Liangshuai Guo, Maokun Li, Shenheng Xu, Fan Yang, and Li Liu

# Electromagnetic Modeling Using an FDTD-Equivalent Recurrent Convolution Neural Network

*Accurate computing on a deep learning framework.*

©SHUTTERSTOCK.COM/ROB NAZH

In this study, a recurrent convolutional neural network (RCNN) is designed for full-wave electromagnetic (EM) modeling. This network is equivalent to the finite difference time domain (FDTD) method. The convolutional kernel can describe the finite difference operator, and the recurrent neural network (RNN) provides a framework for the time-marching scheme in FDTD. The network weights are derived from the FDTD formulation, and the training process is not needed. Therefore, this FDTD-RCNN can rigorously solve a given EM modeling problem as an FDTD solver does.

This study links the FDTD method with the artificial neural network (ANN) such that EM modeling may be accurately

*Digital Object Identifier 10.1109/MAP.2021.3127514*
*Date of current version: 24 December 2021*

computed on the deep learning framework. Benefiting from the recent development of software and hardware in machine learning research, the FDTD-RCNN can model EM-wave propagation on massively parallel architectures with a good parallel efficiency and the same level of accuracy as the traditional FDTD.

## INTRODUCTION

Computational EM (CEM) plays an essential role in modern EM engineering, such as wireless communication, antenna design, biomedical imaging, remote sensing, EM compatibility analysis, and so on. Typical numerical techniques in CEM include the method of moments [1], finite element method [2], and FDTD approach [3], among others. The FDTD method partitions the computational domain into pixels (2D) or voxels (3D) [3] using Cartesian grids. The EM field is approximated by discrete samples defined on the grid and updated recursively in time based on the discrete Maxwell equations. FDTD can solve transient EM problems and is good at wideband EM simulation.

FDTD has a computational complexity of $O(N)$, where $N$ is the number of discrete electric and magnetic field values [4]. Therefore, it can be very efficient. Moreover, massive parallelization can further accelerate its computational speed [5], [6]. Many parallelization schemes for FDTD are based on domain decomposition, in which the entire computational domain is partitioned into several subdomains. The discrete field values in every subdomain can be solved on individual processors and then combined to construct a solution in the entire domain.

For example, Guiffaut and Mahdjoubi [7] presented a parallel FDTD algorithm using the message passing interface library. Krakiwsky et al. [8] discussed the acceleration of FDTD using a GPU. In [5], Bao and Chen showed an efficient parallelization scheme based on domain decomposition for a leapfrog alternating-direction implicit-FDTD method. Zhou et al. [6] studied fast and accurate radar cross-section computation using a parallel FDTD technique on a high-performance computing platform. In these studies, customizing FDTD to parallel architectures and fully releasing their computing power requires careful tuning of the computer program. This may include but is not limited to a reduction of communication among individual processors, control of the cache memory at different levels, and so on.

Recently, much progress has been made in machine learning methods, especially deep learning techniques, which has significantly improved the performance of image classification, speech recognition, and many other applications [9]–[12]. The architecture of deep CNNs improves the generalization ability of ANNs [13]. Massive parallelization makes it possible to train a deep network with millions of parameters.

Deep learning techniques have been applied to CEM. They can improve the quality of the reconstruction for inverse scattering problems, such as in [12] and [14]–[17]. They have also been applied to microwave biomedical imaging [18]–[20], nondestructive testing [21], and through-wall imaging [22]. A similar concept can be found in the learning-by-example scheme proposed by Massa et all [23]. With the help of information learned from data through offline computation, deep learning techniques can improve the accuracy and efficiency in EM sensing and imaging.

In addition to their applications in EM inverse problems, deep learning techniques have also been used in modeling. A machine learning-based CEM solver for EM compatibility analysis is proposed by Jiang et al. [11]. An end-to-end deep CNN was designed and trained to solve Poisson's equation in almost real time and with an average numerical accuracy of two digits [24]. Machine learning has also been used to represent the perfectly matched layer (PML) in FDTD to improve the computational efficiency [25]. The application of deep learning to EM engineering is a rapidly expanding field, and the references here are far from complete due to the authors' limited knowledge. Please refer to [26]–[28] for more comprehensive reviews.

In these studies, various deep neural networks are designed based on different application scenarios. Large data sets are constructed to train the network parameters. With a sophisticated training process, the deep neural network can output the desired results given the input parameters. The prior information embedded in the data sets can help to save a large amount of computing time. In this process, the network can be considered as a "black box." It can approximate highly nonlinear functions even if they are not easy to express with explicit mathematics rigorously. This feature helps its success in image and speech processing.

While every coin has two sides, the physics is hidden inside the network and cannot be interpreted clearly. Therefore, in the application of deep learning to EM problems, it is difficult to link the trained network parameters with the EM theory. This disadvantage limits the network's prediction accuracy and also its ability to generalize. In applications that require both of these features, such as numerical EM modeling, the trained network still cannot reach the same level of accuracy as traditional forward solvers. Recently, physics-embedded deep neural networks were proposed by Guo et al. [29], [30] to solve surface and volume integral equations. The network is built based on physical principles. By incorporating physical simulation into deep neural networks, the proposed networks are physically interpretable, resulting in significant improvements in their accuracy and generalization ability.

In this work, we follow another route by analyzing the computational elements of both the deep neural network and the FDTD algorithm. We found that the discrete finite difference operator in FDTD can be described as a convolution operator in a CNN, and the time-marching scheme can be described in the framework of an RNN. Based on this observation, we can set up an RCNN to rigorously perform the FDTD computation. The parameters in the RCNN can be derived directly from the FDTD formulation.

This scheme allows us to implement the FDTD on a machine learning framework [25], [31], such as TensorFlow [32]–[34] or PyTorch [35], that can fully utilize the parallel computing power without lower-level programming on a CPU or GPU. Numerical examples show that the FDTD-RCNN can solve Maxwell's equations with the same level of accuracy as the traditional FDTD plus a good parallel efficiency on massively parallel architectures. We also notice that the link between the discrete version of wave physics and the RNN has recently been discussed by other researchers in [21] and [36]–[38]. Compared with these works, here, we focus more on the computational aspect of an RCNN-FDTD, directly setting the network weights instead of training.

This article is an expansion of [39]. Here, we present more details of this method, including 1) a more detailed description of the link between FDTD and an RCNN, 2) additional information on the implementation of RCNN-FDTD, and 3) more numerical examples to verify the accuracy and efficiency of FDTD-RCNN.

## FORMULATION

In this section, we briefly review the FDTD formulation and derive the RCNN-FDTD scheme.

### 2D FDTD

The equations for a transverse magnetic (TM)-polarized EM wave in an isotropic domain $(E_z, H_x, H_y)$ can be written as [3]

$$\frac{\partial E_z}{\partial y} = -\mu\frac{\partial H_x}{\partial t} - \sigma_m H_x \tag{1}$$

$$\frac{\partial E_z}{\partial x} = \mu\frac{\partial H_y}{\partial t} + \sigma_m H_y \tag{2}$$

$$\frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y} = \varepsilon\frac{\partial E_z}{\partial t} + \sigma E_z, \tag{3}$$

where $\varepsilon$ and $\mu$ are the electric permittivity and magnetic permeability, respectively, and $\sigma$ and $\sigma_m$ are the electric conductivity and magnetic susceptibility, respectively.

These equations can be discretized using the central difference scheme in both the spatial and time domains, as shown in Figures 1 and 2. In the spatial domain, a Yee grid [3] is used in which the computational domain is partitioned into a square subdomain using Cartesian grids. For 2D TM cases, the discrete magnetic field $E_z$ is defined at the center of the grid, and the discrete electric field ($H_x$ and $H_y$) is defined along the edges of each cell. To preserve the accuracy of the discrete representation of the electric and magnetic fields, the grid sizes are usually less than one tenth of the wavelength at the highest simulation frequency. Using $\Delta x$ and $\Delta y$ to represent the grid size along the $x$- and $y$-axes, respectively, we can write the FDTD formulation as the following:

$$\begin{aligned}
H_{x,i,j}^{t+\frac{1}{2}} &= \frac{1-\frac{\Delta t}{2\mu_{i,j}}\sigma_{m,i,j}}{1+\frac{\Delta t}{2\mu_{i,j}}\sigma_{m,i,j}} H_{x,i,j}^{t-\frac{1}{2}} \\
&\quad - \frac{\frac{\Delta t}{\mu_{i,j}}}{\Delta y\left(1+\frac{\Delta t}{2\mu_{i,j}}\sigma_{m,i,j}\right)}\left(E_{z,i,j+1}^{t}-E_{z,i,j}^{t}\right) \\
&= W_1 \cdot H_{x,i,j}^{t-\frac{1}{2}} + W_2 \cdot \left(E_{z,i,j+1}^{t}-E_{z,i,j}^{t}\right) \\
&= f_1\left(H_{x,i,j}^{t-\frac{1}{2}}\right) + g_1\left(E_{z,i,j}^{t}\right),
\end{aligned} \tag{4}$$

$$\begin{aligned}
H_{y,i,j}^{t+\frac{1}{2}} &= \frac{1-\frac{\Delta t}{2\mu_{i,j}}\sigma_{m,i,j}}{1+\frac{\Delta t}{2\mu_{i,j}}\sigma_{m,i,j}} H_{y,i,j}^{t-\frac{1}{2}} \\
&\quad - \frac{\frac{\Delta t}{\mu_{i,j}}}{\Delta x\left(1+\frac{\Delta t}{2\mu_{i,j}}\sigma_{m,i,j}\right)}\left(E_{z,i+1,j}^{t}-E_{z,i,j}^{t}\right) \\
&= W_3 \cdot H_{y,i,j}^{t-\frac{1}{2}} + W_4 \cdot \left(E_{z,i+1,j}^{t}-E_{z,i,j}^{t}\right) \\
&= f_2\left(H_{y,i,j}^{t-\frac{1}{2}}\right) + g_2\left(E_{z,i,j}^{t}\right),
\end{aligned} \tag{5}$$

$$\begin{aligned}
E_{z,i,j}^{t+1} &= \frac{1-\frac{\Delta t}{2\varepsilon_{i,j}}\sigma_{i,j}}{1+\frac{\Delta t}{2\varepsilon_{i,j}}\sigma_{i,j}} E_{z,i,j}^{t} \\
&\quad + \frac{\frac{\Delta t}{\varepsilon_{i,j}}}{\Delta x\left(1+\frac{\Delta t}{2\varepsilon_{i,j}}\sigma_{i,j}\right)}\left(H_{y,i,j}^{t+\frac{1}{2}}-H_{y,i-1,j}^{t+\frac{1}{2}}\right) \\
&\quad - \frac{\frac{\Delta t}{\varepsilon_{i,j}}}{\Delta y\left(1+\frac{\Delta t}{2\varepsilon_{i,j}}\sigma_{i,j}\right)}\left(H_{x,i,j}^{t+\frac{1}{2}}-H_{x,i,j-1}^{t+\frac{1}{2}}\right) \\
&= W_5 \cdot E_{z,i,j}^{t} + W_6 \cdot \left(H_{y,i,j}^{t+\frac{1}{2}}-H_{y,i-1,j}^{t+\frac{1}{2}}\right) \\
&\quad + W_7\left(H_{x,i,j}^{t+\frac{1}{2}}-H_{x,i,j-1}^{t+\frac{1}{2}}\right) \\
&= f_3\left(H_{y,i,j}^{t-\frac{1}{2}}\right) + g_3\left(H_{z,i,j}^{t}\right),
\end{aligned} \tag{6}$$

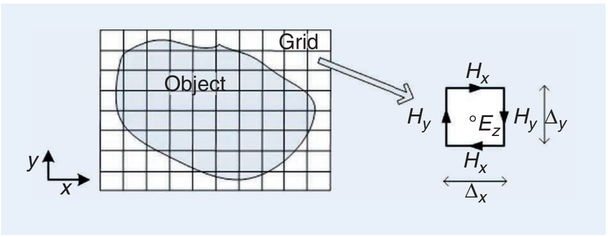

FIGURE 1. The discretization of an object in 2D FDTD.

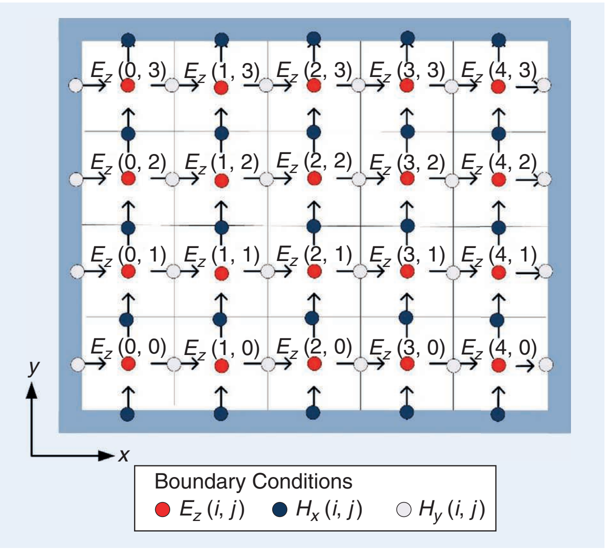

FIGURE 2. The truncation model of 2D FDTD.

where $i$ and $j$ represent the position index in the discrete space, the superscript $t$ is the discrete time index, and $\Delta x$ and $\Delta y$ are the grid sizes in the $x$- and $y$-axes, respectively. Considering all of the discrete field in space, $H_x^{t+\frac{1}{2}}, H_y^{t+\frac{1}{2}}$, and $E_z^t$ can be written into matrix forms corresponding to their locations in the discrete spatial domain. $W_1, W_2, W_3, W_4, W_5, W_6$, and $W_7$ in (4)–(6) represent two coefficient matrices for describing the space grid and time loop of the FDTD process. In this FDTD formulation, $g(x)$ represents the space update function, and $f(x)$ is the time update function.

The FDTD method cannot simulate wave propagation in infinite space unless a proper boundary condition, such as the PML [44]–[46], is used to truncate the infinite domain into a finite one while still maintaining the original wave propagation. The idea of a PML can be expressed in (7)–(10). In these equations, four auxiliary variables $B_x, B_y, P_z$, and $D_z$ are used to ensure the inner field attenuation and no reflection from the boundary between free space and the PML. Here, $\sigma_x$ and $\sigma_y$ represent the conductivity in the PML on the $x$- and $y$-axes, respectively; $\kappa_x$ and $\kappa_y$ are the relative permittivity in the PML on the $x$- and $y$-axes, respectively; $\varepsilon_1$ and $\mu_1$ are the permittivity and permeability inside the PML; $B_x$ and $B_y$ are the magnetic flux density along the $x$- and $y$-axes inside the PML, respectively; $D_z$ is the electric flux density on the $z$-axis inside the PML; and $P_z$ is an intermediate variable:

$$\left.\begin{aligned}\frac{\partial E_z}{\partial y} &= -\kappa_y \frac{\partial B_x}{\partial t} - \frac{\sigma_y}{\varepsilon_0} B_x \\ \frac{\partial E_z}{\partial x} &= \kappa_x \frac{\partial B_y}{\partial t} + \frac{\sigma_x}{\varepsilon_0} B_y\end{aligned}\right\} \text{update } E_z \to B_x, B_y \tag{7}$$

$$\left.\begin{aligned}\kappa_x \frac{\partial B_x}{\partial t} + \frac{\sigma_x}{\varepsilon_0} B_x &= \mu_1 \frac{\partial H_x}{\partial t} \\ \kappa_y \frac{\partial B_y}{\partial t} + \frac{\sigma_y}{\varepsilon_0} B_y &= \mu_1 \frac{\partial H_y}{\partial t}\end{aligned}\right\} \text{update } B_x, B_y \to H_x, H_y \tag{8}$$

$$\left.\begin{aligned}\frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y} &= \varepsilon_1 \frac{\partial P_z}{\partial t} + \sigma_1 P_z \\ \frac{\partial P_z}{\partial t} &= \kappa_x \frac{\partial D_z}{\partial t} + \frac{\sigma_x}{\varepsilon_0} D_z\end{aligned}\right\} \text{update } H_x, H_y \to P_z \to D_z \tag{9}$$

$$\frac{\partial D_z}{\partial t} = \kappa_y \frac{\partial E_z}{\partial t} + \frac{\sigma_y}{\varepsilon_0} E_z \ \text{ update } D_z \to E_z. \tag{10}$$

In FDTD, the time-marching scheme is adopted; therefore, numerical stability is important. The interval between adjacent time steps needs to obey the dispersion formula that sets the maximum time interval based on the spatial discretization:

$$\Delta t \le \frac{1}{c\sqrt{\left(\frac{1}{\Delta x}\right)^2 + \left(\frac{1}{\Delta y}\right)^2}} \tag{11}$$

$$T_{\text{total}} \ge \frac{LD}{c \cdot \Delta t} = LD \cdot \sqrt{\left(\frac{1}{\Delta x}\right)^2 + \left(\frac{1}{\Delta y}\right)^2}, \tag{12}$$

where $c$ is light speed, and $LD$ is the maximum side length of the domain of computation.

## RCNN MODEL

An RCNN is composed of a CNN and an RNN. A CNN is a type of ANN and has been successfully applied to image processing, natural language processing and other cognitive tasks [39]. In general, a CNN is composed of convolution, pooling, and fully connected (FC) layers, as shown in Figure 3. A CNN must contain convolutional layers and may not contain the pooling or full connection layers. The functions of these layers are briefly explained as follows.

### CONVOLUTIONAL LAYER

The convolutional layer serves as the kernel of the CNN. It convolutes with the local receptive field and can extract the feature map of the input images with multiple filters. The parameters of these convolutional operators can be "learned" from the training data set. Figure 4 shows an intuitive interpretation of the convolutional operator. For a given input $E_{M,M}$, the convolutional operator extracts a continuous block (orange frame) with the same size as the filter $K_{f,f}$ and summarizes all of the multiplications between the block and filter elements at the corresponding position to create an output $H$. The total process can be written as the following:

$$\begin{aligned} H(i,j) &= [E \otimes K]_{i,j} \\ &= \sum_{x=1}^{f} \sum_{y=1}^{f} [E(is_0 + x, js_0 + y) \times K(x,y)], \\ i,j &= 1,2,\ldots L, \qquad L = \left\lfloor \frac{M + 2p - f}{s_0} \right\rfloor + 1, \end{aligned} \tag{13}$$

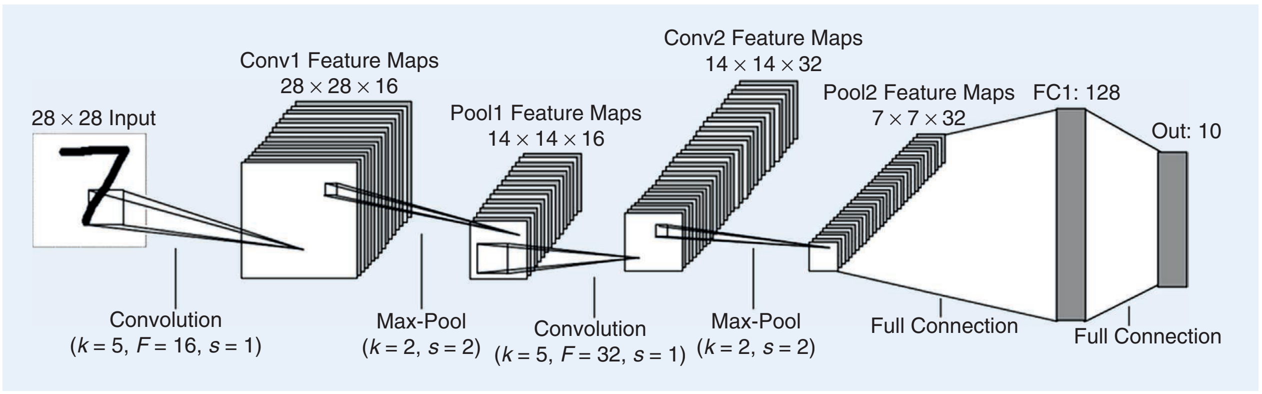

**FIGURE 3.** The CNN architecture. (Source: [47].)

where $s_0$ is the moving step, $M$ is the size of the input, $f$ is the size of the filter, $L$ is the size of the output, $p$ is a padding parameter [10] to set the output size, and $\lfloor \ \rfloor$ means rounding down. The convolutional operator scans along the rows and columns of the input image and computes the output one.

## POOLING LAYER

The pooling layer delivers a down-sampling process. It is used to reduce the number of parameters that describe the input data and enhance the perception. It can also reduce the memory cost and prevent overfitting [10]. With the pooling layer, it becomes feasible to build deeper networks. For a given input $Z_{L\times L}$ and a pooling operator with a size $r \times r$, an $r \times r$ block from the input was extracted and processed by a function to create an output element $M$ that can be represented as the following:

$$M(i,j) = \text{Pooling}(Z) = f(Z_{i,j,l\times l})$$
$$i,j = 1,2,\ldots,n, \ \ n = \left\lfloor \frac{L+2p-r}{s_0} \right\rfloor + 1, \qquad (14)$$

where $n$ is the size of the output, $s_0$ is the step size, and $f$ is a function (often max or mean operations) for the blocks extracted from the input.

## FC LAYER

The FC layer is connected to every neuron; it can map the distributed feature representation to the sample marker space, as shown in Figure 5. The kernel of the FC layer is a multiplication between the input $M$ and weights $w$ as shown in (15); it can also be represented by a matrix-vector multiplication:

$$\text{Out} = Mw. \qquad (15)$$

The features of a CNN include locality, sharing, and pooling. *Locality* means a neuron is only connected to neighboring neurons in the preceding layer. *Sharing* indicates all of the neurons in one layer share the same weights in the convolutional filter. *Pooling* is an operation of activation that outputs a single value from an array in a spatial window. All of these features make CNNs widely used in practice.

RNNs are a class of ANNs in which connections between nodes form a directed graph along a temporal sequence. This allows an RNN to model temporal dynamic behaviors. In an RNN, internal states (memory) are designed to process input sequences. The layers in an RNN not only output to the next layer, but they also output a hidden state for the current layer to process the next sample. The kernel of an RNN is recurrently used with shared weights and other parameters, as shown in Figure 6. Given a time series as the input, the RNN computes the output from the input of not only the current time but also the previous state. It can be written as

$$h_t = V(W_{xh}x_t + W_{hh}h_{t-1} + b), \qquad (16)$$

where $t$ is the time index; $h_t$ is the state element; $x_t$ is the input; and $V$ represents the activation function, which is usually nonlinear, such as rectified linear unit [49] or tanh [50]. $W_xh$ is the weight for connecting the input, $W_hh$ is the weight for connecting the previous state, and $b$ is a bias. $W_hh$ can record the contribution from the previous state; it can count as a serial operator. The RNN can predict the "future phenomenon" based on its past performance, which makes it a dynamic network suitable for time-dependent problems. Combining a CNN and an RNN can keep both their advantages and solve complex problems related to time series.

## *RCNN-FDTD*

If we compare the formulations of an RCNN and FDTD, we can observe their similarities. The functions $g_1$, $g_2$, and $g_3$ in (4)–(6) can be expressed as a convolutional layer because the fields at every pixel share the same local operation. The temporal update of the electrical field $H_x$, $H_y$, and $E_z$ requires the field at the previous time step. This can be modeled as an RNN. Based on these observations, we can formulate FDTD in the framework of an RCNN.

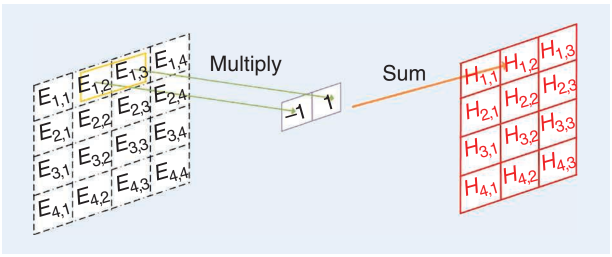

**FIGURE 4.** The convolutional operator.

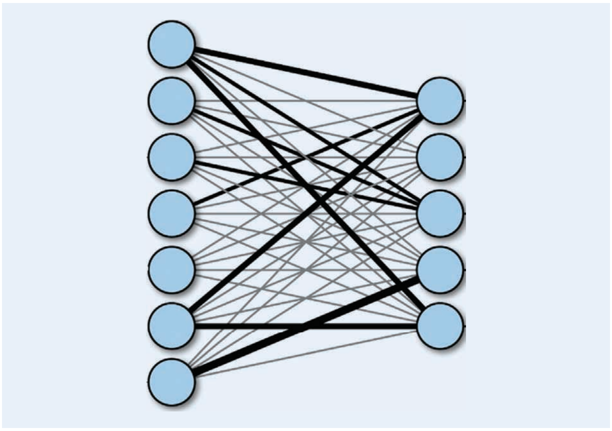

**FIGURE 5.** An FC layer.

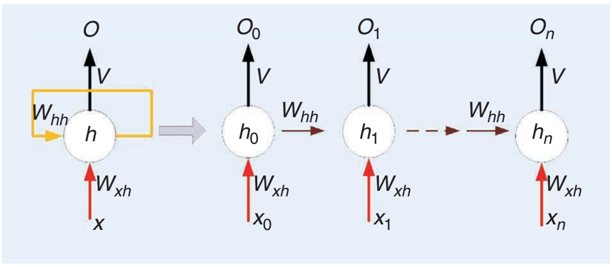

**FIGURE 6.** The RNN architecture. (Source: [48].)

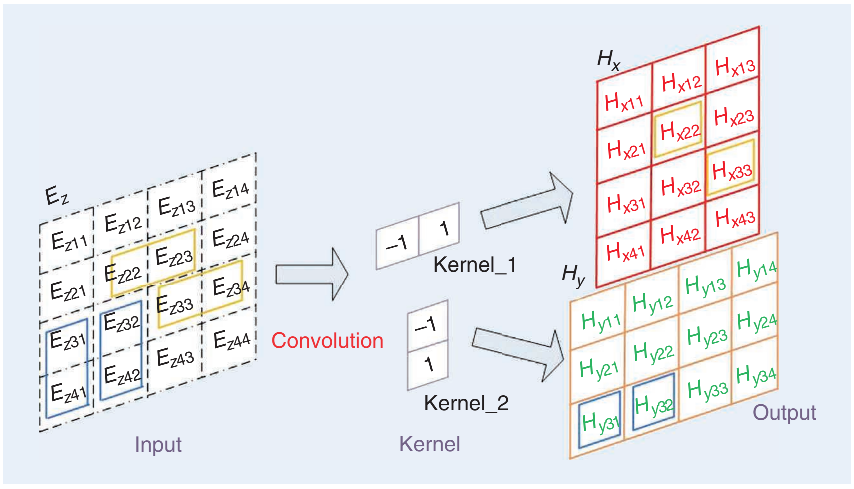
$E_z$
$E_{z11}$ $E_{z12}$ $E_{z13}$ $E_{z14}$
$E_{z21}$ $E_{z22}$ $E_{z23}$ $E_{z24}$
$E_{z31}$ $E_{z32}$ $E_{z33}$ $E_{z34}$
$E_{z41}$ $E_{z42}$ $E_{z43}$ $E_{z44}$
Input
Convolution
–1 1
Kernel_1
–1
1
Kernel_2
Kernel
$H_x$
$H_{x11}$ $H_{x12}$ $H_{x13}$
$H_{x21}$ $H_{x22}$ $H_{x23}$
$H_{x31}$ $H_{x32}$ $H_{x33}$
$H_{x41}$ $H_{x42}$ $H_{x43}$
$H_y$
$H_{y11}$ $H_{y12}$ $H_{y13}$ $H_{y14}$
$H_{y21}$ $H_{y22}$ $H_{y23}$ $H_{y24}$
$H_{y31}$ $H_{y32}$ $H_{y33}$ $H_{y34}$
Output

**FIGURE 7.** The FDTD-based framework of an ANN.

In the modeling of EM-wave propagation in a homogeneous medium, $W$ in FDTD is invariant on the grid; hence, the update process of FDTD can be directly written into an RCNN process. For an inhomogeneous medium, the material parameters can be modeled using an additional weight multiplied to the output of the convolutional layer. This way, we can keep the convolutional kernel consistent across the domain of the simulation. A simple choice for the weight in the convolutional kernel can be $[-1, 1]$ or its transpose to model the finite difference operator, as shown in Figure 7. The update of the electric and magnetic fields in traditional FDTD is translated into a convolution, which can be implemented as a convolution kernel. The computation can be paralleled with high efficiency by taking advantage of the machine learning framework.

In the RCNN for 2D FDTD, a two-layer tensor for the $H$-field and one-layer tensor for the $E_z$-field is constructed. The structure of the RCNN is shown in Figure 6. In this network, one CNN is used to compute $H_x$ and $H_y$ from $E_z$, and the other CNN is used to compute $H_x$, $H_y$ from $E_z$. The time-marching process of FDTD can be directly mapped into the RNN. Moreover, Table 1 describes the update sequence of FDTD with a PML included. Additional variables $B_x$, $B_y$, and $D_z$ are included in the network and updated sequentially. In this study, we implement the uniaxial anisotropic perfectly matched layer update equations [43] through the entire grid (Figure 8). This process is also depicted in Figure 9 ($B$ and $D$ are the magnetic and electric flux density, respectively), in which the 2D TM scattering problem is modeled using an RCNN.

This proposed scheme builds an RNN that is equivalent to the FDTD formulation. Therefore, it performs the same computation as FDTD, and its stability condition is also the same, as shown in (11) and (12).

**TABLE 1. THE FDTD UPDATE CORRESPONDING TO AN RCNN.**

| Parameter | FDTD in a Single Update | RCNN |
|---|---|---|
| Step 1 | Initial $E_z$ | Static diagram |
| Step 2 | $B_x, B_y$ | CNN |
| Step 3 | $H_x, H_y$ | Process |
| Step 4 | $D_z$ | CNN |
| Step 5 | $E_z$ | Process |
| Step 6 | Loop step in time | RNN |

## NUMERICAL EXAMPLES

In this section, several numerical examples are carried out to benchmark the proposed method. All of the computation is executed on a PC (a six-Intel core i5-9600K CPU at 3.7 GHz)

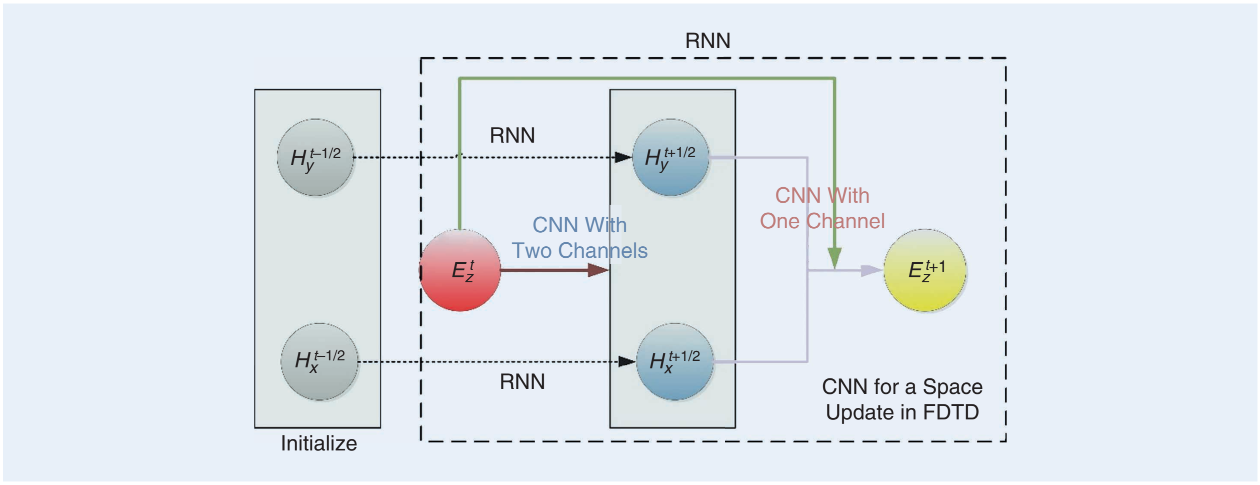

**FIGURE 8.** A simulation framework of 2D FDTD.

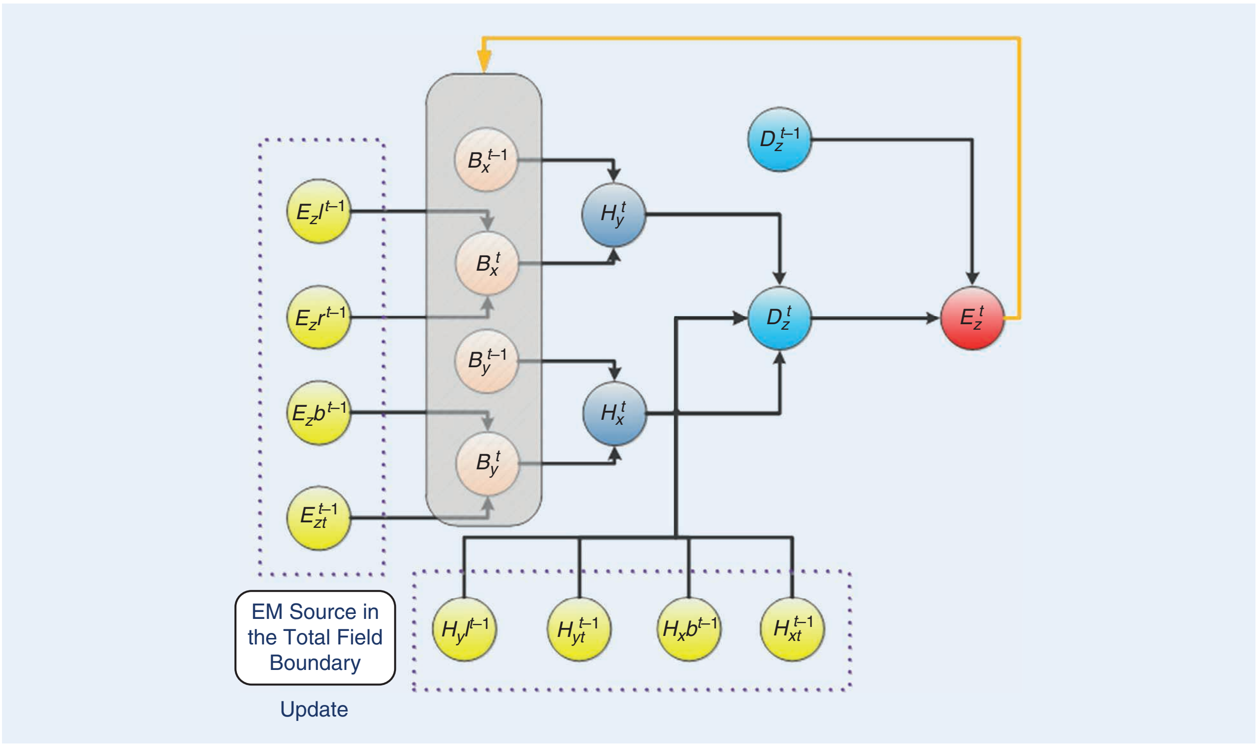

**FIGURE 9.** The RCNN-FDTD with TM mode.

and a computing station with one NVIDIA P100 GPU. The FDTD algorithm is implemented on a CPU and made parallel using OpenMP. The RCNN-FDTD is implemented in the TensorFlow framework [51] and executed on the CPU and GPU to achieve high performance through massive parallelization.

## 1D FDTD MODELING

The formulation of 1D FDTD can degrade from 2D FDTD and be used for validating the algorithm efficiency as well as the effectiveness of the PML [40]–[43]. The definition of the staggered grid is shown in Figure 10. The formulation of 1D FDTD in free space can be written as the following:

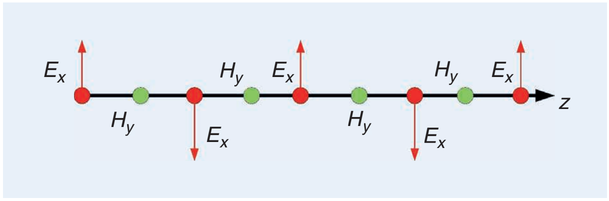

**FIGURE 10.** A simulation of 1D FDTD.

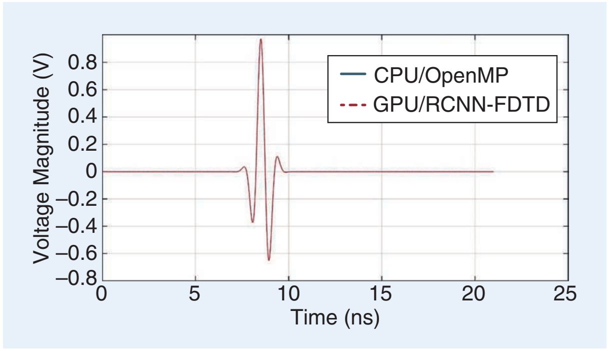

**FIGURE 11.** A comparison of the computed 1D electric field ($E_z$ at the center of the domain) using FDTD on a CPU and RCNN-FDTD on a GPU.

**TABLE 2. A COMPARISON OF THE COMPUTING TIMES OF 1D FDTD.**

| Grid Size | Parallel FDTD on a CPU (s) | RCNN-FDTD on a CPU (s) | RCNN-FDTD on a GPU (s) |
|---|---|---|---|
| 1,000 | $7 \times 10^{-2}$ | $1.6 \times 10^{0}$ | $9.7 \times 10^{0}$ |
| 5,000 | $5 \times 10^{-1}$ | $1.3 \times 10^{1}$ | $4 \times 10^{1}$ |
| 10,000 | $1.4 \times 10^{0}$ | $3.8 \times 10^{1}$ | $7.7 \times 10^{1}$ |
| 50,000 | $4.9 \times 10^{1}$ | $3.5 \times 10^{2}$ | $2.6 \times 10^{2}$ |
| 100,000 | $3.9 \times 10^{2}$ | $6.1 \times 10^{2}$ | $5.3 \times 10^{2}$ |
| 200,000 | $2.6 \times 10^{3}$ | $3 \times 10^{3}$ | $1.1 \times 10^{3}$ |
| 500,000 | $1.7 \times 10^{4}$ | $1.6 \times 10^{4}$ | $3.7 \times 10^{3}$ |
| 700,000 | $3.4 \times 10^{4}$ | $3.1 \times 10^{4}$ | $6.7 \times 10^{3}$ |
| 1,000,000 | $7.4 \times 10^{4}$ | $6.6 \times 10^{4}$ | $1.1 \times 10^{4}$ |
| 1,200,000 | $1.1 \times 10^{5}$ | $9.3 \times 10^{4}$ | $1.4 \times 10^{4}$ |
| 1,500,000 | $1.7 \times 10^{5}$ | $1.4 \times 10^{5}$ | $1.9 \times 10^{4}$ |
| 1,700,000 | $2.2 \times 10^{5}$ | $1.8 \times 10^{5}$ | $2.3 \times 10^{4}$ |
| 2,000,000 | $3.1 \times 10^{5}$ | $2.4 \times 10^{5}$ | $3.4 \times 10^{4}$ |

$$\begin{aligned}&\frac{\partial E_x}{\partial z}=-\frac{\mu}{c_0}\frac{\partial H_y}{\partial t}\rightarrow\\&\qquad\frac{E^t_{x,i,j+1}-E^t_{x,i,j}}{\Delta z}=-\frac{\mu}{c_0}\frac{H^{t+\frac{1}{2}}_{y,i,j}-H^{t-\frac{1}{2}}_{y,i,j}}{\Delta t}\\&\qquad-\frac{\partial H_y}{\partial z}=\frac{\varepsilon}{c_0}\frac{\partial E_x}{\partial t}\rightarrow\\&\qquad-\frac{H^{t+\frac{1}{2}}_{y,i,j+1}-H^{t+\frac{1}{2}}_{y,i,j}}{\Delta z}=\frac{\varepsilon}{c_0}\frac{E^{t+1}_{x,i,j}-E^t_{x,i,j}}{\Delta t}.\end{aligned}\tag{17}$$

In the 1D FDTD simulation, an EM-wave propagation is simulated in free space, as shown in Figure 10. The grid gap $\Delta z$ is 0.003 m, the grid size of $E_x$ is 1,400 along the $z$-axis, and the number of time steps (0.005 ns) is 4,200. The excitation signal is assigned on the left boundary of the simulation domain, expressed as $y=-\cos(2\pi f_0 t)\cdot\exp(-4\pi(t-t_0)^2/\tau^2)$. The center frequency $f_0$ is 1 GHz; $t$ is the time factor; $\tau$ is twice reciprocal of $f_0$; and $t_0$ is the delay time, designed as $0.8\tau$ here. Figure 11 plots $E_z(t)$ at the center of the simulation domain, computed using parallel FDTD on a CPU and RCNN-FDTD on a GPU. We can observe that the results agree with each other well.

Table 2 compares the computing times of the traditional FDTD and RCNN-FDTD. (Marching time steps are fixed at three times at the grid size.) As the number of grids increases, RCNN-FDTD achieves a better performance; this benefit is mainly attributable to the machine learning platform, i.e., TensorFlow in this work. In this comparison, the computer program of RCNN-FDTD is the same as the GPU version. It verifies that RCNN-FDTD has the flexibility of executing on multiple platforms. With the help of a machine learning framework like TensorFlow, an RCNN can fully release the power of the GPU.

Another advantage of RCNN-FDTD is in the adaptation to various computing platforms and ease of implementation. It is much easier to program in TensorFlow or PyTorch than a lower-level programming platform, such as CUDA [52], [53], in which we would need to directly manage memory at different cache levels. With the help of these machine learning frameworks, RCNN-FDTD can also be applied to other computing platforms without much modification. This gives RCNN-FDTD the ability of adaptation to emerging computing hardware.

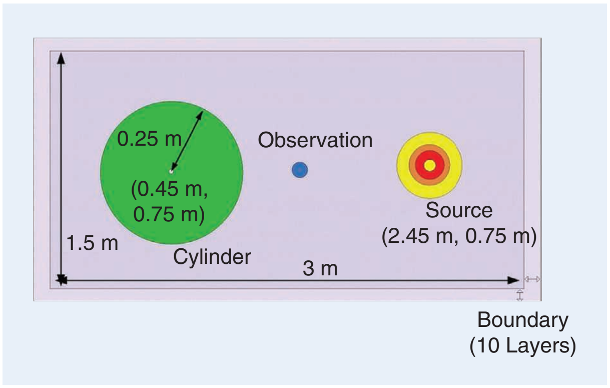

**FIGURE 12.** The model setup for the 2D FDTD simulation.

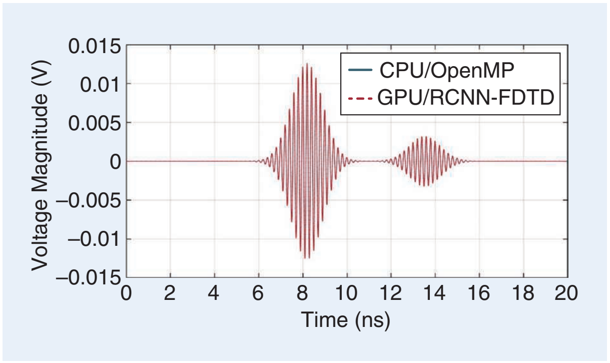

**FIGURE 13.** A comparison of the $E_z$ field at the observation point.

## 2D FDTD MODELING

In this example, point-source radiation near a perfect electric conductor cylinder is considered as shown in Figure 12. The domain of the simulation is $3\text{ m}\times 1.5\text{ m}$ along the $x$- and $y$-axes. It is discretized into $1{,}000\times 500$ pixels along the $x$- and $y$-axes, respectively. The source is located at (2.45 m,0.75 m) in the 2D plane. A circular cylinder is located at (0.45 m,0.75 m) with a radius of 0.25 m. The excitation at the source is a modulated Gaussian pulse with its waveform $y=\sin(2\pi f_0(t-t_0))\cdot\exp(-(t-t_0)^2/\tau^2)$ along the $z$-axis. The center frequency $f_0$ is 5 GHz; $\tau$ is the pulsewidth, set as 1 ns; $t_0$ is the delay time, set up to 5 ns; and the number of time-marching steps (0.005 ns) is 4,000. A 10-layer PML is used for truncation around the domain [3]. An observer is set at the center of the domain of the simulation.

The $E_z$-field is used for verifying the accuracy. An OpenMP paralleled FDTD is implemented on a CPU with six cores [54]. RCNN-FDTD is implemented on a GPU. The field at the observation point is shown in Figure 13. The first pulse is from the radiation of the point source, and the second one is from the scattering of the cylinder, which is similar to the first. The field computed by RCNN-FDTD agrees well with the one

**TABLE 3. A COMPARISON OF THE COMPUTING TIMES OF 2D FDTD.**

| Grid Size | Parallel FDTD on a CPU (s) | RCNN-FDTD on a CPU (s) | RCNN-FDTD on a GPU (s) |
|---|---|---|---|
| $1{,}000\times 500$ | $4.3\times 10^1$ | $9.6\times 10^1$ | $4.6\times 10^1$ |
| $2{,}000\times 1{,}000$ | $3.5\times 10^2$ | $7.2\times 10^2$ | $1.7\times 10^2$ |
| $4{,}000\times 2{,}000$ | $2.9\times 10^3$ | $5.7\times 10^3$ | $8.1\times 10^2$ |
| $6{,}000\times 3{,}000$ | $1\times 10^4$ | $1.9\times 10^4$ | $2.4\times 10^3$ |
| $8{,}000\times 4{,}000$ | $2.4\times 10^4$ | $4.6\times 10^4$ | $5.4\times 10^3$ |
| $10{,}000\times 5{,}000$ | $4.7\times 10^4$ | $8.9\times 10^4$ | $1.1\times 10^4$ |
| $12{,}000\times 6{,}000$ | $8\times 10^4$ | $1.5\times 10^5$ | $1.9\times 10^4$ |
| $14{,}000\times 7{,}000$ | $1.3\times 10^5$ | $2.5\times 10^5$ | $2.3\times 10^4$ |

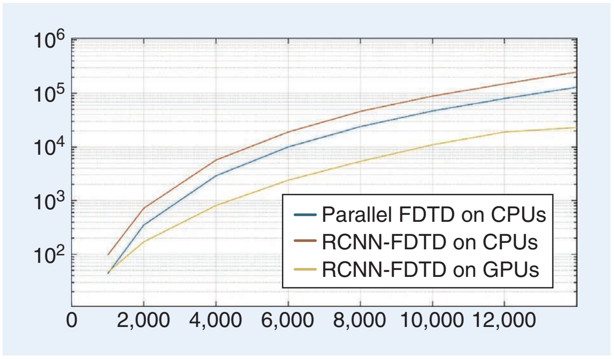

**FIGURE 14.** The computing time versus the number of grids in 2D FDTD.

computed by CPU-FDTD, which further verifies the accuracy of this scheme.

Table 3 compares the computational time between the two algorithms. Here, the center frequency $f_0$, delay time $t_0$, and $\tau$ are fixed, as is the radius of the cylinder. Keeping the relative relationship of the cylinder and source position in the domain, the number of time-marching steps is four times the grid number in the $x$-axis, enlarging the simulating domain in the $x$- and $y$-axes synchronously with the same scale. The computing times with different numbers of grids are shown in the table.

The computing times are also plotted with the number of grids along the $x$-axis, as shown in Figure 14. We can observe that the RCNN-FDTD on the GPU is 5.7 times faster than the parallel FDTD on the CPU. When executed on a CPU, RCNN-FDTD gets 1.9 times slower because the convolution in RCNN-FDTD has two convolutions in the update of the $E$- and $H$-fields in 2D FDTD that are executed apart, while in traditional FDTD, they are performed at the same time by careful programming. Even then, RCNN-FDTD has an acceptable efficiency. In these problems, the ability of massive parallelization on GPUs and CPUs can be fully realized with the help of a machine learning framework.

## DISCUSSION AND CONCLUSIONS

In this study, we investigate the similarities in the computing structures between FDTD and an RCNN. Based on this observation, we implemented FDTD on the machine learning framework. Putting it another way, we may also say that we designed an RCNN that does not need training to solve EM modeling problems with an accuracy that is the same as that of FDTD. Numerical examples verify the accuracy and efficiency of RCNN-FDTD. With the help of massive parallelization, RCNN-FDTD outperforms parallel FDTD on a CPU. A further comparison between a GPU-FDTD and the RCNN-FDTD should be carried out. This has not been done here because we lack a finely optimized GPU-FDTD. We will continue to work on it.

The idea in this study is not abstract as long as we could open up the black box of machine learning and focus on the computing elements between deep neural networks and the finite difference scheme. With the help of modern computing platforms designed for deep learning, RCNN-FDTD allows us to release the power of massive parallelization and perform fast FDTD computation. We do not need to finely tune the memory cache and other control parameters as in low-level programming, such as CUDA. This also permits us to easily transfer the algorithm to other computing architectures, as long as the machine learning framework is implemented. One step further, we can also observe the link between CEM and machine learning at the computational level. This may allow us to bring together physics and data to design fast and intelligent algorithms for EM modeling.

## AUTHOR INFORMATION

***Liangshuai Guo*** (gls7768@136.com) is with Tsinghua University, Beijing, 100084, China, where he is pursuing the Ph.D. degree. His research interests include fast algorithms in computational electromagnetics, signal processing, and artificial intelligence applications.

***Maokun Li*** (maokunli@tsinghua.edu.cn) is with Tsinghua University, Beijing, 100084, China. He served as the guest editor of the special issue on electromagnetic inverse problems for sensing and imaging in *IEEE Antennas and Propagation Magazine*.

***Shenheng Xu*** (shxu@tsinghua.edu.cn) is with Tsinghua University, Beijing, 100084, China. His research interests include novel designs of high-gain antennas for advanced applications, artificial electromagnetic structures, and electromagnetic and antenna theories.

***Fan Yang*** (fan_yang@tsinghua.edu.cn) is with Tsinghua University, Beijing, 100084, China. He was an IEEE Antennas and Propagation Society Distinguished Lecturer for 2018–2020. He is a fellow of the Applied Computational Electromagnetics Society and a Fellow of IEEE.

***Li Liu*** (lliu@birentech.com) is with BirenTech, Shanghai, 201114, China. He joined BirenTech as a senior researcher in 2020. His research interests include computational optimization, machine vision, and neural architecture optimization.

## REFERENCES

[1] R. F. Harrington, *Field Computation by Moment Methods*. Piscataway, NJ, USA: IEEE Press, 1993.

[2] J.-M. Jin, *The Finite Element Method in Electromagnetics*. Hoboken, NJ, USA: Wiley, 2015.

[3] K. Yee, “Numerical solution of initial boundary value problems involving Maxwell’s equations in isotropic media,” *IEEE Trans. Antennas Propag.*, vol. 14, no. 3, pp. 302–307, 1966.

[4] A. Taflove and S. C. Hagness, Computational electrodynamics: The finite-difference time-domain method, in *The Electrical Engineering Handbook*, vol. 3, Elsevier, 2005.

[5] H. Bao and R. Chen, “An efficient domain decomposition parallel scheme for leapfrog ADI-FDTD method,” *IEEE Trans. Antennas Propag.*, vol. 65, no. 3, pp. 1490–1494, 2017, doi: 10.1109/TAP.2016.2647587.

[6] X. L. Zhou, X. Y. Wang, J. F. Zhang, and J. W. You, “Fast and accurate RCS evaluation via high-performance parallel FDTD simulation,” *J. Eng.*, vol. 2019, no. 21, pp. 7322–7325, 2019, doi: 10.1049/joe.2019.0489.

[7] C. Guiffaut and K. Mahdjoubi, “A parallel FDTD algorithm using the MPI library,” *IEEE Antennas Propag. Mag.*, vol. 43, no. 2, pp. 94–103, 2001, doi: 10.1109/74.924608.

[8] S. E. Krakiwsky, L. E. Turner, and M. M. Okoniewski, “Acceleration of finite-difference time-domain (FDTD) using graphics processor units (GPU),” in *Proc. IEEE MTT-S Int. Microw. Symp. Dig.*, 2004, vol. 2, pp. 1033–1036, doi: 10.1109/MWSYM.2004.1339160.

[9] R. Girshick, “Fast R-CNN,” in *Proc. IEEE Int. Conf. Comput. Vision (ICCV)*, 2015, pp. 1440–1448, doi: 10.1109/ICCV.2015.169.

[10] A. Sharif Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, "CNN features off-the-shelf: An astounding baseline for recognition," in *Proc. IEEE Conf. Comput. Vision Pattern Recogn. Workshops*, 2014, pp. 806–813, doi: 10.1109/CVPRW.2014.131.
[11] L. Jiang, H. Yao, H. Zhang, and Y. Qin, "Machine learning based computational electromagnetic analysis for electromagnetic compatibility," in *Proc. Int. Conf. Comput. Electromagn.*, 2018, pp. 1–2, doi: 10.1109/COMPEM.2018.8496540.
[12] Z. Wei and X. Chen, "Deep-learning schemes for full-wave nonlinear inverse scattering problems," *IEEE Trans. Geosci. Remote Sens.*, vol. 57, no. 4, pp. 1849–1860, 2018, doi: 10.1109/TGRS.2018.2869221.
[13] G. Hinton, "Deep learning—A technology with the potential to transform health care," *JAMA*, vol. 320, no. 11, pp. 1101–1102, 2018, doi: 10.1001/jama.2018.11100.
[14] L. Li, L. G. Wang, F. L. Teixeira, C. Liu, A. Nehorai, and T. J. Cui, "DeepNIS: Deep neural network for nonlinear electromagnetic inverse scattering," *IEEE Trans. Antennas Propag.*, vol. 67, no. 3, pp. 1819–1825, 2018, doi: 10.1109/TAP.2018.2885437.
[15] R. Guo, X. Song, M. Li, F. Yang, S. Xu, and A. Abubakar, "Supervised descent learning technique for 2-D microwave imaging," *IEEE Trans. Antennas Propag.*, vol. 67, no. 5, pp. 3550–3554, 2019, doi: 10.1109/TAP.2019.2902667.
[16] L.-Y. Xiao, J. Li, F. Han, W. Shao, and Q. H. Liu, "Dual-module NMM-IEM machine learning for fast electromagnetic inversion of inhomogeneous scatterers with high contrasts and large electrical dimensions," *IEEE Trans. Antennas Propag.*, vol. 68, no. 8, pp. 6245–6255, 2020, doi: 10.1109/TAP.2020.2990222.
[17] K. Xu, L. Wu, X. Ye, and X. Chen, "Deep learning-based inversion methods for solving inverse scattering problems with phaseless data," *IEEE Trans. Antennas Propag.*, vol. 68, no. 11, pp. 7457–7470, 2020, doi: 10.1109/TAP.2020.2998171.
[18] P. Shah and M. Moghaddam, "Super resolution for microwave imaging: A deep learning approach," in *Proc. IEEE Int. Symp. Antennas Propag. USNC/URSI Nat. Radio Sci. Meeting*, 2017, pp. 849–850, doi: 10.1109/APUSNCURSINRSM.2017.8072467.
[19] G. Chen, P. Shah, J. Stang, and M. Moghaddam, "Learning-assisted multimodality dielectric imaging," *IEEE Trans. Antennas Propag.*, vol. 68, no. 3, pp. 2356–2369, 2020, doi: 10.1109/TAP.2019.2948565.
[20] P. Mojabi, M. Hughson, V. Khoshdel, I. Jeffrey, and J. Lovetri, "CNN for compressibility to permittivity mapping for combined ultrasound-microwave breast imaging," *IEEE J. Multiscale Multiphys. Comput. Techn.*, vol. 6, pp. 62–72, Apr. 2021, doi: 10.1109/JMMCT.2021.3076827.
[21] P. Ran, Y. Qin, D. Lesselier, and M. Serhir, "Subwavelength microstructure probing by binary- specialized methods: Contrast source and convolutional neural networks," *IEEE Trans. Antennas Propag.*, vol. 69, no. 2, pp. 1030–1039, 2021, doi: 10.1109/TAP.2020.3016175.
[22] X. Ye, Y. Bai, R. Song, K. Xu, and J. An, "An inhomogeneous background imaging method based on generative adversarial network," *IEEE Trans. Microw. Theory Techn.*, vol. 68, no. 11, pp. 4684–4693, 2020, doi: 10.1109/TMTT.2020.3015495.
[23] A. Massa, G. Oliveri, M. Salucci, N. Anselmi, and P. Rocca, "Learning-by-examples techniques as applied to electromagnetics," *J. Electromagn. Waves Appl.*, vol. 32, no. 4, pp. 516–541, 2018, doi: 10.1080/09205071.2017.1402713.
[24] T. Shan, W. Tang, X. Dang, M. Li, F. Yang, S. Xu, and J. Wu, "Study on a fast solver for Poissonś equation based on deep learning technique," *IEEE Trans. Antennas Propag.*, vol. 68, no. 9, pp. 6725–6733, 2020, doi: 10.1109/TAP.2020.2985172.
[25] H. M. Yao and L. Jiang, "Machine-learning-based PML for the FDTD method," *IEEE Antennas Wireless Propag. Lett.*, vol. 18, no. 1, pp. 192–196, 2018, doi: 10.1109/LAWP.2018.2885570.
[26] A. Massa, D. Marcantonio, X. Chen, M. Li, and M. Salucci, "DNNs as applied to electromagnetics, antennas, and propagation – A review," *IEEE Antennas Wireless Propag. Lett.*, vol. 18, no. 11, pp. 2225–2229, 2019, doi: 10.1109/LAWP.2019.2916369.
[27] X. Chen, Z. Wei, M. Li, and P. Rocca, "A review of deep learning approaches for inverse scattering problems," *Progress Electromagn. Res.*, vol. 167, pp. 67–81, Jun. 2020, doi: 10.2528/PIER20030705.
[28] M. Li et al., "Machine learning in electromagnetics with applications to biomedical imaging: A review," *IEEE Antennas Propag. Mag.*, vol. 63, no. 3, pp. 39–51, 2021, doi: 10.1109/MAP.2020.3043469.
[29] R. Guo et al., "Solving combined field integral equation with deep neural network for 2-D conducting object," *IEEE Antennas Wireless Propag. Lett.*, vol. 20, no. 4, pp. 538–542, 2021, doi: 10.1109/LAWP.2021.3056460.
[30] R. Guo et al., "Physics embedded deep neural network for solving volume integral equation: 2D case," *IEEE Trans. Antennas Propag.*, early access, 2021, doi: 10.1109/TAP.2021.3070152.
[31] J. Li and C. Miao, "An efficient FDTD implementation of the CFS-PML based on the ADE method and its validation along with the PLRC method in dispersive media," in *Proc. Int. Conf. Microw. Millimeter Wave Technol.*, 2008, vol. 2, pp. 766–769, doi: 10.1109/ICMMT.2008.4540510.
[32] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, p. 436, 2015, doi: 10.1038/nature14539.
[33] Easy TensorFlow. https://www.easy-tensorflow.com/tf-tutorials/convolutional-neural-nets-cnns (accessed Aug. 2020).
[34] M. Abadi et al., "Tensorflow: A system for large-scale machine learning," in *Proc. 12th {USENIX} Symp. Operating Syst. Des. Implementation ({OSDI} 16),* 2016, pp. 265–283.
[35] N. Ketkar, "Introduction to Pytorch," in *Deep Learning with Python*. Berlin: Springer-Verlag, 2017, pp. 195–208.
[36] T. W. Hughes, I. A. D. Williamson, M. Minkov, and S. Fan, "Wave physics as an analog recurrent neural network," *Sci. Adv.*, vol. 5, no. 12, 2019, doi: 10.1126/sciadv.aay6946.
[37] Y. Li, G. Lei, G. Bramerdorfer, S. Peng, X. Sun, and J. Zhu, "Machine learning for design optimization of electromagnetic devices: Recent developments and future directions," *Appl. Sci.*, vol. 11, no. 4, p. 1627, 2021, doi: 10.3390/app11041627.
[38] Y. Hu, Y. Jin, X. Wu, and J. Chen, "A theory-guided deep neural network for time domain electromagnetic simulation and inversion using a differentiable programming platform," *IEEE Trans. Antennas Propag.*, early access, 2021, doi: 10.1109/TAP.2021.3098585.
[39] L. Guo, M. Li, S. Xu, and F. Yang, "Study on a recurrent convolutional neural network based FDTD method," in *Proc. Int. Appl. Comput. Electromagn. Soc. SY~P.—China (ACES)*, 2019, vol. 1, pp. 1–2, doi: 10.23919/ACES48530.2019.9060707.
[40] J.-P. Berenger, "A perfectly matched layer for the absorption of electromagnetic waves," *Comput. Phys.*, vol. 114, no. 2, pp. 185–200, 1994, doi: 10.1006/jcph.1994.1159.
[41] W. C. Chew and W. H. Weedon, "A 3D perfectly matched medium from modified Maxwell's equations with stretched coordinates," *Microw. Opt. Technol. Lett.*, vol. 7, no. 13, pp. 599–604, 1994, doi: 10.1002/mop.4650071304.
[42] J. Wang, O. Fujiwara, S. Kodera, and S. Watanabe, "FDTD calculation of whole-body average SAR in adult and child models for frequencies from 30 MHz to 3 GHz," *Phys. Med. Biol.*, vol. 51, no. 17, p. 4119, 2006, doi: 10.1088/0031-9155/51/17/001.
[43] L. Xu and Y. Xu, "Implementation and optimization of three-dimensional UPML-FDTD algorithm on GPU clusters," in *Proc. Int. Supercomput. Conf.*, 2014, pp. 478–486.
[44] M. Ruiz-Cabello, N. M. Abaļenkovs, L. M. Diaz Angulo, C. Cobos Sanchez, F. Moglie, and S. G. Garcia, "Performance of parallel FDTD method for shared-and distributed-memory architectures: Application tobioelectromagnetics," *Plos One*, vol. 15, no. 9, p. e0238115, 2020, doi: 10.1371/journal.pone.0238115.
[45] J. E. Diener and A. Z. Elsherbeni, "FDTD acceleration using MATLAB parallel computing toolbox and GPU," *Appl. Comput. Electromagn. Soc. J.*, vol. 32, no. 4, 2017.
[46] A. Weiss, A. Elserbeni, V. Demir, and M. Hadi, "Accelerating the FDTD algorithm on CPUs with MATLAB's parallel computing toolbox," in *Proc. Int. Appl. Comput. Electromagn. Soc. Symp*. (ACES), 2019, pp. 1–2.
[47] T.-Y. Kim and S.-B. Cho, "Predicting residential energy consumption using CNN-LSTM neural networks," *Energy*, vol. 182, pp. 72–81, Sep. 2019, doi: 10.1016/j.energy.2019.05.230.
[48] A. Sherstinsky, "Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network," *Phys. D: Nonlinear Phenomena*, vol. 404, p. 132306, 2020, doi: 10.1016/j.physd.2019.132306.
[49] A. F. Agarap, "Deep learning using rectified linear units (relu)," 2018, arXiv:1803.08375.
[50] B. Karlik and A. V. Olgac, "Performance analysis of various activation functions in generalized MLP architectures of neural networks," *Int. J. Artif. Intell. Expert Syst.*, vol. 1, no. 4, pp. 111–122, 2011.
[51] Citing TenserFlow. https://www.tensorflow.org/about/bib (accessed Nov. 2021).
[52] CUDA toolkit documentation. https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html (accessed Nov. 2021).
[53] M. Livesey, J. F. Stack, F. Costen, T. Nanri, N. Nakashima, and S. Fujino, "Development of a CUDA implementation of the 3d FDTD method," *IEEE Antennas Propag. Mag.*, vol. 54, no. 5, pp. 186–195, 2012, doi: 10.1109/MAP.2012.6348145.
[54] M. F. Su, I. El-Kady, D. A. Bader, and S.-Y. Lin, "A novel FDTD application featuring OpenMP-MPI hybrid parallelization," in *Proc. Int. Conf. Parallel Process.*, 2004, pp. 373–379, doi: 10.1109/ICPP.2004.1327945.