Progress In Electromagnetics Research M, Vol. 112, 15–28, 2022

A Neural Network Representation of Generalized Multiparticle Mie-Solution

Ying Li Thong and Tiem Leong Yoon*

Abstract—Generalized Lorentz-Mie Theory (GLMT) provides analytical far-field solutions to electromagnetic (EM) scattering of an aggregate of spheres in a fixed orientation. One of the computational codes that implements the GLMT calculation is provided by Xu, dubbed GMM. which returns EM responses such as the extinction cross section, σ_{ext}, given the information of incident wavelength, particle arrangement, common radius, and reflective indices of the aggregate. We have attempted to represent the GMM code in the form a neural network dubbed NNGMM. The NNGMM obtained was stress tested and systematically quantified for its accuracy by comparing the σ_{ext} predicted against that produced by the original GMM code. The σ_{ext} produced by the NNGMM for arbitrary aggregates at random wavelength yielded a good fidelity with respect to that calculated by the GMM calculator up to an R-squared value of above 99% level and mean squared error of ≈ 5.0. The realization of NNGMM proves the feasibility of representing the GMM code by a neural network. The optimally-performing NNGMM obtained in this work can serve as an alternative computational tool for calculating σ_{ext} in place of the original GMM code at a much cheaper cost, albeit with a slight penalty in terms of absolute accuracy.

1. INTRODUCTION

The scattering of electromagnetic (EM) radiation with an aggregate made up of spherical particles is a complicated process that could be modelled from first-principles within the framework of Maxwell's equations. The simplest solution to this problem was proposed by Mie Gustav in 1908 for the simplest case of a homogeneous sphere [1]. The solution to the general case of multiparticle scattering for an arbitrary aggregate made up of spherical particles is referred to as the generalized Lorentz-Mie theory (GLMT). There are many rigorous models at different levels of approximation to solve such a complex calculation in EM scattering [2–19]. One of such models is the Generalized Multi-particle Mie-solution (GMM) [13–18, 20–25] which provides analytical far-field solutions to electromagnetic scattering by an aggregate of spheres in a fixed orientation. GMM was implemented as a Fortran package by Xu [26–29]. It can be used to compute the optical responses of an EM scattering event with a multiparticle aggregate, such as the cross sections of the scattered EM radiation, extinction, or absorption curves as a function of scattering angle at fixed wavelength. The aggregate is assumed to be made of non-intersecting homogeneous spheres embedded in a vacuum surrounding. GMM has been shown to provide impressive agreement with experimental results as compared with the GLMT and geometrical optics results [26]. The input information required by the GMM code includes geometrical arrangement in 3-dimension, common radius, and both the real and complex refractive indices of each sphere in the multi-particle aggregate. EM responses such as the extinction cross section (σ_{ext}) in a scattering event between an incident EM radiation and the aggregate is returned as an output.

Received 5 May 2022, Accepted 1 July 2022, Scheduled 24 July 2022

* Corresponding author: Tiem Leong Yoon (tlyoon@usm.my).
The authors are with the School of Physics, Universiti Sains Malaysia, 11800 USM, Penang, Malaysia.

Quite generally, the computational cost of the GMM code increases with size parameter, i.e., the number of spheres in the aggregate. In practice, the GMM code is limited by the amount of RAM in the computer hardware. The code was developed around the end of 1990s, and in principle can handle calculations involving a very large number of spheres. In practice, however, the maximum number of spheres that can be handled by the code is limited by the available memory in the computer hardware.

From a computational perspective, the GMM code reads in the input of geometrical arrangement of the spheres sitting in 3-dimension space, radius, and reflective indices (of the spheres), to return an EM response based on the generalized multiparticle Mie solution. In this sense, the GMM code is merely a 'calculator' that output non-trivial numerical results when it is fed with a list of numerical input representing a particular 'point' in the multi-dimensional configuration space of the aggregates. In the subsequent discussion, we shall use the term 'configuration' to refer to the set of numerical lists required by the GMM as input for producing an optical response in an EM scattering event. Specifically, the configuration of an aggregate made up of N spheres with a common radius composed of the following variables: $\{x_i, y_i, z_i; r; N; \lambda; n, \kappa\}$, $i \in [1, N]$. Here, $\{x_i, y_i, z_i\}$ refers to the variables representing the x-, y-, and z-coordinates, r the common radius of the spheres, N the number of spheres, λ the incident EM wavelength, $\{n, \kappa\}$ the real and imaginary parts of the reflective indices which are assumed common to all spheres, respectively. Given a list of numerical values $\{x_i, y_i, z_i; r; N; \lambda; n, \kappa\}$, $i \in [1, N]$, the GMM code shall return either a converged numerical output or otherwise (e.g., when the coordinates of distinct spheres overlap, or the numerical solution to the GLMT is not found).

Acting as a 'calculator', the GMM code maps input configuration data $\{x_i, y_i, z_i; r; N; \lambda; n, \kappa\}$, $i \in [1, N]$ living in a large dimensional space (where each variable is continuous) to a numerical output (e.g., a cross section of extinction σ_{ext}) living in a 1D continuous parameter space. A large amount of such configuration data $\{x_i, y_i, z_i; r; N; \lambda; n, \kappa\}$, $i \in [1, N]$, can be artificially generated and fed into the GMM code to produce a corresponding converged or non-converged numerical output. This opens up an interesting question: Suppose that a neural network (NN) is trained with a sufficiently large number of artificially generated random configurations. Is such a trained NN capable of predicting, to a reasonably acceptable level of accuracy, the extinction cross section of configurations to which the NN has never been exposed before? For the sake of easy referencing, such a NN that can supposedly represent the GMM calculator is dubbed 'Neural Network representation of the Generalize Multiparticle Mie-solution', or 'NNGMM' for short, hereafter. In machine learning jargon, this is known as a regression problem, in which numerical prediction of a continuous output is made based on patterns or rules identified from the training dataset.

The major objective of the present work is to demonstrate a proof of concept whereby the inherent knowledge coded in the GMM program can be transferred into the weights of the trained neutral network [which in practice is digitally saved in the form of, e.g., a *.h5 file in the Hierarchical Data Format (HDF)]. In other words, we wish to show that the GMM code can be represented (or approximated) by a regression model. It is known that any complex function can be approximated using powerful function fitting method based on deep neural networks [30]. Neural networks have excellent performance for function fitting. As such, NN is a natural choice for building a regression model as it offers as many free parameters (i.e., the weights relating the nodes in the neural network) as required to approximate the GMM as a single function of multivariable input (i.e., the coordinates, size, and reflective indices of the aggregates). The NNGMM can be considered as a proxy that can practically 'represent' the GMM calculator without the need to know the detailed algorithm coded into the GMM theory.

The present work, prompted by the above-mentioned motivation, reports the results of our attempt to realize such a possibility. It was found that the NNGMM program could achieve a good level of accuracy in predicting the extinction cross section based on the metric measured in terms of R-squared value and mean squared error.

It is worth mentioning that the present attempt to represent the GMM by an NN is relevant to the so called 'forward electromagnetic problem' [31]. In this problem, one seeks to obtain the information of the EM scattered fields given the information of the scatterers in the computational domain and the incidences. Machine learning based algorithms have been adopted as an alternative approach to devise the so-called EM 'solvers' in place of convention computational electromagnetic methods, such as finite difference method (FDM) [32, 33], finite element method (FEM) [34], boundary element method (BEM) [35], method of moments (MoM), and their variants [36, 37]. These are computationally costly methods for solving forward EM problems, especially for electrically large structures. The book by Ren et al. [38] provides a detailed discussion on the technicality of applying deep learning (DL) techniques in EM scattering problems, assessing its potential to replace traditional numerical solvers in real-time forecast scenarios. Generally speaking, ML based EM solvers are gaining popularity, see, e.g., [39–43]. For example, [43] proposes a generative adversarial network (GAN) based fast electromagnetic scattering solver. It is a full-fledged learning-based approach to solve forward EM scattering problems for a dielectric scatterer located in a predefined spatial domain. Generally speaking, ML based approaches for solving the forward EM scattering problem reported in the literature vary from one to another in terms of scope, system, theoretical strategy, implementation details, and the type of ML algorithm used. As a comparison, the scope of the present manuscript is simpler. It mainly aims to demonstrate the feasibility of representing a GMM, a specific form of forward EM solver, by a NN.

The paper is organized as follows. In Section 2, the methodology for generating configurations for the purpose of training and testing is elaborated. The procedure of training and testing the trained NN is also detailed in the same section. In Section 3, we discuss the results and interpret the performance of the trained NN. A conclusion follows in Section 4.

2. METHODOLOGY

The program of creating a NN representation of GMM was divided into two stages. The first stage involves the generation of training and testing configuration data, while the second retraining and stress testing of the *.h5 files. These two stages are detailed in the following two subsections.

2.1. Data Generation

The GMM Fortran code reads in the configuration of an aggregate in the form of a data file with a common file name 'bk7s2.k'. The format of the 'bk7s2.k' is shown in Table 1. The format of the bk7s2.k file is as follows:

- The first line refers to the wavelength λ, in nm.
- The integer in second row N refers to the number of spheres.
- In row three or above, the lines are interpreted as $x_i, y_i, z_i, r, n, \kappa$.

The following lists the lower and upper limits of each of the variables in the bk7s2.k input file:

- $\lambda \in [\lambda_{\text{init}}, \lambda_{\text{last}}]$
- $N \in [1, N_{\max}]$, $N_{\max}$ a positive integer equal or larger than 1.
- $r \in [r_{\min}, r_{\max}]$, with $r_{\min} = 0.99r_0$, $r_{\max} = r_0 + 3r_{\min}$. r_0 a positive real value in units of nm representing a reference radius.
- $x_i \in [x_{\min}, x_{\max}] = [-L/2, L/2]$, where $L = 2r_{\max} \cdot N_{\max} \cdot f$, f a positive real factor. y_i and z_i share the same lower and upper limits as x_i.
- $n \in [n_{\min}, n_{\max}]$, with $n_{\min}, n_{\max}$ real, positive numbers.
- $\kappa \in [\kappa_{\min}, \kappa_{\max}]$, with $\kappa_{\min}, \kappa_{\max}$ real, positive numbers.

Table 1. The format of the bk7s2.k file used as input to the GMM code.

λ
N
$x_1, y_1, z_1, r, n, \kappa$
$x_2, y_2, z_2, r, n, \kappa$
$x_3, y_3, z_3, r, n, \kappa$
...
$x_N, y_N, z_N, r, n, \kappa$

All variables in the bk7s2.k file, namely, $\{x_i, y_i, z_i; r; N; \lambda; n, \kappa\}$, $i \in [1, N]$ were generated randomly within their respective limits as stated. The lower and upper limits of these random variables in turn were controlled by the free parameters $\lambda_{\rm init}, \lambda_{\rm last}, N_{\rm max}, r_0, f, n_{\rm min}, n_{\rm max}, \kappa_{\rm min}, \kappa_{\rm max}$. Table 2 lists the values of these free parameters used in this work. Each random configuration could contain different numbers of spheres, equal or less than the positive integer $N_{\rm max}$.

Table 2. Free parameters used to control the ranges of the input variables in Table 1.

Variable	values
$[\lambda_{\rm init}, \lambda_{\rm last}]$	$[50\,{\rm nm}, 100\,{\rm nm}]$
$N_{\rm max}$	10
r_0	1 nm
f	0.99
$[n_{\rm min}, n_{\rm max}]$	[0.1, 1.0]
$[\kappa_{\rm min}, \kappa_{\rm max}]$	[0.1, 1.0]

A simple shell script was written to generate a bk7s2.k file that contains random values assigned to the variables $\{x_i, y_i, z_i; r; N; \lambda; n, \kappa\}$, $i \in [1, N]$, according to the above prescription. Once the random input data in the bk7s2.k was generated, it was fed into the GMM calculator for execution to return an output file 'gmm01f.out'. It contained the numerical responses calculated by the GMM calculator for an EM wave interacting with the aggregate, among others, the extinction ($\sigma_{\rm ext}$), absorption ($\sigma_{\rm abs}$), scattered ($\sigma_{\rm sca}$) cross sections at zero Euler angles, and the dimensionless polarization components of the scattered intensity as a function of scattering angle. In the present work, for the sake of demonstrating the proof-of-concept of the proposed NNGMM approach, only the $\sigma_{\rm ext}$ cross section was monitored as the numerical target. This process is depicted in Fig. 1.

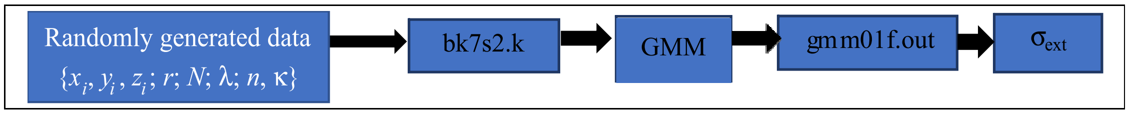


Figure 1. Flowchart of a randomly generated input file bk7s2.k being fed into the GMM calculator to produce a value of extinction cross section $\sigma_{\rm ext}$.

The value of $\sigma_{\rm ext}$ calculated by GMM was appended into a *.npy file named 'label.npy' while the corresponding input configuration $\{x_i, y_i, z_i; r; N; \lambda; n, \kappa\}$, $i \in [1, N]$ into config.npy. The *.npy file format is popularly used in Python software package for intensive data processing. A simple program was developed to repeatedly generate many configurations and the corresponding $\sigma_{\rm ext}$'s to populate a config.npy file (and the corresponding label.npy file) with $\sim 10^5$ non-repeated random configurations. Not all of the generated configurations would produce a converged output by the GMM code. Non-convergent random configurations were filtered out and not appended into the config.npy file. A specifically designed boundary condition was imposed in the random generation of the configuration to minimize overlapping spheres, so that about $\sim 25\%$ of the total random configurations generated got filtered out. These random configurations were divided in a mutually exclusive manner into two separate directories, data_train/ and data_test/. data_train/ contains 170 while data_test/ 19 pairs of {config.npy, label.npy} files. Each of the paired {config.npy, label.npy} files contains approximately $\sim 10^5$ random configurations (after filtration). The files in the data_train/ directory were to be used for training the NN model while those in data_test/ were to be used for stress testing the retrained model.

As a detailed technical note, in a config.npy file, a single random configuration containing N sphere is represented by a row vector with $4N_{\rm max} + 4$ components, despite $N \leq N_{\rm max}$. The first $4N_{\rm max}$ components were reserved for $\{x_i, y_i, z_i; r\}$, $i \in [1, N]$, with the last 4 for $\{N, \lambda, n; \kappa\}$. Those components reserved for $\{x_i, y_i, z_i; r\}$ where $i \in [N + 1, N + 2, ..., N_{\rm max}]$ would be simply set to zero. The shape of the row vectors in config.npy was designed in this manner to provide flexibility to accommodate the varying number of spheres N in each random configuration.

2.2. Incremental Retraining and Systematic Monitoring of the NN

A neural network which can 'predict' the value of σ_{ext} when it was presented with an arbitrary configuration was to be trained. To this end, a multi-layer neural network, namely, the Sequential model tensorflow.keras.models package from Keras, a high-level API that is built on top of TensorFlow [44, 45], was compiled and saved after it was trained with a data set. The saved model, which is basically a *.h5 file that stores the weights and biases of the neurons in its layers, was then called to make prediction against all data sets kept in the data_test/ directory.

Each NN model was compiled with a specific set of hyperparameters. The hyperparameters included the number of layers (depth of the NN), number of neurons (width of the NN), dropout rate, activation function, and epochs. Table 3 lists illustrative values of hyperparameters used in the NN. In particular, the ReLu activation function, defined as $f(x) = \max(0, x)$, was used throughout our work. The NN obtained with this activation function produces acceptably good accuracy (as discussed in Section 3). With a fixed set of hyperparameters, a model was first compiled and trained by feeding the data using a {config.npy, label.npy} data pair from the data_train/ directory. For the sake of easy referencing, this model shall be identified as 0.h5. It was subjected to a series of stress tests by using it to predict the corresponding σ_{ext} value of all config.npy files sitting in the data_test/ directory. The values of σ_{ext} predicted by the model 0.h5 for all config.npy files, each composed of $N_c \sim 10^5$ random configurations, were compared against the values in the corresponding ground-truth files (i.e., label.npy). The process as described above is depicted in Fig. 2.

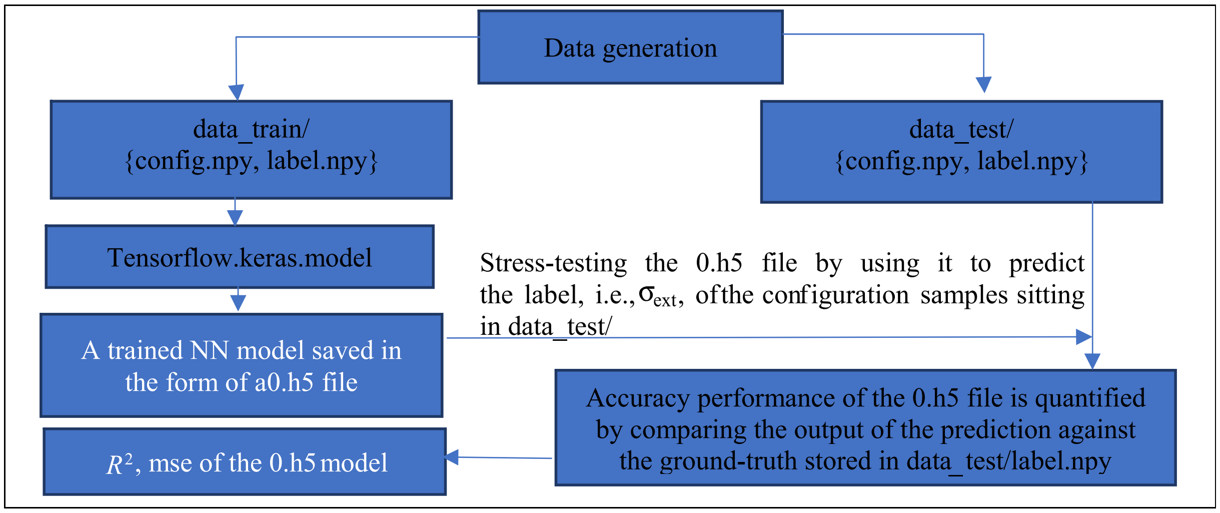

Data generation
data_train/ {config.npy, label.npy}
data_test/ {config.npy, label.npy}
Tensorflow.keras.model
Stress-testing the 0.h5 file by using it to predict the label, i.e.,σext, ofthe configuration samples sitting in data_test/
A trained NN model saved in the form of a0.h5 file
Accuracy performance of the 0.h5 file is quantified by comparing the output of the prediction against the ground-truth stored in data_test/label.npy
R², mse of the 0.h5 model

Figure 2. Flowchart showing the training of a .h5 file using a {config.npy, label.npy} data set stored in the directory data_train/. The resultant .h5 file is then used to predict the output of sample configurations stored in the directory data_test/config.npy. The performance accuracy of 0.h5 is evaluated in the form of R^2 and mse.

The comparison can be conveniently visualized by displaying a scatter plot of $\sigma_{\text{ext}}(\text{prediction})$ vs. $\sigma_{\text{ext}}(\text{ground truth})$ (for an illustrative example, see the scatter plots in the last column in Fig. 4). By examining such scatter plots, the qualitative agreement between the predicted values $\sigma_{\text{ext}}(\text{prediction})$ and $\sigma_{\text{ext}}(\text{ground truth})$ can be conveniently visualized. E.g., if the data points in these scatter plot distribute closely about the $y = x$ symmetrical line, the agreement is good, and vice versa. The accuracy of the prediction by a NN model could be quantitatively quantified by calculating the mean square error (mse) and coefficient of determination (R^2, a.k.a. R-squared) values of the $\sigma_{\text{ext}}(\text{prediction})$ vs. $\sigma_{\text{ext}}(\text{ground truth})$ plot. R-squared value, which is related to the fraction of variance unexplained (FVU), is defined as per

$$R^2 = 1 - \text{FVU}, \quad \text{FVU} = \frac{\text{SS}_{\text{res}}}{\text{SS}_{\text{tot}}}, \tag{1}$$

$$\mathrm{SS_{res}} = \sum_{i=1}^{N_c} \left(\sigma_{\mathrm{ext},i}\,(\mathrm{prediction}) - \sigma_{\mathrm{ext},i}\,(\mathrm{ground\,truth})\right)^2 , \tag{2}$$

$$\mathrm{SS_{tot}} = \sum_{i=1}^{N_c} \left(\sigma_{\mathrm{ext},i}\,(\mathrm{prediction}) - \overline{\sigma_{\mathrm{ext}}}\right)^2 , \tag{3}$$

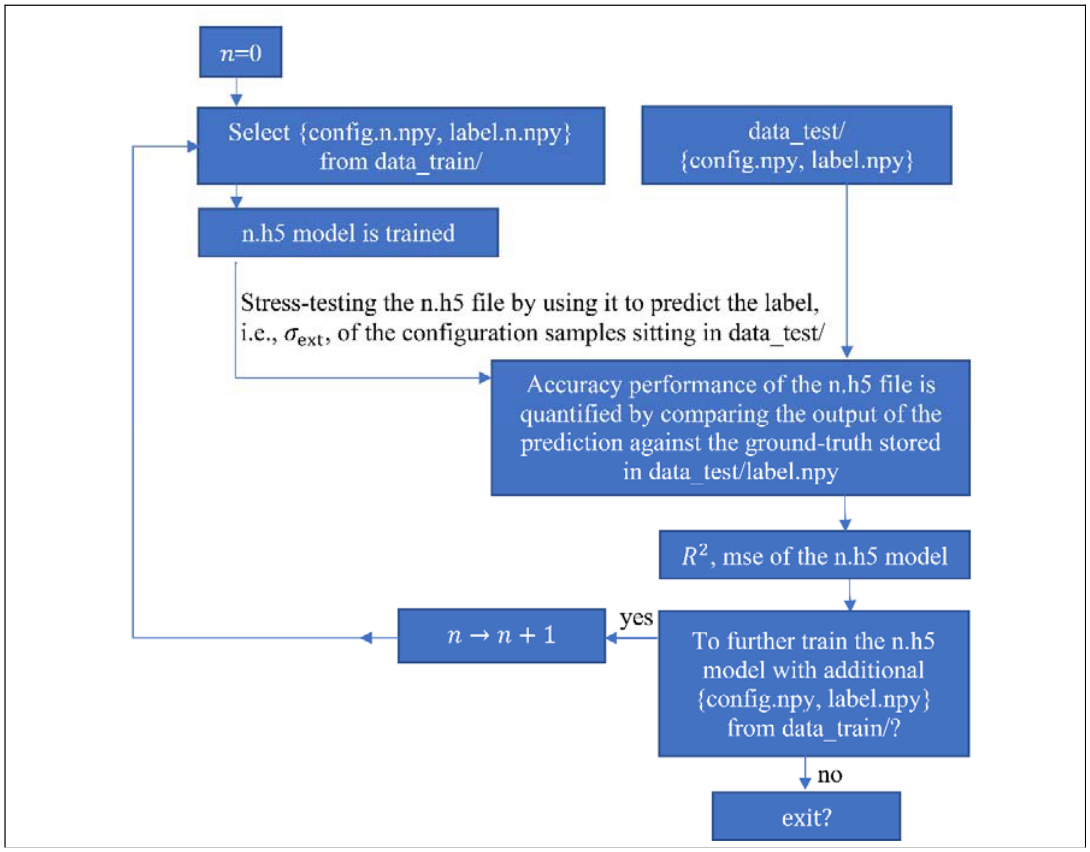
$$\overline{\sigma_{\mathrm{ext}}} = \frac{1}{N_c} \sum_{i=1}^{N_c} \sigma_{\mathrm{ext},i}\,(\mathrm{prediction}) \tag{4}$$

mse is defined as

$$\mathrm{mse} = \frac{1}{N_c} \sum_{i}^{N_c} \left(\sigma_{\mathrm{ext},i}\,(\mathrm{prediction}) - \sigma_{\mathrm{ext},i}\,(\mathrm{ground\,truth})\right)^2 \tag{5}$$

where $\sigma_{\mathrm{ext},i}$ refers to the extinction cross section associated with a random configuration i, and N_c is the total number of random configurations in a config.npy file. Note that the values of σ_{ext} generated by the GMM calculator approximately lie in the range ~ 0 to $\sim 6 \times 10^2$.

The results can also be presented in the form of a scatter plot in which both the values of $\sigma_{\mathrm{ext}}(\mathrm{ground\,truth})$ and $\sigma_{\mathrm{ext}}(\mathrm{prediction})$ were plotted at their common configuration represented by a numbered integer in the horizontal axis. An example of $\sigma_{\mathrm{ext}}(\mathrm{prediction})$, $\sigma_{\mathrm{ext}}(\mathrm{ground\,truth})$ scatter plots as a function of their common configuration label is shown in the last third column from the right in Fig. 4.



n=0
Select {config.n.npy, label.n.npy} from data_train/
data_test/ {config.npy, label.npy}
n.h5 model is trained
Stress-testing the n.h5 file by using it to predict the label, i.e., σ_{ext}, of the configuration samples sitting in data_test/
Accuracy performance of the n.h5 file is quantified by comparing the output of the prediction against the ground-truth stored in data_test/label.npy
R^2, mse of the n.h5 model
To further train the n.h5 model with additional {config.npy, label.npy} from data_train/?
yes
$n \rightarrow n+1$
no
exit?

Figure 3. Flowchart showing the process of 'retraining' and 'systematic monitoring' of the NN models. It is a generalization of the procedure as depicted in Fig. 2, where the n.h5 model is incrementally updated with more config.npy data sitting in the data_train/ directory.

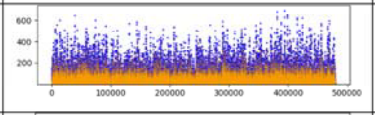
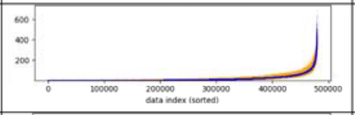
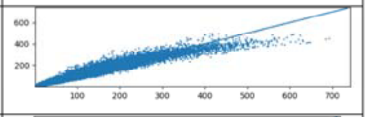
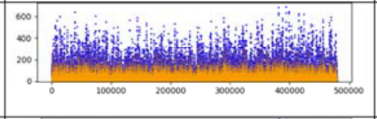
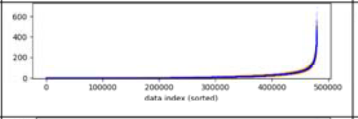
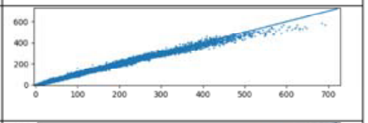
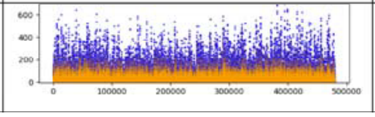
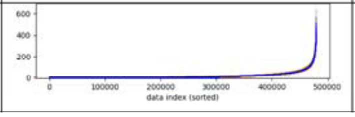
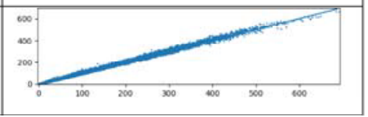
Hyperparameter set	*.h model	Number of accumulated data o train the model	Metric: R-squared	Metric: mse	Scatter plot of σ_{ext}(prediction) and σ_{ext}(ground truth) as a function of common configuration label. Blue (orange) dots represent ground truth (prediction).	Sorted version of the scatter plot, where σ_{ext}(prediction), σ_{ext}(ground truth) sorted in ascending order are plotted as a monotonically increasing curve.	σ_{ext}(prediction) vs. σ_{ext}(ground truth).
p2@c22	1.h5	366540	0.96	46.32			
	21.h5	4030760	0.99	6.78			
	40.h	7505169	1.0	4.0			

Figure 4. The test results of the NN model compiled with hyperparameter set p2@c22 for three selected stages retraining, namely 1.h5, 21.h5 and 40.h, when they were subjected to a common stress test using configuration file test.479600_c_p10@c27.

A third way for visual comparison of the prediction against the ground truth was to sort the values of the σ_{ext}(ground truth) (blue dots) and σ_{ext}(prediction) (orange dots) in ascending order and then display them as a monotonically increasing curve on a common x-y plane. Both sorted σ_{ext}(ground truth) and σ_{ext}(prediction) curves obtained were then overlaid on the same plot such as that illustrated in the last second column from the right in Fig. 4.

The three types of scatter plots mentioned above display the overlapping pattern of the predicted and the ground truth values for all configuration labels in a given config.npy file, serving a useful perspective of how well the predicted and ground-truth data sets agree to each other.

The model .h5, after finishing its stress test, was retrained by using another {config.npy, label.npy} data pair from the data_train/ directory at a fixed learning rate. The resultant retrained model, which was labelled as 1.h5, was expected to contain the knowledge already acquired in .h5 plus that from the current training data. 1.h5 was subjected to the same stress tests as experienced by .h5 using the same set of test data in the data_test/ directory. The average mse and R-squared value of the stress tests for 1.h5 were calculated and recorded. The same procedure was repeated, each time using a new training data pair stored in data_train/ to produce a new retrained *.h5 model file. The performance metric of the successive *.h5 models, such as 2.h5, 3.h5, ..., was consecutively monitored. The two averaged metrics (i.e., mse and R-squared) for each *.h5 model were calculated based on a total of 7,768,766 random configurations separately stored in 19 config.npy files in the data_test/ directory. The huge number of random configurations used in the stress tests should be statistically sufficient as a justifiable means to measure the robustness and accuracy of the retrained NN models. It is noted that a higher numbered *.h5 model was trained with a relatively larger random data than a lowered number *.h5 file. As such it is logically feasible to anticipate that the performance of the *.h5 model would grows as it was trained with more data until it hits a saturation level above which additional training data of random samples would not lead to performance improvement. The above-mentioned procedure is dubbed 'retraining' and 'systematic monitoring'. Fig. 3 illustrates the flowchart of this procedure.

The hyperparameters of the NN models in the present study were fine-tuned in the following way. As mentioned in the paragraph above, the performance of any given *.h5 model was measured in terms of average R-squared value and mse over the test data sets. The two performance metrics for all incrementally increasing *.h5 models were first measured at a fixed set of hyperparameters composed of {depth of the NN, width of the NN, dropout rate, learning rate, activation function, batch size, epochs}. The values in the set of hyperparameters were then systematically varied. For each set of varied hyperparameter values, the two performance metrics for all incrementally increased *.h5 models were measured. The trend of the performance metrics in the incrementally increased *.h5 models as a function of hyperparameter set was then systematically monitored and compared. Generally, NN models compiled with different parameter sets would display different trends of performance. The desired trend would be one where the accuracy performance gets saturated to an R-squared value $\sim$ 1.00 and a minimum mse after a NN model was trained beyond a threshold amount of training data.

When a model underwent a training process, the weights in the NN were updated in a cyclic manner for a number of rounds specified by the 'epochs' hyperparameter. One of hyperparameters, namely, validation_split, was set to a value of validation_split = 0.2 in the present study. In each training epoch, a fraction 0.7 (= validation_split) of the given training data was chosen randomly for training the model. The rest of the training data, 0.3(= 1 − validation_split), was used for the validation of the retrained model in that epoch. The effective accuracy of the NN in predicting σ_{ext} was not sensitive to the hyperparameter validation_split. For pragmatic application of the NN representation of the GMM calculator, the accuracy metric of the validation on the NN model during the epoch training cycles was not an important indicator to monitor. The effective accuracy of the NN model was not determined by the validation metric of the model in training epochs. A retrained NN model displaying very impressive validation metrics at the end of the epoch cycles was not necessarily capable of producing prediction accuracy at a similar level when it was challenged with not-seen-before configurations. From a practical point of consideration, ultimately the accuracy of the NN model was to be independently justified via the stress tests using the test data sets but not the validation metric during training epochs. Another hyperparameter, namely the loss function (which was set to 'mean_squared_error'), was also insensitive to the accuracy of the NN model. As such, both hyperparameters were not tuned in the present study. The mean squared error loss function is defined via mse $= \frac{1}{N}\sum_i^N (y_i - \hat{y}_i)^2$, where $\hat{y}_i$ are the expected or target outputs; y_i are the predicted outputs from the NN; and N is the number of samples.

3. RESULTS AND DISCUSSION

Using the methodology as described in the previous section, the prediction performance of NN models compiled with different hyperparameter sets were systematically investigated. Table 3 displays two selective examples to illustrate such hyperparameter sets. The trend and results of NN models obtained

Table 3. Illustrative hyperparameter sets.

	hyperparameters	set name	
		p2@c22	p10@c25
Tunned	depth of the NN	2	2
	width of the NN	750 (layer 1) 100 (layer 2)	750 (layer 1) 100 (layer 2)
	dropout rate	0.6 (layer 1) 0.6 (layer 2)	0.6 (layer 1) 0.6 (layer 2)
	learning rate	0.001	0.001
	activation function	relu (layer 1) relu (layer 2)	relu (layer 1) relu (layer 2)
	batch size	8	8
	Epochs	6	6
Fixed	loss function	mean_squared_error	
	validation_split	0.3	

with other hyperparameter sets were also investigated but not presented in this manuscript, as they were largely similar to that of the two illustrative NN models in the limit the accumulated data for retraining the *.h5 models became sufficiently large.

As mentioned in the methodology section, a model with a fixed set of hyperparameters was retrained in an incremental manner. The NN after each retraining was successively saved and named 0.h5, 1.h5, 2.h5, ..., etc. The prediction accuracy of two selective NN models compiled with different hyperparameter sets, namely p2@c22 and p10@c25, using one of the stress test configurations labelled test.479600_c_p10@c27, is displayed in the last columns of Figs. 4 and 5. This test data contains a total of 479600 configurations, where each configuration was essentially a set of numerical values in the format as given in Table 1 and was generated randomly based on the prescription as explained in Section 2.1. The results produced by three selected stages of retraining, namely, 1.h5, 21.h5, and 40.h5, were displayed in Fig. 4 (for p2@c22) and 2 (for p10@c25). Be reminded that in the present study, there were 19 test configurations such as test.479600_c_p10@c27 in the data_test/ directory. test.479600_c_p10@c27 was arbitrarily chosen merely for the purpose as an illustration. The pattern and trend of the output by the NN model using other test configurations were largely the same. Monitoring the rows in Fig. 4 (for p2@c22) in the sequence 1.h5 $\rightarrow$ 21.h5 $\rightarrow$ 40.h5 provides a glimpse of the trend of the evolution of a NN in its ability to predict the extinction cross section as the accumulated data used for retraining each *.h5 models increases. The accumulative training data for the initially trained model 1.h5 and the subsequently retrained models 21.h5 and 40.h5 were 366540, 4030760, and 7505169 respectively for p2@c22. Fig. 5 displays similar information to that in Fig. 4 but for a different model compiled with hyperparameter set p10@c25. The performances for these two models were largely similar in that the R^2 value in both NN models saturates to ~ 1 and a relatively small mse when they were retrained with a sufficiently amount of training data.

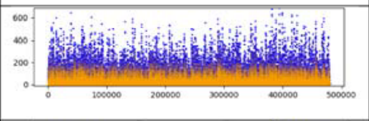
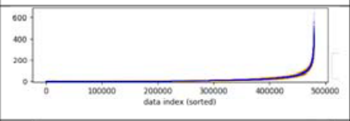
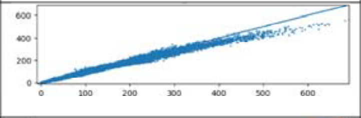
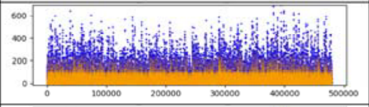
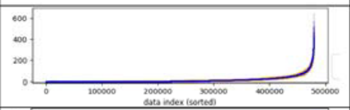
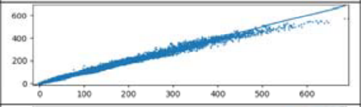
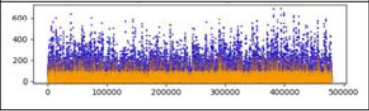
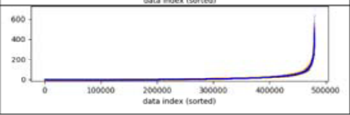
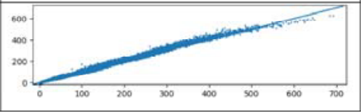
Hyperparameter set	*.h model	Number of accumulated data o train the model	Metric R-squared	mse	Scatter plot of σ_{ext}(prediction) and σ_{ext}(ground truth) as a function of common configuration label. Blue (orange) dots represent ground truth (prediction).	Sorted version of the scatter plot, where σ_{ext}(prediction), σ_{ext}(ground truth) sorted in ascending order are plotted as a monotonically increasing curve.	σ_{ext}(prediction) vs. σ_{ext}(ground truth).
p10@c25	1.h5	369780	0.99	15.29			
	21.h5	4018251	0.99	9.69			
	40.h	7762475	0.99	7.34			

Figure 5. The test results of the NN model compiled with hyperparameter set p10@25 for three selected stages for retraining, namely 1.h5, 21.h5 and 40.h, when they were subjected to a common stress test using configuration file test.479600_c_p10@c27.

Figure 6 shows the trend of the accuracy performance as a function of the amount of training data for NN models with hyperparameter set p2@c22 and p10@c25. The accuracy performance is quantified in terms of R^2 and mse. In Fig. 6 the initial part of the R^2 vs. amount of accumulative training data curves does not display gradual or incremental climb in accuracy as more data were added in the retraining process. Instead, the curves show that the *.h5 models have achieved a near-to-ideal accuracy earlier on after being trained with only the first few 10^5 of configuration samples. Be reminded that each config.npy file contains a large amount ($\sim 10^5$) of configuration samples. Each discrete point in the scatter plot in these figures represents a NN retrained with about $\sim 10^5$ or more data samples than the previous one. The 'abrupt' jump to a near-to-idea accuracy is likely due to the following possibility: With just one or two chunks of $\sim 10^5$ training sets, the NN has already been able to learn the encoded information of the GMM calculator up to a saturation level. It is anticipated that R^2 would display an incremental increase in the initial segment of the curve if each chuck of training data file (i.e., config.npy) contains a much smaller amount of training data than $\sim 10^5$. Despite that the process of incremental increase in the learning accuracy is not explicitly demonstrated in Fig. 6 due to the excessively large chuck of training configuration samples in each config.npy file, what ultimately counts is that the resultant NN *.h5 files are able to make accurate prediction. This has clearly been demonstratively achieved by the near-to-ideal behaviour in R^2 and mse of these curves.

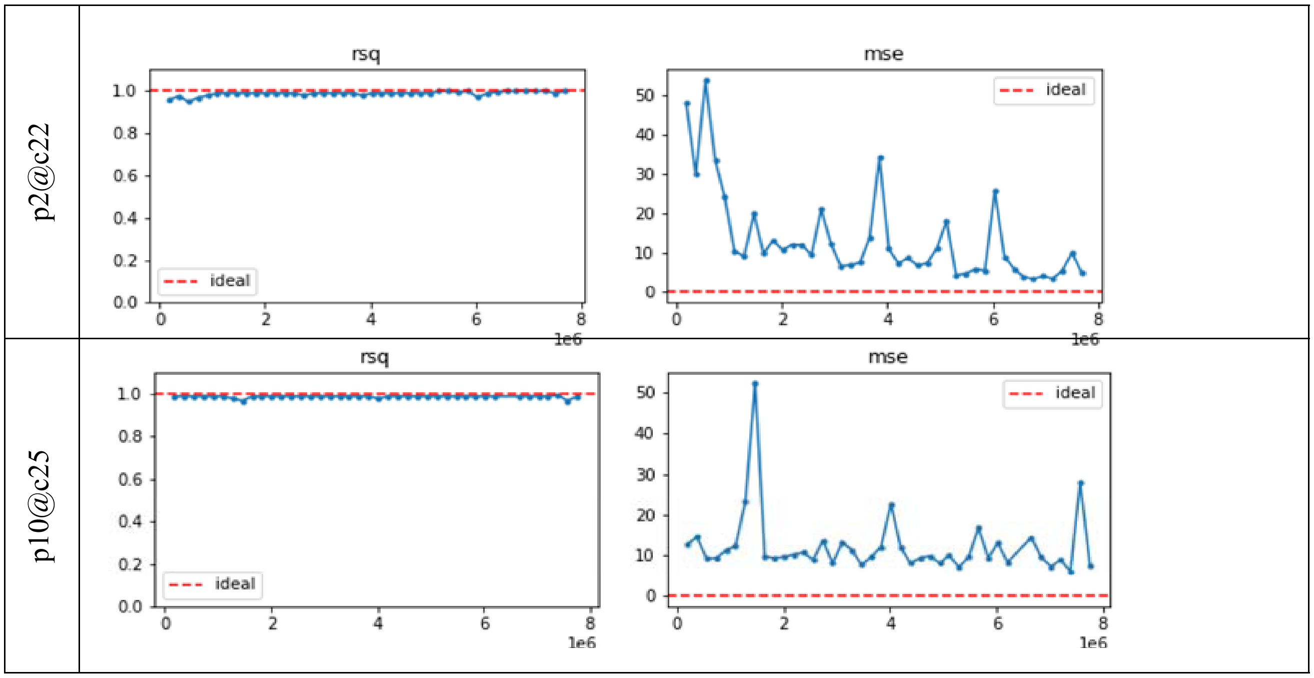


Figure 6. The trend of accuracy performance varies with incremental amount of training data. Each dot in the scatter plot denotes the accuracy performance of a *.h5 model retrained with an accumulated number of training configuration indicated in the horizontal axis.

R^2 was observed to be a less sensitive measure of the prediction accuracy, evidenced by the rather flat scatter plot that runs closely along the 'ideal' (red-dotted) line in Fig. 6. On the other hand, the scatter plots of mse in Fig. 6 for both illustrative hyperparameter sets display erratic peaks at certain retrained *.h5 model. Despite such erratic peaks, the mse fluctuates about a mean value of about ≈ 10.0 (for p10@c25) and ≈ 5.0 (for p2@c22). The mse achievable in other hyperparameter sets (not reported here) were largely similar to these two illustrative results. The smallest possible mse achievable by all NN models with various hyperparameter sets was found to be about ≈ 5.0. From the generic results shown in Fig. 6, it is seen that the accuracy of retrained NN models could achieve an average mse as low as 5.0 and an $R^2 \sim 1.0$, but cannot attain the ideal value of mse = 0. If an NN has zero mse, it means that such a model could 100% accurately predict the value of σ_{ext} for any given arbitrary aggregate configuration with number of spheres less or equal to 14 but larger or equal than 2. However, given the range of parameters in Table 2 used to generate the random configurations, most σ_{ext} values lie below ≈ 450 (this is easily seen from the scatter plots in Figs. 4 and 5), while some will spread beyond 450 and extend up to ≈ 600. With the assumption that σ_{ext} was approximately distributed uniformly in the interval ≈ 0 to ≈ 450, an averaged mse of ≈ 5 in the σ_{ext} prediction by the retrained NN model appears feasibly small and acceptable for practical applications. In principle any *.h5 model producing an acceptably small mse in the stress tests could be used as a NN representation of the GMM calculator. Since the NN model compiled with hyperparameter set p2@c22 appears more accurate than p10@c25 by and large, the last *.h5 model, namely, 40.h5 with the hyperparameter set p2@c22 was selected and used as the NN representation of the GMM calculator in future application.

The computational cost of both NNGMM and GMM has been estimated in a server equipped with an AMD Ryzen 5 3600X 6-core processor and a Nvidia GeForce GTX 1060 GPU. Note that when a trained NNGMM file is invoked to make a σ_{ext} prediction, it does so by using the "model.predict(X_test)" function in Tensorflow, which in turn makes use of the available GPU to execute the computation. On the other hand, the GMM code can only be executed in a CPU processor. It took about 5.3 hours to produce the output (i.e., σ_{ext}) of 150 k configurations (28.3 k configurations per hour) in parallel mode using 11 threads with the Ryzen 5 processor. On the other hand, the trained NNGMM spent roughly 10 minutes to produce the predicted value of σ_{ext} of 479.6 k configurations ($\sim 2,900$ k configurations per hour) using one CPU thread plus the GPU. The enhancement of about 100 times in the computational speed-up of the NNGMM is partly due to the advantage of GPU-assisted hardware over the CPU-only GMM. However, the major contributor of the impressive enhancement is because NNGMM 'calculates' a predicted output by inserting the input configuration into the *.h5 file, which almost instantly returns an output by the "model.predict(X_test)" function packaged into Tensorflow. In contrast, the GMM has to go through a full-fledge computational routine to calculate the output via the built-in algorithmic solutions to the Maxwell's equations. The speed-up of computational efficiency in NNGMM is an obvious value-added advantage over the GMM code.

4. CONCLUSION

The current work has provided a proof of concept methodology to represent the Generalized Multi-particle Mie-solution calculator in the form of a neural network representation using data generated within the parameter space as specified in Table 2. The results presented in the present work illustrate the robust possibility to realise a NN representation of the GMM calculator when it is trained with a sufficiently large amount of random data. The amount of accumulated training data required to achieve an acceptably good mse, and R-squared *.h5 model is not a fundamental concern, because random configuration data for training purposes could be generated at a low computational cost. The NN representations of the GMM calculator obtained in the present work achieves a near-to-ideal accuracy earlier on after being trained with only the first few 10^5 of configuration samples. The hyperparameters of a working NN model that could effectively represent the GMM calculator required some fine-tuning. From a pragmatic point of view, any set of hyperparameters that could produce a NN *.h5 file that passes the stress test to a satisfactory level (e.g., $R^2 \approx 1.0$, mse ≈ 5.0) suffices, irrespective of their exact values. Table 3 exemplifies the possible choices of hyperparameters that could do the job. The exact choice of the hyperparameters was by no means unique nor perfectly optimum. Neither were they difficult to tune. The NN model, compiled with hyperparameter set p2@c22, 40.h5, was shown to predict with a good accuracy of $R^2 \approx 1.0$ and mse ≈ 5.0. It could be deployed as an effective NN representation of the GMM calculator to calculate predicted values of σ_{ext} for any arbitrary aggregate of spheres with number $N \leq 10$. It is reasonable to expect that a sufficiently accurate NN representation of the GMM calculator for a much larger parameter space (e.g., for number of spheres in the aggregate much larger than $N_{\text{max}} = 10$, or the case where the reflective indices are different for each sphere in the aggregate) than that listed in Table 2 could be realized by following the methodology described in this manuscript in a straight-forward manner.

ACKNOWLEDGMENT

Y. L. Thong acknowledges the support of a scholarship from the Ministry of Education, Government of Malaysia.

REFERENCES

1. Mie, G., "Beiträge zur Optik trüber Medien, speziell kolloidaler Metallösungen," *Annalen der Physik.*, Vol. 330, No. 3, 377–445, 1908.
2. Stremme, M. J., "Fast Mie calculations with a radial basis function neural network," M.Sc. Thesis, University of Bergen, Norway, 2019.
3. Berdnik, V. V., K. Gilev, A. Shvalov, V. Maltsev, and V. A. Loiko, "Characterization of spherical particles using high-order neural networks and scanning flow cytometry," *Journal of Quantitative Spectroscopy and Radiative Transfer*, Vol. 102, No. 1, 62–72, 2006, doi: https://doi.org/10.1016/j.jqsrt.2006.03.002.
4. Gugliotta, L. M., G. S. Stegmayer, L. A. Clementi, V. D. G. Gonzalez, R. J. Minari, J. R. Leiza, and J. R. Vega, "A neural network model for estimating the particle size distribution of dilute latex from multiangle dynamic light scattering measurements," *Particle & Particle Systems Characterization*, Vol. 26, Nos. 1–2, 41–52, 2009, doi: https://doi.org/10.1002/ppsc.200800010.
5. Atsushi, Y., S. Tomonobu, A. Y. Saber, F. Toshihisa, S. Hideomi, and C. Kim, "Application of neural network to 24-hour-ahead generating power forecasting for PV system," *2008 IEEE Power and Energy Society General Meeting — Conversion and Delivery of Electrical Energy in the 21st Century*, Jul. 20–24, 2008.
6. Draine, B. T. and P. J. Flatau, "Discrete-dipole approximation for scattering calculations," *Journal of the Optical Society of America A*, Vol. 11, No. 4, 1491–1499, 1994, doi: 10.1364/JOSAA.11.001491.
7. Moon, C. Y., "Particle sensing in gas turbine inlets using optical measurements and machine learning," Doctoral Dissertations, Blacksburg, Virginia, 2020, http://hdl.handle.net/10919/101969.
8. Stegmayer, G. S., O. A. Chiotti, L. M. Gugliotta, and J. R. Vega, "Particle size distribution from combined light scattering measurements. A neural network approach for solving the inverse problem," *2006 IEEE International Conference on Computational Intelligence for Measurement Systems and Applications*, Jul. 12–14, 2006.
9. Guerrero, J. A., F. M. Santoyo, D. Moreno, M. Funes-Gallanzi, and S. Fernandez-Orozco, "Particle positioning from CCD images: Experiments and comparison with the generalized Lorenz-Mie theory," *Measurement Science and Technology*, Vol. 11, No. 5, 568–575, 2000, doi: 10.1088/0957-0233/11/5/318.
10. Wang, H. and X. Xu, "Determination of spread constant in RBF neural network bygenetic algorithm," *Int. J. Adv. Comput. Technol. (IJACT)*, Vol. 5, No. 9, 719–726, 2013.
11. Akashi, N., M. Toma, and K. Kajikawa, "Design of metamaterials using neural networks," *SPIE*, Vol. 11194, 2019.
12. Mamun, M. M. and D. Müller, "Retrieval of intensive aerosol microphysical parameters from multiwavelength Raman/HSRL lidar: Feasibility study with artificial neural networks," *Atmos. Meas. Tech. Discuss.*, 1–46, 2016, doi: 10.5194/amt-2016-7.
13. Xu, Y.-L., "Electromagnetic scattering by an aggregate of spheres: Errata," *Applied Optics*, Vol. 37, No. 27, 6494–6494, 1998, doi: 10.1364/AO.37.006494.
14. Xu, Y.-L., "Electromagnetic scattering by an aggregate of spheres: Far field," *Applied Optics*, Vol. 36, No. 36, 9496–9508, 1997, doi: 10.1364/AO.36.009496.
15. Xu, Y.-L., "Electromagnetic scattering by an aggregate of spheres: Asymmetry parameter," *Physics Letters A*, Vol. 249, No. 1, 30–36, 1998, doi: https://doi.org/10.1016/S0375-9601(98)00708-7.
16. Xu, Y.-L. and R. T. Wang, "Electromagnetic scattering by an aggregate of spheres: Theoretical and experimental study of the amplitude scattering matrix," *Physical Review E*, Vol. 58, No. 3, 3931–3948, 1998, doi: 10.1103/PhysRevE.58.3931.
17. Xu, Y.-L., B. Å. S. Gustafson, F. Giovane, J. Blum, and S. Tehranian, "Calculation of the heat-source function in photophoresis of aggregated spheres," *Physical Review E*, Vol. 60, No. 2, 2347–2365, 1999, doi: 10.1103/PhysRevE.60.2347.

18. Xu, Y.-L. and R. T. Wang, "Electromagnetic scattering by an aggregate of spheres: Theoretical and experimental study of the amplitude scattering matrix," *Physical Review E*, Vol. 58, No. 3, 3931–3948, 1998, doi: 10.1103/PhysRevE.58.3931.
19. Guerrero, J. A., F. M. Santoyo, D. Moreno, M. Funes-Gallanzi, and S. Fernandez-Orozco, "Particle positioning from CCD images: Experiments and comparison with the generalized Lorenz-Mie theory," *Measurement Science and Technology*, Vol. 11, No. 5, 568–575, 2000, doi: 10.1088/0957-0233/11/5/318.
20. Lock, J. A. and G. Gouesbet, "Generalized Lorenz-Mie theory and applications," *Journal of Quantitative Spectroscopy and Radiative Transfer*, Vol. 110, No. 11, 800–807, Jul. 2009.
21. Ren, K. F., G. Gréhan, and G. Gouesbet, "Prediction of reverse radiation pressure by generalized Lorenz-Mie theory," *Applied Optics*, Vol. 35, No. 15, 2702–2710, 1996, doi: 10.1364/AO.35.002702.
22. Pellegrini, G., G. Mattei, V. Bello, and P. Mazzoldi, "Interacting metal nanoparticles: Optical properties from nanoparticle dimers to core-satellite systems," *Materials Science and Engineering: C*, Vol. 27, 1347–1350, 2007, doi: 10.1016/j.msec.2006.07.025.
23. Xu, F., K. Ren, G. Gouesbet, G. Gréhan, and X. Cai, "Generalized Lorenz-Mie theory for an arbitrarily oriented, located, and shaped beam scattered by a homogeneous spheroid," *Journal of the Optical Society of America A*, Vol. 24, No. 1, 119–131, 2007.
24. Lock, J. A. and G. Gouesbet, "Generalized Lorenz-Mie theory and applications," *Journal of Quantitative Spectroscopy and Radiative Transfer*, Vol. 110, No. 11, 800–807, 2009, doi: https://doi.org/10.1016/j.jqsrt.2008.11.013.
25. Jia, X., J. Shen, and H. Yu, "Calculation of generalized Lorenz-Mie theory based on the localized beam models," *Journal of Quantitative Spectroscopy and Radiative Transfer*, Vol. 195, 44–54, 2017, doi: https://doi.org/10.1016/j.jqsrt.2016.10.021.
26. Xu, Y.-L. and B. S. Gustafson, "A generalized multiparticle Mie-solution: Further experimental verification," *Journal of Quantitative Spectroscopy and Radiative Transfer*, Vol. 70, No. 4, 395–419, 2001, doi: https://doi.org/10.1016/S0022-4073(01)00019-X.
27. Xu, Y.-L., "Efficient evaluation of vector translation coefficients in multiparticle light-scattering theories," *Journal of Computational Physics*, Vol. 139, 137–165, 1998.
28. Xu, Y.-L., "Fortran source codes for calculation of radiative scattering by aggregated particles in both fixed and random orientations for homogeneous spheres, core-mantle and for ensembles of variously shaped (meaning rotationally symmetric) particles," *GMM — Generalized Multiparticle Mie-Solution [Fortran Source Code]*, Oct. 2, 2013, https://scattport.org/index.php/light-scattering-software/multiple-particle-scattering/135-gmm-generalized-multiparticle-mie-solution.
29. https://scattport.org/files/xu/codes.htm, assessed on 15 Dec. 2021.
30. LeCun, Y., Y. Bengio, and G. Hinton, "Deep learning," *Nature*, Vol. 521, No. 7553, 436–444, 2015.
31. Chen, X., Z. Wei, M. Li, and P. Rocca, "A review of deep learning approaches for inverse scattering problems (invited review)," *Progress In Electromagnetics Research*, Vol. 167, 67–81, 2020.
32. Thomée, V., "From finite differences to finite elements: A short history of numerical analysis of partial differential equations," *Numerical Analysis: Historical Developments in the 20th Century*, 361–414, Elsevier, 2001.
33. Yee, K. S. and J. S. Chen, "The finite-difference time-domain (FDTD) and the finite-volume time-domain (FVTD) methods in solving Maxwell's equations," *IEEE Transactions on Antennas and Propagation*, Vol. 45, No. 3, 354–363, 1997.
34. Jin, J. M., *The Finite Element Method in Electromagnetics*, John Wiley & Sons, 2015.
35. Banerjee, P. K., P. K. Banerjee, and R. Butterfield, *Boundary Element Methods in Engineering Science*, McGraw-Hill, UK, 1981.
36. Harrington, R. F., *Field Computation by Moment Methods*, Wiley-IEEE Press, 1993.
37. Chew, W. C., E. Michielssen, J. M. Song, and J. M. Jin, *Fast and Efficient Algorithms in Computational Electromagnetics*, Artech House, Inc., 2001.
38. Ren, Q., Y. Wang, Y. Li, and S. Qi, *Sophisticated Electromagnetic forward Scattering Solver via Deep Learning*, Springer Singapore Pte. Limited, 2021.

39. Jia, R., X. Zhang, F. Cui, G. Chen, H. Li, H. Peng, and S. Pei, "Machine-learning-based computationally efficient particle size distribution retrieval from bulk optical properties," *Applied Optics*, Vol. 59, No. 24, 7284–7291, 2020.
40. Giannakis, I., 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*, Vol. 57, No. 7, 4417–4426, 2019.
41. Tang, W., T. Shan, X. Dang, M. Li, F. Yang, S. Xu, and J. Wu, "Study on a Poisson's equation solver based on deep learning technique," *2017 IEEE Electrical Design of Advanced Packaging and Systems Symposium (EDAPS)*, 1–3, IEEE, Dec. 2017.
42. Wu, H., Y. Zhang, W. Fu, C. Zhang, and S. Niu, "A novel pre-processing method for neural network-based magnetic field approximation," *IEEE Transactions on Magnetics*, Vol. 57, No. 10, 1–9, 2021.
43. Ma, Z., K. Xu, R. Song, C. F. Wang, and X. Chen, "Learning-based fast electromagnetic scattering solver through generative adversarial network," *IEEE Transactions on Antennas and Propagation*, Vol. 69, No. 4, 2194–2208, 2020.
44. Abadi, M., A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous distributed systems," [Application software], 2015, https://www.tensorflow.org/.
45. Abadi, M., P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, and X. Zheng, "TensorFlow: A system for large-scale machine learning," *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, Savannah, GA, USA, 2016.