Machine Learning Based Forward Solver: An Automatic Framework in gprMax

Utsav Akhaury*, Iraklis Giannakis†, Craig Warren‡, and Antonios Giannopoulos§

*Laboratoire d'astrophysique, École Polytechnique Fédérale de Lausanne (EPFL), Switzerland
†School of Geosciences, University of Aberdeen, Aberdeen, United Kingdom
‡Department of Mechanical and Construction Engineering, Northumbria University, Newcastle, United Kingdom
§School of Engineering, The University of Edinburgh, Edinburgh, United Kingdom
f2016684p@alumni.bits-pilani.ac.in, iraklis.giannakis@abdn.ac.uk, craig.warren@northumbria.ac.uk, a.giannopoulos@ed.ac.uk

***Abstract*—General full-wave electromagnetic solvers, such as those utilizing the finite-difference time-domain (FDTD) method, are computationally demanding for simulating practical GPR problems. We explore the performance of a near-real-time, forward modeling approach for GPR that is based on a machine learning (ML) architecture. To ease the process, we have developed a framework that is capable of generating these ML-based forward solvers automatically. The framework uses an innovative training method that combines a predictive dimensionality reduction technique and a large data set of modeled GPR responses from our FDTD simulation software, gprMax. The forward solver is parameterized for a specific GPR application, but the framework can be extended in a straightforward manner to different electromagnetic problems.**

***Index Terms*—Full-Waveform Inversion (FWI), Machine Learning (ML), Principle Component Analysis (PCA), Singular Value Decomposition (SVD), Random Forest, XGBoost (Extreme Gradient Boosting)**

I. Introduction

Recent progress in the field of GPR simulations [1] enabled research on developing advanced modelling tools based on big data and machine learning (ML). In this paper, we explore the concept of a near-real-time, forward modeling approach for GPR that is based on ML architectures. The initial idea for our ML approach was developed in [2] and further enhanced in [3]. Our approach greatly eases the ML-based framework generation by introducing the Random Parameter Generation feature in gprMax [4], our bespoke FDTD simulation software. Using this feature, we generate a large number of GPR models with randomly varying parameters. Subsequently, we reduce the dimensionality of these data similar to [3], and finally, we compare the performance of different ML regressors that take as inputs these random parameters and the compressed A-Scans of the models for training. The ultimate goal is to predict the output responses of GPR models depending on the selected parameters used as inputs by the user. The whole process is automatic, and although the training is time-consuming and computationally demanding, the final output is a near real-time forward solver capable of predicting the resulting A-Scan subject to the given inputs.

A fast numerical solver can greatly accelerate full-waveform inversion (FWI), since the most computationally intensive module of the latter is the emended numerical modelling in each iteration. This would make FWI commercially appealing and attainable using minimum computational resources without the need for high performance computing.

II. Dataset Generation

A. Random Parameter Generation - New Module in gprMax

This new feature facilitates the generation of random parameters for a specific gprMax model and uses the *numpy.random* module from NumPy [5] as backend. It allows the user to specify the Probability Distribution Function (PDF) from which these random parameters are drawn. Subsequently, for every numerical parameter defining the GPR scenario, the user must enter two values (in pairs) that define the specified PDF's parameters. The following convention is followed to activate the random parameter generation mode:

- *#command_name: distr parameter_1.1 parameter_1.2 parameter_2.1 parameter_2.2 parameter_3.1 parameter_3.2 ...*

distr specifies the PDF from which random numbers are drawn. For example, to specify a uniform distribution, the user must enter *u*, and the two subsequent values (*parameter_x.1 & parameter_x.2*) entered would define the lower & upper bounds for the x^{th} parameter. Note that if *parameter_x.1 = parameter_x.2*, then the random number generation is skipped and the constant value is used.

As a quick demonstration, the following command creates a material called *my_sand* which has a relative permittivity ϵ_r drawn from a uniform distribution within the range $[2, 5]$, a conductivity of $\sigma = 0.01 S/m$, $\mu_r = 1$ and $\sigma_* = 0$.

- *#material: u 2 5 0.01 0.01 1 1 0 0 my_sand*

In case the generated random parameter exceeds the model domain bounds, it is automatically constrained to fit inside the domain, which ensures that the FDTD execution is not stopped midway. Finally, all the randomly generated parameters for every GPR model generated are saved to a pickle file. We

automatically compress the pickle file by removing redundant features (i.e. those features that have a constant value for every GPR model generated) in order to improve the training efficiency.

B. Metal Cylinder buried in a Dielectric Medium

Using the Random Parameter Generation feature detailed above, we generated a dataset of 6250 2D GPR models for a metal cylinder buried in a dielectric half-space. We then randomly split the dataset into train-test subsets - 5000 models for training and 1250 for testing. The geometry of the simple 2D scenario is straight-forward and an image from the geometry view is shown in Fig. 1. The transparent region around the boundary of the domain depicts the PML (Perfectly matched layer) region. The line source used for the model's excitation is polarized along the z direction, and is excited using a Ricker waveform with amplitude of one and centre frequency of 1.5 GHz. Table I summarizes the parameters that remain constant for every model.

In each of these models, we vary the following defining parameters, which are drawn from a Uniform Distribution.

- The cylinder radius
- The depth at which the cylinder is placed in the dielectric medium
- The electromagnetic permittivity (ϵ_r) of the dielectric half-space

These are summarized in Table II.

The electromagnetic response (output A-Scans) of all these models are simulated using the gprMax software and saved along with all the randomly generated parameters. These data-pairs would eventually be fed to the ML scheme. Note that since the line source is aligned along the z direction, only the E_z component of the electric field is non-zero. And hence, we only use the E_z component for training.

TABLE I
CONSTANT MODEL PARAMETERS

Parameter	Values
Domain Bounds (x, y, z)	0-240 mm, 0-210 mm, 0-2 mm
Dielectric half-space bounds (x, y, z)	0-240 mm, 0-170 mm, 0-2 mm
Spatial Discretization (dx, dy, dz)	2 mm, 2 mm, 2 mm
Time Window	3 ns
Source Location (x, y, z)	100 mm, 170 mm, 0 mm
Receiver Location (x, y, z)	140 mm, 170 mm, 0 mm
σ - dielectric half-space	0
σ_* - dielectric half-space	0
μ_r - dielectric half-space	1

TABLE II
RANDOMLY VARYING PARAMETERS

Parameter	Lower Bound	Upper Bound
Cylinder Radius (r)	5 mm	20 mm
Cylinder Depth	20 mm	140 mm
ϵ_r - dielectric half-space	4	8

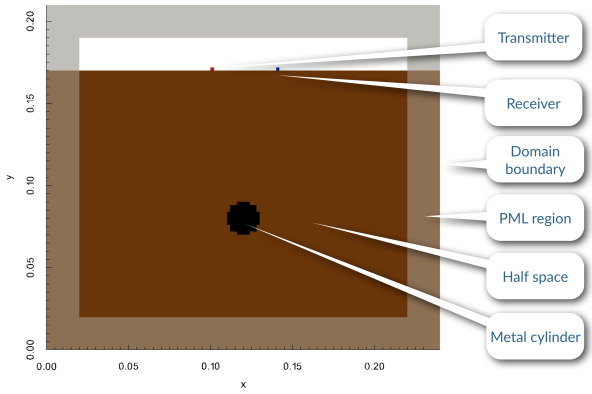

Fig. 1. Geometry of a sample GPR model generated.

III. DIMENSIONALITY REDUCTION

To increase the efficiency of our ML-based forward solver, we pre-process our data in a suitable manner. To that end, we reduce the dimensionality of the A-Scans before feeding them to the ML model. It is observed that the A-Scans can effectively be represented using much lesser number of components without any significant loss in signal quality. We compared the performance of two popular algorithms, namely Principle Component Analysis (PCA) and Singular Value Decomposition (SVD), which give us very similar results in terms of degree of compression.

In either case, we start by using 10 components and computing the Normalized Mean Squared Error (NMSE) between the original and reconstructed signals. We then iteratively increase the number of components used for representing the signal until the difference between successive NMSEs falls below a certain threshold (10^{-12}). We noticed that most of the signal information is captured by the first 10 to 15 components. Obviously, adding more components would further minimize the NMSE. However, even with around 30 components, we get an NMSE $\approx 10^{-12}$ on 1250 test samples, which is sufficiently low for our purpose.

A. Principle Component Analysis (PCA)

Principal component analysis (PCA) is among the oldest methods for reducing the dimensionality of datasets [6], and it does so by increasing interpretability while minimizing information loss. It performs this operation by creating new uncorrelated variables that consecutively maximize variance. These new variables are called the principal components. Finding these principle components reduces to solving an eigenvalue/eigenvector problem, and the new variables are defined by the dataset at hand, not a priori, thus rendering PCA an adaptive data analysis technique. PCA has already been successfully applied to GPR data for clutter reduction in [7] - [8], and more precisely for dimensionality reduction in [9].

Fig. 2 shows a few comparison plots between the A-Scans reconstructed from their compressed representations (by applying inverse PCA transform) and the original A-Scans.

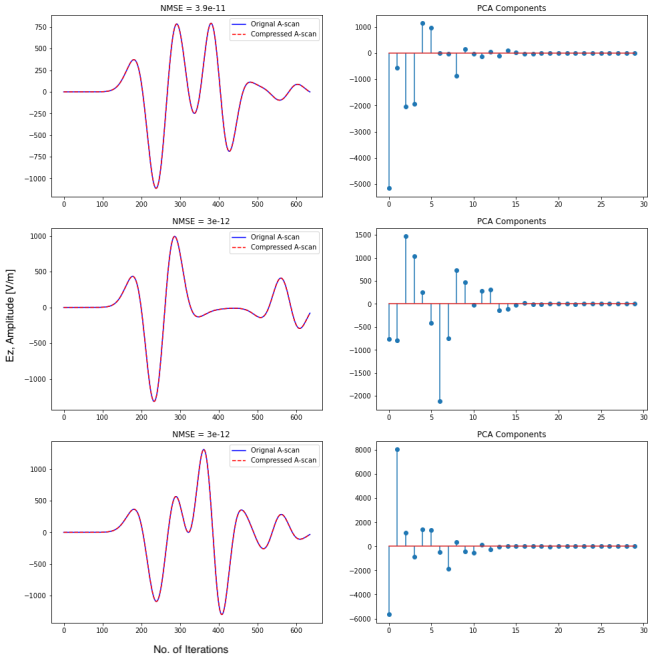

Fig. 2. Dimensionality Reduction using PCA. The dimension is reduced from 637 to 30, and yet the reconstruction is in excellent agreement with the original signal. Notice that most of the signal information is captured by the first 10 components, which are much larger in magnitude compared to the ones that follow.

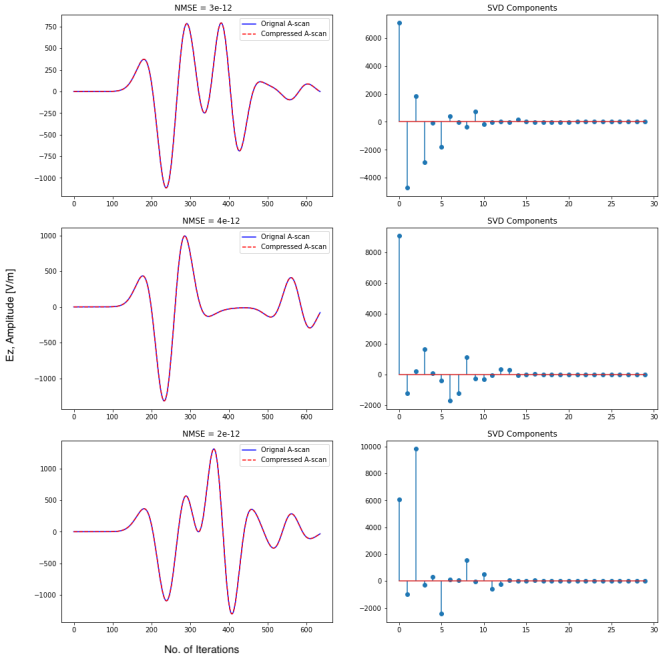

Fig. 3. Dimensionality Reduction using TruncatedSVD. The dimension of the A-Scans is reduced from 637 to 30.

B. Singular Value Decomposition (SVD)

Singular value decomposition (SVD) is a factorization method of a real or complex matrix that extends the concept of the eigen-decomposition of a square normal matrix with an orthonormal eigenbasis to any $m \times n$ matrix [10]. Truncated SVD is a quicker and more compact type of SVD in which the data matrix is truncated and the rest of it is discarded. This can be much quicker and more economical than simple SVD.

For our work, we use scikit-learn's TruncatedSVD transformer. As opposed to PCA, this estimator does not center the data before performing SVD, which implies that it can efficiently handle sparse matrices. Fig. 3 shows a few comparison plots between the A-Scans reconstructed from their compressed representations (by applying inverse SVD transform) and the original A-Scans.

IV. Machine Learning based Forward Solver

Once we have the compressed representations of the output responses of our GPR models, we feed them to our ML-based model for training. Supervised machine learning will unravel the physical causal relationship between the inputs and the resulting trace, and effectively approximate the underlying physical laws of the given problem. To that end, we compare the performance of a few popular ML regressors, such as Random Forest [11] and XGBoost [12], that we extend to perform multi-variate multi-output regression. We had also evaluated the performance of other regressors such as Support Vector Machines (SVM) [13], etc. However, they do not even come close to the accuracy achieved by Random Forest or XGBoost, and hence, we only focus on these two algorithms in our work.

The ML solver outputs the compressed form of the output A-Scan, which can be reconstructed by inverting the same dimensionality reduction algorithm that was used to compress the training data.

A. Random Forest

Random Forest [11] is a supervised machine learning algorithm that is constructed using decision trees [14] and invokes the concept of ensemble learning [15], a technique that combines predictions from multiple ML algorithms to make a more accurate prediction compared to a single model. As an improvement to the decision tree algorithm, it reduces overfitting to datasets and increases precision. It is a bagging-based algorithm in which only a subset of features is selected at random. The maximum tree depth, which is a tunable input parameter, controls the tendancy of the model to overfit to the input data. In case of regression, Random Forest returns the mean or average prediction of the individual trees. It can handle large datasets with ease but does not perform well on very sparse data [16]. The idea of using Random Forest regression for GPR-based predictions has been introduced in [17].

We evaluate the performance of our trained Random Forest model on our test dataset. A few output predictions using Random Forest are shown in Fig. 4.

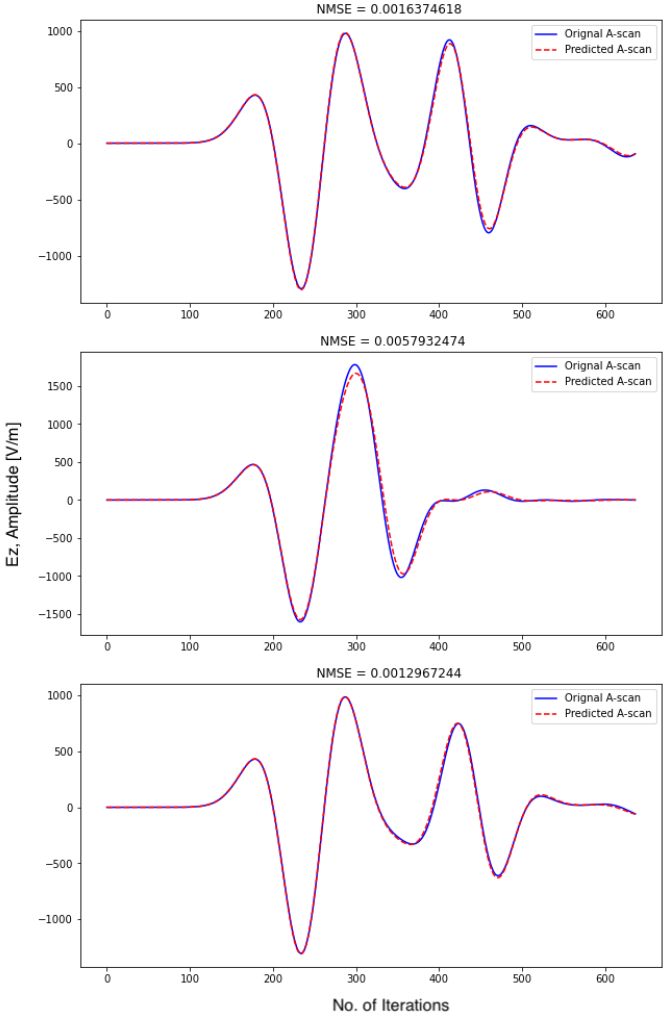

Fig. 4. Output predictions for Random Forest.

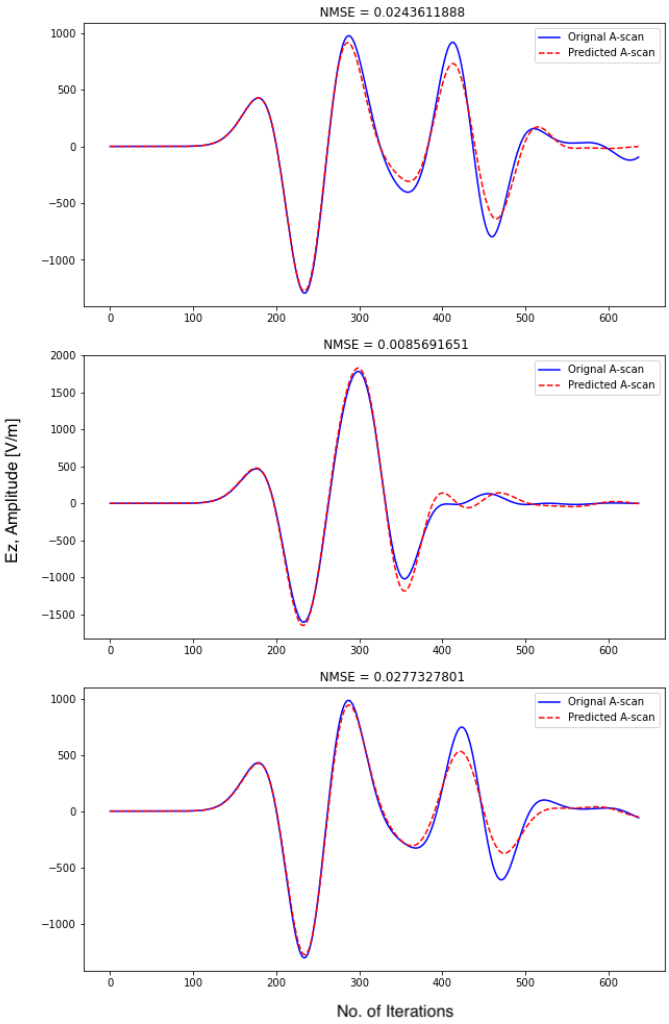

Fig. 5. Output predictions for XGBoost.

B. XGBoost

XGBoost [12], or Extreme Gradient Boosting, is also a decision-tree-based ensemble Machine Learning algorithm that uses a gradient boosting framework. It builds upon the concept of gradient boosting algorithm and provides a more efficient implementation by invoking parallelization, tree pruning, and hardware optimization. For most regression problems, XGBoost provably has the best combination of prediction performance and processing time compared to other algorithms. A few output predictions using XGBoost are shown in Fig. 5.

C. Chain Regression

A Chain Regressor is a multi-label model that arranges regressions into a chain. Every model makes a prediction within the order determined by the chain using all of the available features provided to the model and the predictions of models that precede in the chain. Chain Regressors are suitable where multi-output regression is required and can improve the training performance, as shown in [18]. However, they also take significantly longer to train.

We test the performance of Random Forest and XGBoost arranged in regression chains. However, it is observed that there is no visible improvement when the two cases are applied to our test dataset. On the contrary, it leads to a degradation in performance. A few output predictions using Chain Regression for XGBoost are shown in Fig. 6.

V. Results and Comparison

Table III shows a comparison of the performance of the various ML schemes we have tested. We applied all the methods discussed above to our test dataset of 1250 GPR models. All algorithms were tested on a 2.3 GHz Quad-Core Intel Core i7 processor. For a quantitative comparison, we computed the Normalized Mean Squared Error (NMSE) between the predicted and the simulated ground-truth output responses (that were obtained using gprMax). Note that a lower value of NMSE implies better performance. Alongside, we also noted the time taken by the algorithm for training.

Random Forest achieves the least NMSE on our test dataset, followed by XGBoost. As mentioned earlier, introducing

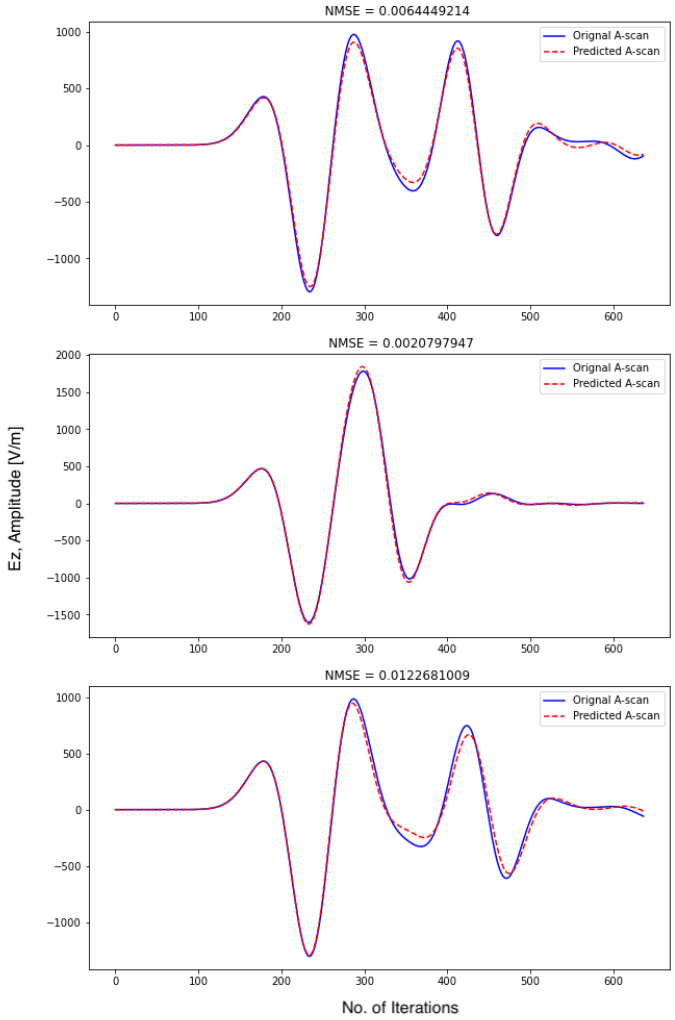

Fig. 6. Output predictions for XGBoost with Chain Regression.

Chain Regression does not show any improvement compared to the original multi-output regressors. Furthermore, it significantly consumes more time during training.

TABLE III
PERFORMANCE COMPARISON

Method	NMSE	Training Time
Random Forest	0.0182	2.8s
Random Forest + Chain Regression	0.0809	179.5s
XGBoost	0.0285	8.8s
XGBoost + Chain Regression	0.1011	17.4s

VI. CONCLUSION

We successfully implemented a framework for generating ML-based forward solvers in gprMax, our FDTD simulation software. This was greatly eased by introducing the Random Parameter Generation feature. We observed that for our test dataset, Random Forest gives us the best performance, taking into consideration both, the NMSE between the ground truth & output predictions, and the training time. Ultimately, gprMax gives the user the flexibility to train & test the performance of any other suitable ML model to better fit a specific dataset. Furthermore, as shown in [3], the performance of neural networks [19] could also be investigated on our dataset in detail, which remains the future scope of this project. The ML solver can also easily handle 3D geometries, in which case, there would only be a greater number of parameters. The underlying principle remains the same.

ACKNOWLEDGMENT

The project was funded via the Google Summer of Code (GSoC) 2021 programme. GSoC initiative is a global program focused on bringing student developers into open source software development. The source code for this project can be found at https://github.com/gprMax/gprMax/pull/294.

REFERENCES

[1] C. Warren et al., "A CUDA-based GPU engine for gprMax: Open source FDTD electromagnetic simulation software," Comput. Phys. Commun., to be published. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0010465518303990.

[2] I. Giannakis, A. Giannopoulos, and C. Warren, "A machine learning approach for simulating ground penetrating radar," in Proc. 17th Int. Conf. Ground Penetrating Radar, Rapperswil-Jona, Switzerland, 2018, pp. 1–4.

[3] I. Giannakis, A. Giannopoulos, and C. Warren, "A Machine Learning Based Fast Forward Solver for Ground Penetrating Radar With Application to Full Waveform Inversion," IEEE Transactions on Geoscience and Remote Sensing. 2019 ; Vol. 57, No. 7. pp. 4417-4426.

[4] C. Warren, A. Giannopoulos, and I. Giannakis, "gprMax: Open source software to simulate electromagnetic wave propagation for ground penetrating radar," Comput. Phys. Commun., vol. 209, pp. 163–170, Dec. 2016.

[5] C.R. Harris, K.J. Millman, S.J. van der Walt, et al., "Array programming with NumPy", Nature 585, 357–362, 2020.

[6] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," IEEE Trans. Pattern Anal. Mach. Intell., vol. 35, no. 8, pp. 1798–1828, Aug. 2013.

[7] G. B. Kaplan, O. Içoglu, A. B. Yoldemir and M. Sezgin, "Real-time object detection using dynamic principal component analysis," in Proc. 13th Int. Conf. Ground Penetrating Radar, Lecce, Italy, Jun. 2010, pp. 1–6.

[8] E. Tebchrany, F. Sagnard, V. Baltazart, J. Tarel and X. Dérobert, "Assessment of statistical-based clutter reduction techniques on ground-coupled GPR data for the detection of buried objects in soils," in Proc. 15th Int. Conf. Ground Penetrating Radar, Brussels, Belgium, Jun./Jul. 2014, pp. 604–609.

[9] P. Kaczmarek and J. Pietrasinski, "Principal component analysis in interpretation of A-Scan measurements in GPR system," in Proc. 15th Int. Radar Symp. (IRS), Gdansk, Poland, Aug. 2014, pp. 1–5.

[10] V. Klema and A. Laub, "The singular value decomposition: Its computation and some applications," in IEEE Transactions on Automatic Control, vol. 25, no. 2, pp. 164-176, April 1980.

[11] L. Breiman, "Random Forests", Machine Learning 45, 5–32, 2001.

[12] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system", Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining, pp. 785-794, 2016.

[13] M. A. Hearst, S. T. Dumais, E. Osuna, J. Platt and B. Scholkopf, "Support vector machines," in IEEE Intelligent Systems and their Applications, vol. 13, no. 4, pp. 18-28, July-Aug. 1998.

[14] A. Navada, A. N. Ansari, S. Patil and B. A. Sonkamble, "Overview of use of decision tree algorithms in machine learning," 2011 IEEE Control and System Graduate Research Colloquium, 2011, pp. 37-42.

[15] F. Huang, G. Xie and R. Xiao, "Research on Ensemble Learning," 2009 International Conference on Artificial Intelligence and Computational Intelligence, 2009, pp. 249-252.

[16] C. Tang, D. Garreau, and U. V. Luxburg, "When do random forests fail?", In Proceedings of the 32nd International Conference on Neural Information Processing Systems (NIPS'18). Curran Associates Inc., Red Hook, NY, USA, 2987–2997.
[17] I. Giannakis, A. Giannopoulos and C. Warren, "A Machine Learning Scheme for Estimating the Diameter of Reinforcing Bars Using Ground Penetrating Radar," in IEEE Geoscience and Remote Sensing Letters, vol. 18, no. 3, pp. 461-465, March 2021.
[18] H. Borchani, G. Varando, C. Bielza, P. Larrañaga, "A survey on multi-output regression", WIREs Data Mining Knowl Discov, 5: 216-233, 2015.
[19] C. M. Bishop and G. Hinton, Neural Networks for Pattern Recognition. London, U.K.: Oxford Univ. Press, 1996.