots from the MATLAB's TreeBagger random forest model to find lowest MAE of (a) three-beam pattern, (b) five-beam pattern, (c) saddle-beam pattern, and (d) widebeam pattern

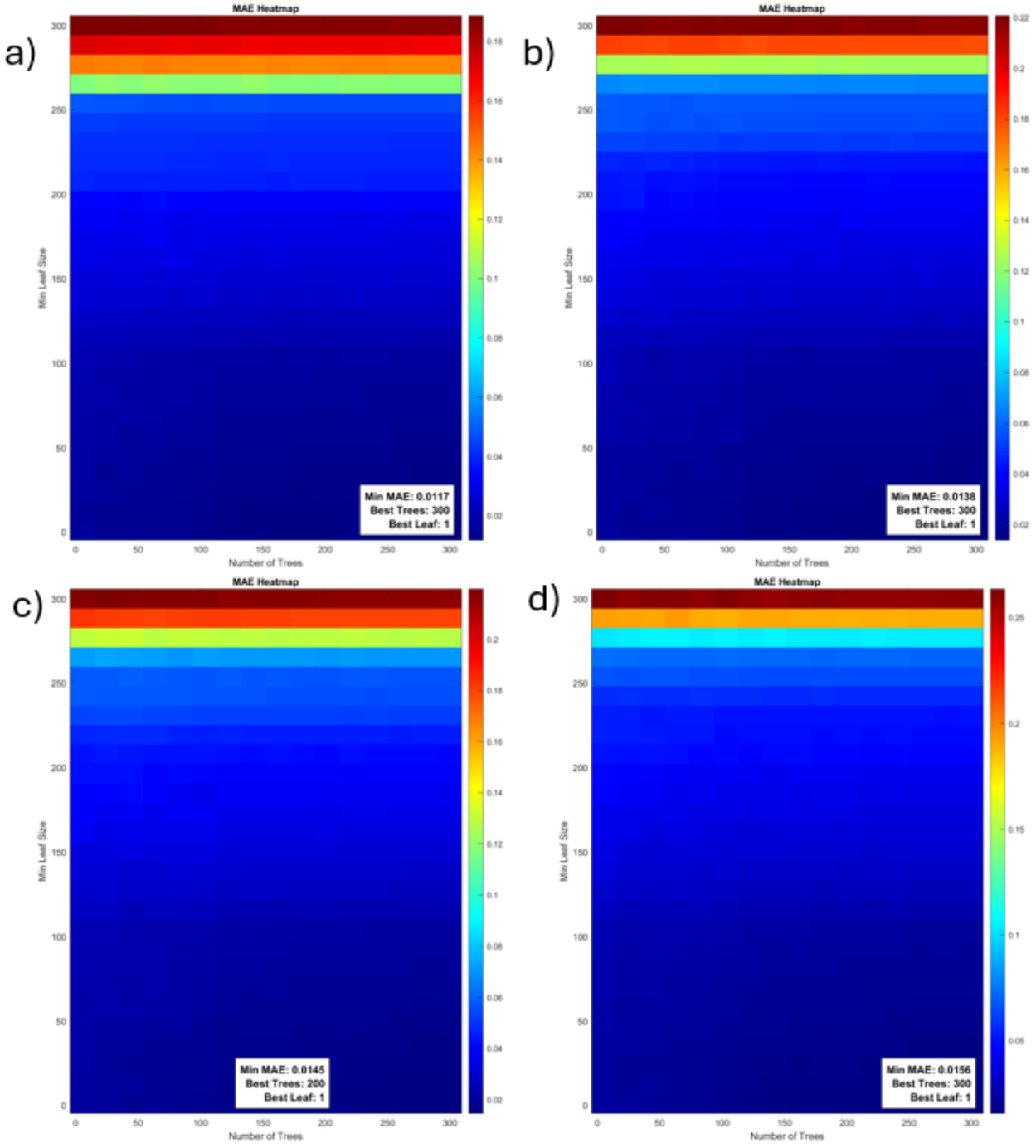


Table IV

MAE of Each Model After Cross-Validation

	Three-Beam	Five-Beam	Saddle-Shaped Beam	Widebeam
	Paper's GBT Model			
Best Number of Iterations (Trees)	25	5	25	35
Best Learning Rate	0.04	0.02	0.04	0.03
Min MAE	0.2640	0.2000	0.3824	0.4920
	MATLAB GBT with LSBoost			
Best Number of Iterations (Trees)	300	300	200	300
Best Learning Rate	0.04	0.04	0.04	0.04
Min MAE	0.0016	0.0011	0.0014	0.0400
	MATLAB TreeBagger() Random Forest			
Best Number of Iterations (Trees)	300	300	200	300
Best Leaf Split Size	1	1	1	1
Min MAE	0.0117	0.0138	0.0145	0.0156

The heatmaps from the cross-correlation note how different the models work from one another using data sets emulating the paper's training and test setup. While using the paper's GBT model, the MAE is higher but can generate a solution in a smaller amount of iterations, noting the small cost and complexity of the model. The MATLAB GBT and random forest model are more accurate, and the number of trees needed to optimize the model is high. Table V also notes model complexity, which shows how long each model is required to cross-validate the 10,000 samples and their respective outcomes. Complexity is an essential factor and leads to considering doing the calculations locally or remotely. Also, the speed at which a new model could be trained may affect its deployment time for new beam pattern requirements. GPU and parallel processing could decrease the time required to train a new model. Still, the difference between the paper's GBT model and MATLAB's shows how optimized a small computational model can be.

Table V

Each model cross-validation time required to find the best optimization

	Paper's GBT Model			
	Three-Beam	Five-Beam	Saddle-Shaped Beam	Widebeam
Cross Validation Time (seconds)	438	455	432	443
	MATLAB GBT with LSBoost			
Cross Validation Time (seconds)	16902	16805	16200	16587
	MATLAB TreeBagger() Random Forest			
Cross Validation Time (seconds)	15342	14737	14536	14112

The time taken for cross-validation of the data notes a vast difference between the paper's GBT model and MATLAB's. Accuracy of every phase with an overall percentage within a 10% margin is indicated in Table VI. The paper's GBT model compares to the random forest model, as both models had difficulty predicting the ψ_1 and ψ_{16} as a 0 phase, lowering the overall accuracy.

Table VI

Overall percentage of phases within 10% margin of each beam pattern prediction

	Paper's GBT Model			
	Three-Beam	Five-Beam	Saddle-Shaped Beam	Widebeam
Percentage of predicted phases within 10% margin of actual phase	85.3	84.4	84.2	83.7
	MATLAB GBT with LSBoost			
Percentage of predicted phases within 10% margin of actual phase	98.2	98	97.94	97.82
	MATLAB TreeBagger() Random Forest			
Percentage of predicted phases within 10% margin of actual phase	85.1	83.1	83.1	82.7

Ultimately, each model effectively predicted each model, showcasing the effectiveness of decision tree models. All models could predict the phases with a millidegree of accuracy using aggregated prediction models with regression or a random forest approach with a majority vote.

Fig 13 through Fig. 15 denote the predicted versus the actual pattern of the models produced after training. The accuracy of Fig. 13, using the paper's GBT method, is noticeable compared to MATLAB's decision tree functions. However, tweaking the author's model should produce better results while retaining far lower complexity.

Fig. 13

Azimuth plots from the paper's GBT model noting the actual (blue) versus predicted (red) patterns for (a) three-beam pattern, (b) five-beam pattern, (c) saddle-beam pattern, and (d) widebeam pattern

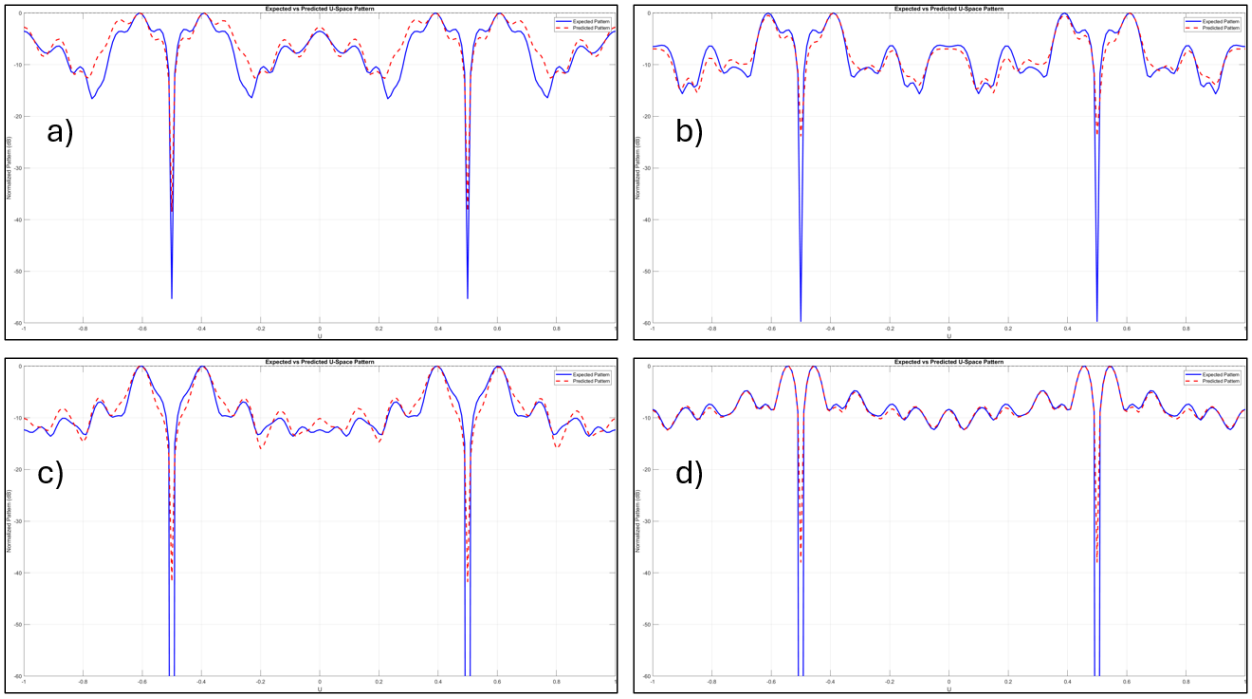


Fig. 14

Azimuth plots from MATLAB'S GBT model with LSBoost noting the actual (blue) versus predicted (red) patterns for (a) three-beam pattern, (b) five-beam pattern, (c) saddle-beam pattern, and (d) widebeam pattern

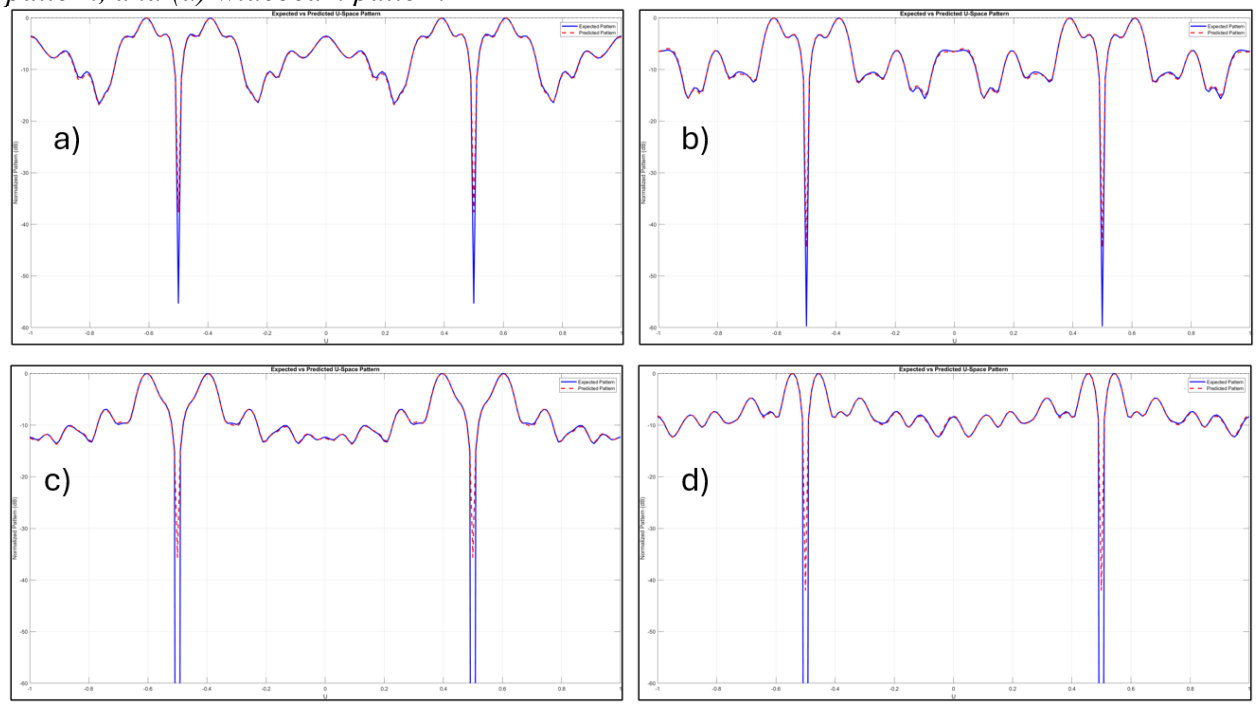
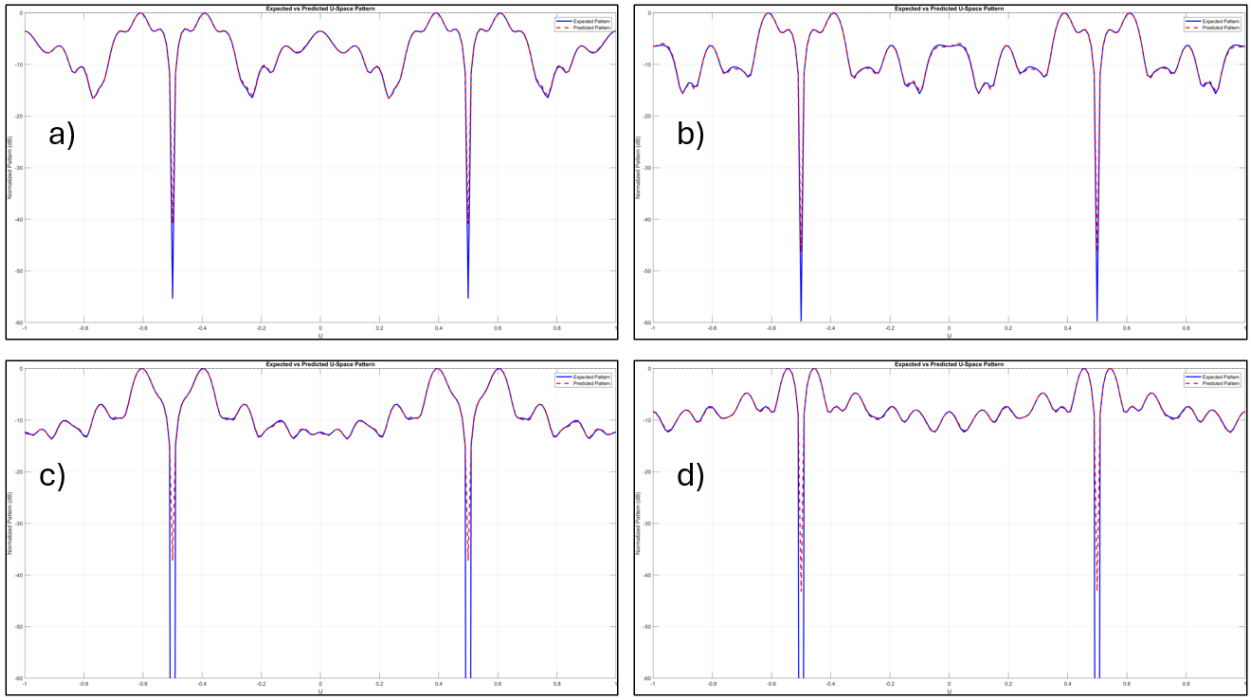


Fig. 15

Azimuth plots from MATLAB's random forest model noting the actual (blue) versus predicted (red) patterns for (a) three-beam pattern, (b) five-beam pattern, (c) saddle-beam pattern, and (d) widebeam pattern



Conclusion

The optimal decision tree method should include a comparison between complexity and accuracy. Within this research review, the authors of "Millimeter Wave Phased Array Antenna Synthesis Using a Machine Learning Technique for Different 5G Applications" demonstrated a low complexity, very accurate GBT method for producing far-field phased array patterns by simply using a pattern factor. Tweaking the GBT model's hyperparameters, using the dataset for training and testing, and verifying the created GBT code to emulate the authors could help improve accuracy. While the authors focused on developing 5G technologies while writing the paper, the advent of 6G technology can further strengthen the importance of more dynamic and

reconfigurable requirements that the telecommunication markets will require. New research on employing smaller phased arrays for vehicle autonomy or self-updating broadband networks could enhance research using GBT models for phased array patterns.

Overall, this research helped me understand GBT models for graduate student studies. I am researching various techniques of RF pattern creation for phased array antennas, such as the phase-only synthesis technique, improving my insight into how the industry is creating newer techniques to tackle markets that need better coverage for ad-hoc events and business. The machine learning approach solves multiple complex engineering problems and utilizes the increasing machine learning technology to benefit industry advancement.

References

5G mmWave frequency bands: 5G mmWave band n257 (28GHz). (2024, July 22). Retrieved from https://www.5gmmwave.com/5g-mmwave-frequency-bands/5g-mmwave-band-n257-28ghz/

An introduction to gradient boosting decision trees. (2024, July 22). Retrieved from https://www.machinelearningplus.com/machine-learning/an-introduction-to-gradient-boosting-decision-trees/

Danesh, S., Amin, G. A., Mahmoud, K. R., & Hamdi, K. N. (2020). Millimeter wave phased array antenna synthesis using a machine learning technique for different 5G applications. In *Proceedings of the IEEE International Symposium on Networks, Computers and Communications (ISNCC)* (pp. 1-6). IEEE. https://doi.org/10.1109/ISNCC49221.2020.9297196

Ensemble algorithms. (2024, July 22). Retrieved from https://www.mathworks.com/help/stats/ensemble-algorithms.html

Guney, K., & Onay, M. (2007). Amplitude-only pattern nulling of linear antenna arrays with the use of bees algorithm. *Progress In Electromagnetics Research*, 68, 247-259. https://doi.org/10.2528/PIER07011204

Leveraging the potential of 5G millimeter wave. (2024, July 22). Retrieved from https://www.ericsson.com/en/reports-and-papers/further-insights/leveraging-the-potential-of-5g-millimeter-wave

Liu, Y., Jiao, Y.-C., Zhang, Y.-M., & Tan, Y.-Y. (2014). Synthesis of phase-only reconfigurable linear arrays using multiobjective invasive weed optimization based on decomposition. *International Journal of Antennas and Propagation*, 2014, 1-9. https://doi.org/10.1155/2014/573405

Mahanti, G. K., Chakraborty, A., & Das, S. (2007). Phase-only and amplitude-phase only synthesis of dual-beam pattern linear antenna arrays using floating-point genetic algorithms. *Progress In Electromagnetics Research*, 68, 247-259.

Rosu, I. (n.d.). *Phased Array Antennas* (pp. 9-22). Retrieved from http://www.qsl.net/va3iul

TreeBagger: Ensemble of bagged decision trees. (2024, July 22). Retrieved from https://www.mathworks.com/help/stats/treebagger.html

Unicast vs. multicast. (2024, July 23). Retrieved from https://www.cbtnuggets.com/blog/technology/networking/unicast-vs-multicast