Supplementary Information (SI)

nanoNET: Machine Learning Platform for Predicting Nanoparticles Distribution in a Polymer Matrix

Kumar Ayush, Abhishek Seth and Tarak K Patra*

*Corresponding Author: E-mail: tpatra@iitm.ac.in

Department of Chemical Engineering, Center for Atomistic Modeling and Materials Design and Center for Carbon Capture Utilization and Storage, Indian Institute of Technology Madras, Chennai TN 600036, India

1. Regression model for directly mapping a composition vector to the corresponding RDF

a) Random Forest (RF)

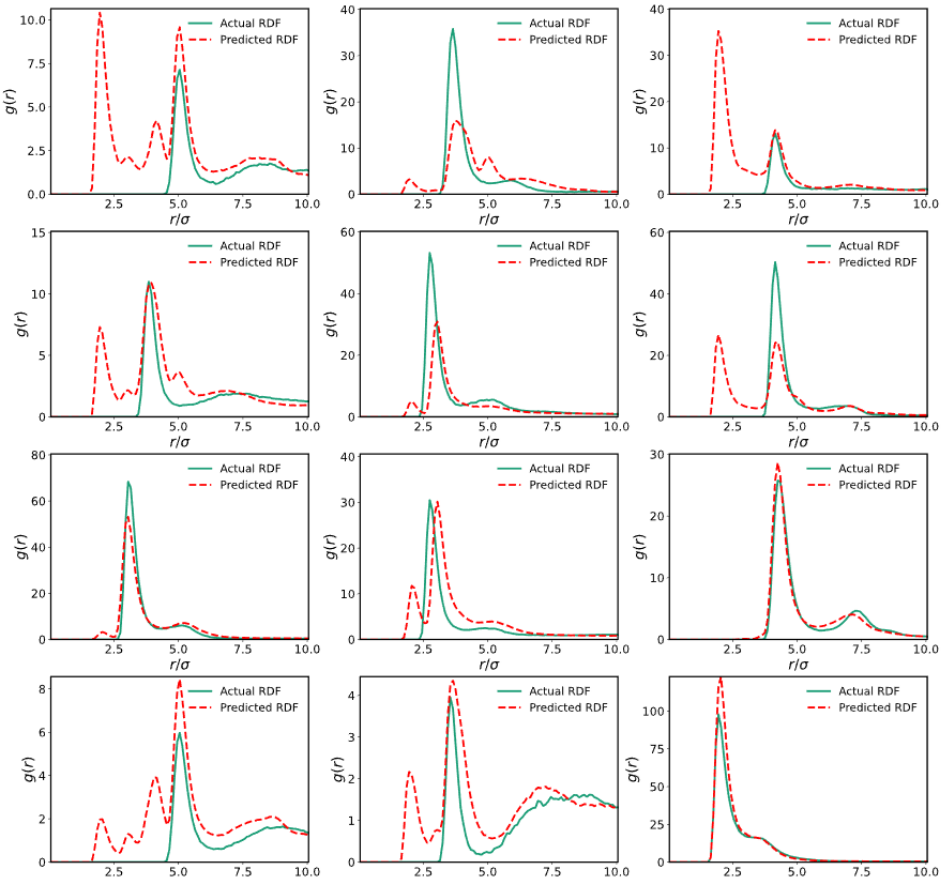

Figure S1: Comparison of actual RDF and predicted RDF for 12 different compositions in case of a RF model that directedly correlates composition vector to the RDF function. The Composition parameters are the same as tabulated in Table S1.

We use the random forest (RF) algorithm to develop a regression model between the 5-dimensional composition vector and 100-dimensional RDF. The RF regressor model employs several important hyperparameters, including *max_depth*, *min_samples_leaf*, and *min_samples_split*. We adjust these parameters to control the complexity of the decision trees in the RF and to prevent the overfitting to the training data. The *max_depth* specifies the maximum depth of each decision tree in the forest. Setting a maximum depth helps to prevent the tree from becoming too complex and overfitting to the training data. We have selected the *max_depth*= 4 in this model, indicating that each decision tree will have a maximum depth of four levels. The *min_samples_leaf* specifies the minimum number of samples required to form a leaf node in the decision tree. Setting a higher *min_samples_leaf* can prevent the tree from overfitting to noisy or irrelevant data points. In this model, the *min_samples_leaf* is set to 5, indicating that each leaf node in the decision tree must contain at least five samples. Finally, the *min_samples_split* specifies the minimum number of samples required to split an internal node in the decision tree. Setting a higher *min_samples_split* can prevent the tree from overfitting to small or insignificant subsets of the data. We have set *min_samples_split= 3*, indicating that a node must have at least 3 samples to be split into two child nodes. Overall, these hyperparameters help to control the complexity of the decision trees and prevent overfitting, leading to a more accurate RF regressor model. The choice of this set of parameters is made based on many initial trials in order to achieve the optimal performance. The performance of the RF model is shown in Figure S1. It suggests a significant mismatch between an actual RDF and RF-predicted RDF.

b) Extreme Gradient Boosting (XG-Boost)

We use XG-Boost regressor, a powerful machine learning model, to build a regression model between the 5-dimensional compositional input vector and the 100-dimensional RDF. In this model, the *learning_rate*, *max_depth*, and *n_estimators* are the hyperparameters. The *learning_rate, which is defined as* the step size of the gradient descent optimization process, is set to be 0.1. The *max_depth* hyperparameter is set to be 5, which controls the maximum depth of each decision tree. We set the *n_estimators* to be 51, which is the number of decision trees in the model. Finally, we use mean squared error as the objective function of the model, which is minimized during the training. The performance of the XG-Boost model is shown in Figure S2.

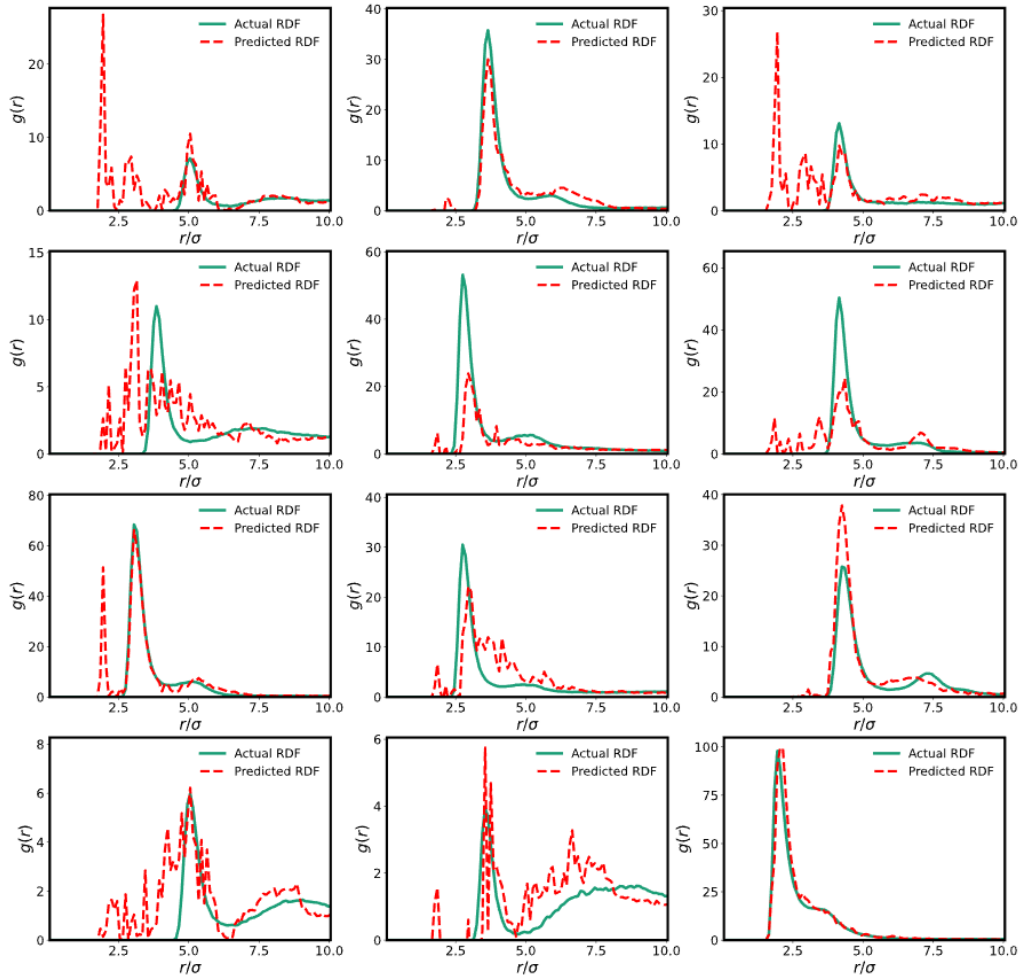

Figure S2: Comparison of actual RDF and predicted RDF for 12 different compositions in case of an XG Boost model that directedly correlates composition vector to the RDF function. The Composition parameters are the same as tabulated in Table S1.

c) Deep Neural Network (DNN)

We implement a sequential neural network model for the regression analysis within the Keras opensource package. The objective is to create a correlation between a 5-dimensional input compositional vector and 100-dimensional output RDF vector. The model architecture consists of three hidden layers each with the rectified linear unit (ReLU) activation function. The first hidden layer has 200 neurons, the second and third layers each have 100 neurons. The output layer has 100 neurons with the linear activation function appropriate for the regression task. We use the Adam optimizer with a learning rate of 0.0001 with the mean squared error as the loss function. We train the model for 1000 epochs during which weights between connecting neurons are optimized. We chose this set of parameters based on several trials to optimize the model performance. The performance of the best DNN model is shown in Figure S3.

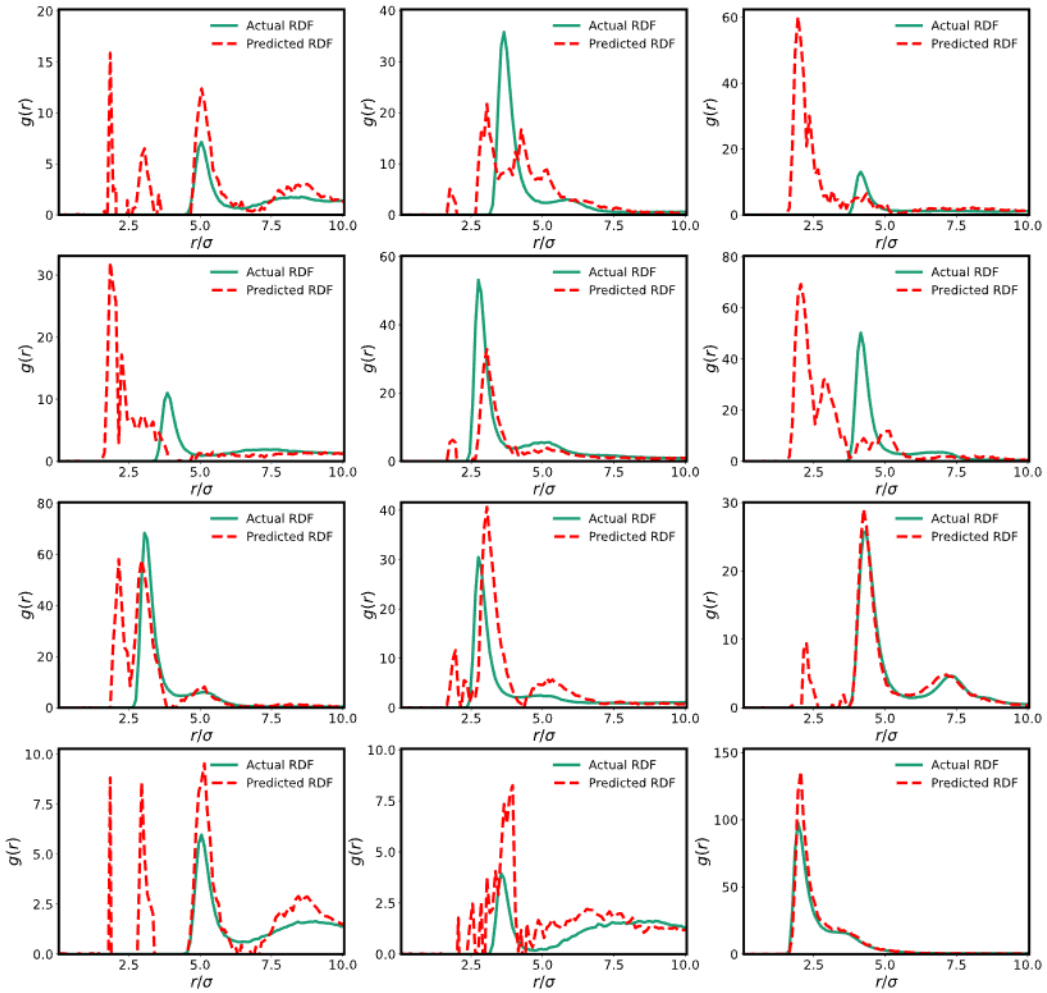

Figure S3: Comparison of actual RDF and predicted RDF for 12 different compositions in case of a DNN model that directedly correlates composition vector to the RDF function. The Composition parameters are the same as tabulated in Table S1.

d) **Support Vector Regression (SVR)**

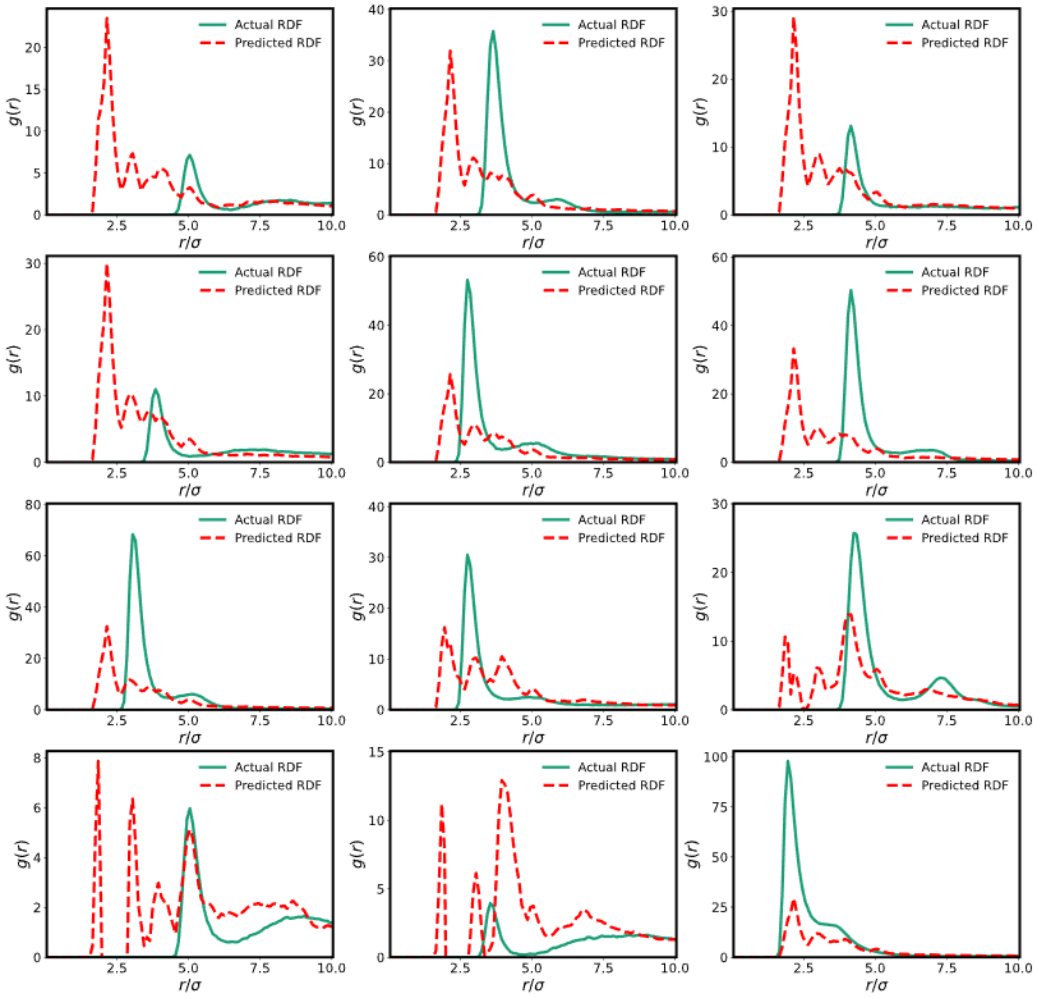

Figure S4: Comparison of actual RDF and predicted RDF for 12 different compositions in case of a SVM model that directedly correlates composition vector to the RDF function. The Composition parameters are same as tabulated in Table S1.

The support vector regression (SVR) model is a powerful machine learning technique used for predicting continuous output values. We implement the SVR model using the scikit-learn library. We use three hyperparameters to fine-tune the model performance: *C*, gamma, and *kernel*. The *C* parameter controls the regularization strength of the model and adjusts the trade-off between achieving a large margin and minimizing the prediction error. The *gamma* parameter determines the width of the radial basis function (RBF) kernel and controls the smoothness of the decision boundary. The *kernel* parameter specifies the type of kernel function used for the SVM model, and we have used the polynomial kernel in this study. We have set the value of the *C* parameter to *3*. The *MultiOutputRegressor* module has been used to perform multi-output regression, which helps us predict multiple continuous variables simultaneously. We chose hyperparameters based on previous research and empirical experiments, and their values have been selected through trial and error to achieve optimal performance on the given dataset.

2. Dimensionality reduction of RDF

As discussed in the main text, there are various methods available for feature extraction and dimensionality reduction. Here, we test the performance of principal component analysis (PCA) and deep neural network (DNN) autoencoder for reducing the dimension of nanoparticles RDF in a polymer matrix. Upon dimensionality reduction, we build an RF model to correlate the composition vector of a PNC and the corresponding latent space representation of an RDF. The performance of these two approaches is discussed below.

a) PCA-based dimensionality reduction

The PCA model is used to reduce the dimensionality of RDF data by extracting a new set of uncorrelated variables called principal components. The main hyperparameter of PCA is the number of principal components to retain, which directly affects the amount of variance in the data that is preserved. We build the PCA model with 16 principal components. The 16-dimensional latent space gives the optimal performance. The reconstructed RDF image shows

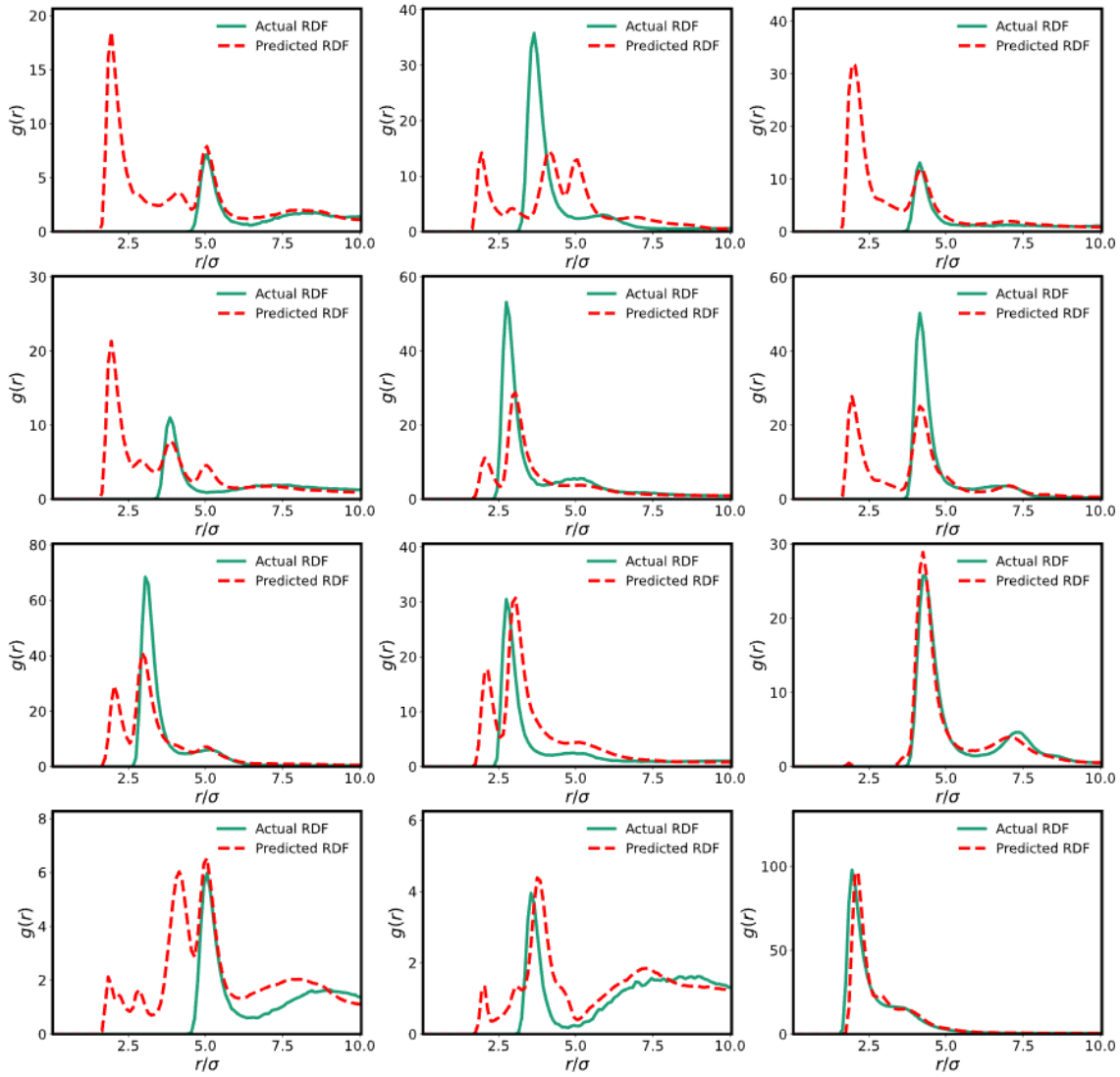

Figure S5: PCA is used to map RDFs to a latent space of dimension 12, and then RF is used to correlate composition matrix and latent space representation. The performance of this ML pipeline is shown here for 12 unknown compositions. The Composition parameters are same as tabulated in Table S1.

a close resemblance to the original RDF image, indicating a good match. We then build a RF model to find the correlation between a 5-dimensional compositional input vector and a 16-dimensional reduced vector of an RDF image. We primarily focusing on two key parameters during the regression: the number of decision trees and the size of the feature subsets used when splitting a node. We perform a detailed analysis of the model's performance with varying numbers of trees, and our results indicate that using 100 decision trees results in the lowest prediction error. The prediction of RF is then mapped to the actual RDF using inverse transform method. We utilize the scikit-learn library for this model building. The performance of such a regression model is shown in Figure S5.

b) Deep neural network (DNN)-based dimensionality reduction:

We use a DNN for RDF dimensionality reduction. The architecture of the DNN autoencoder creates a latent space of dimension 10 for a RDF vector of dimension 100. The decoder part of the encoder consists of four hidden layers. The number of nodes in the successive dense layers of the DNN is 1000, 500, 100, and 10, respectively. The topology of the decoder is a mirror

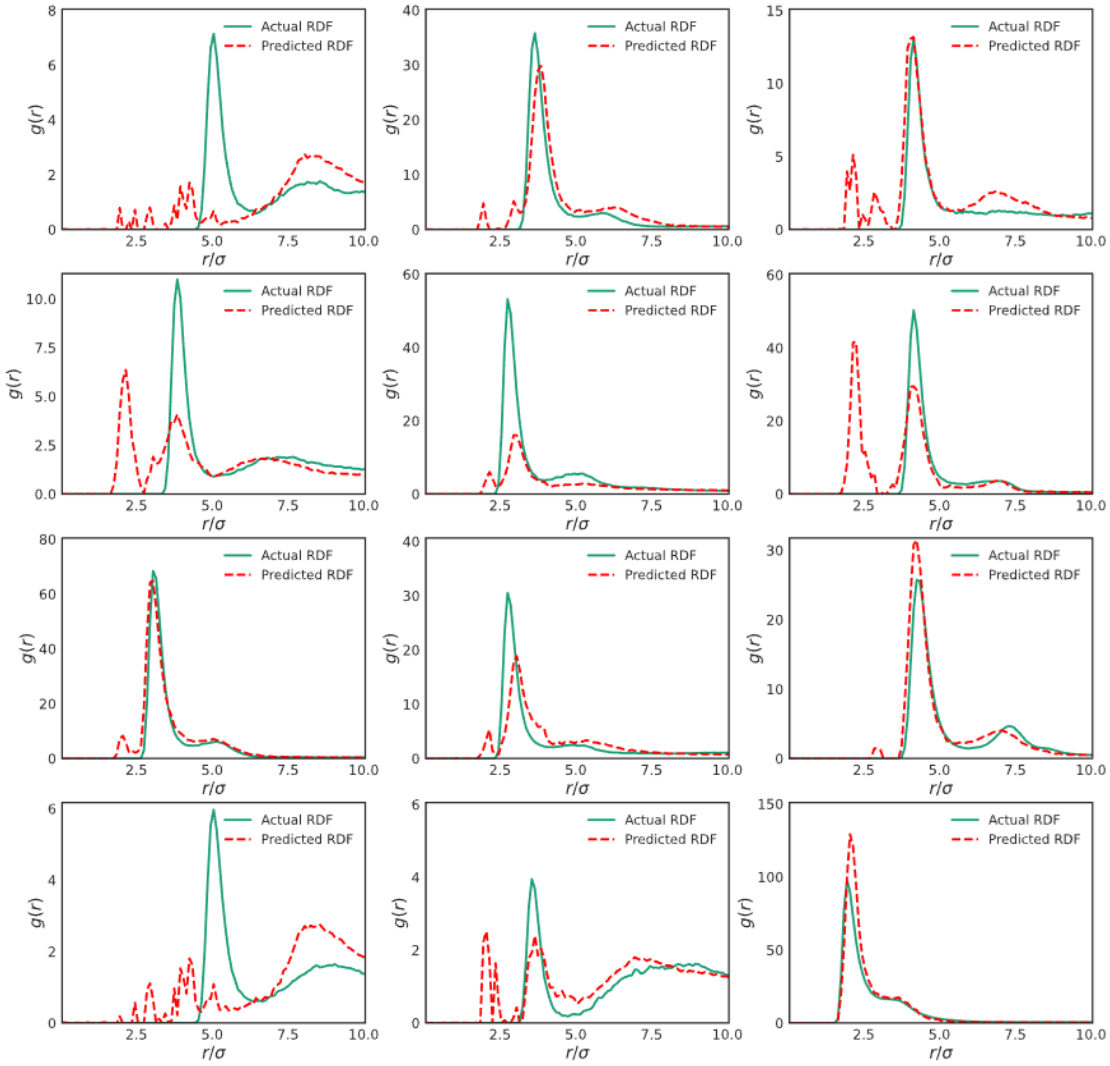

Figure S 6: DNN autoencoder is used to map RDFs to a latent space of dimension 10, and then RF is used to correlate the composition matrix and latent space representation of a PNC. The performance of this ML pipeline is shown here for 12 unknown compositions. The Composition parameters are the same as tabulated in Table S1.

image of the encoder. Therefore, the number of nodes in the successive hidden layers of the decoder is 10, 100, 500, and 1000 respectively. The input and output of the autoencoder is of dimension 100, which is same the RDF vector. The ReLU activation function is used to activates all the neurons of all the hidden layers. We use the Adam optimizer with a learning rate of 0.0003. The mean squared error (MSE), which measures the difference between the predicted and actual values of the output, is considered as the loss function of the model. We train the model for 1000 epochs with shuffling of the training data after each epoch. The choice of the hyperparameters is based on several trials in order to optimize the performance of the model. Subsequently, we build a RF regression model between a 5-dimensional compositional input vector and a 10-dimensional latent vector. The two main parameters that we focus on are the number of decision trees and the size of the subsets of features selected randomly when splitting a node. We conduct a thorough analysis of the model's performance with different numbers of trees and have found that utilizing 400 decision trees leads to the lowest prediction error. We build the RF regression model using the scikit-learn library. The performance of the DNN-based RF model is shown in Figure S6.

3. Dimension of the Latent Space of the nanoNET pipeline:

We carry out a systematic analysis to identify the appropriate dimension of the latent space of RDFs, which can be correlated to the composition vector via a random forest regressor. As shown in Figure S7, the latent space of dimension 8 appears to be the most efficient.

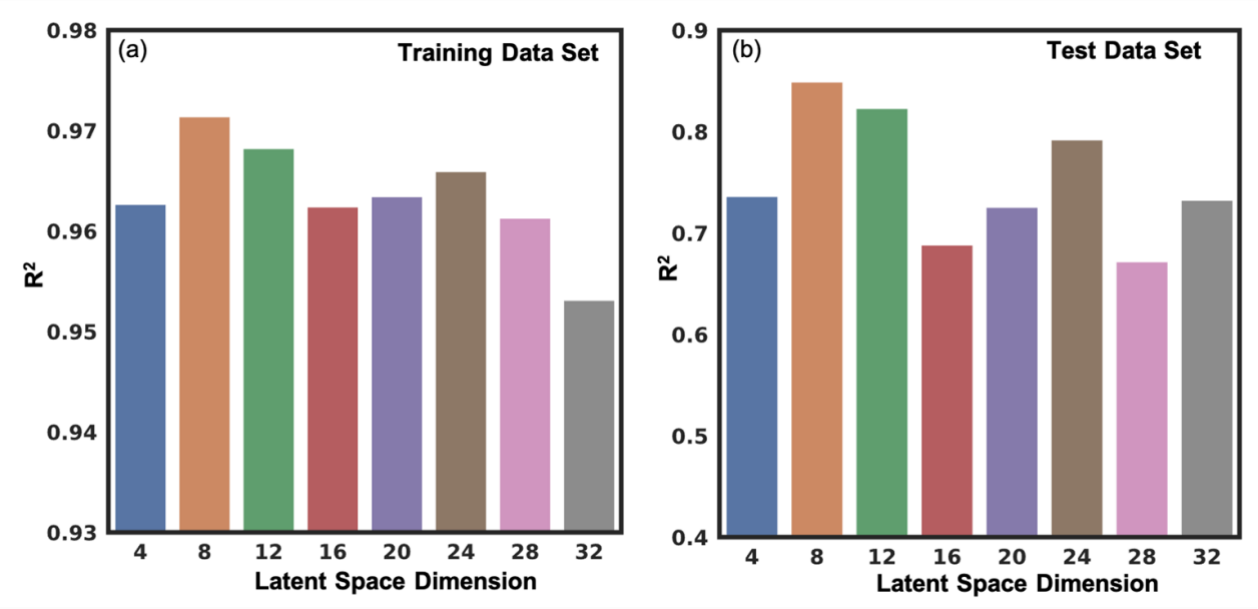

Figure S7: Performance of random forest regressor. R2 score for training and test data for models that build a correlation between the composition vector and latent space vector for different dimensions of the latent space.

4. Composition Parameters of the PNCs (Training and Test Data)

Here, ϵ_{P-NP} is the interaction parameter between the polymer and nanoparticle, ϵ_{NP-NP} is the interaction parameter between a pair of nanoparticles, **D** is diameter of a nanoparticle, $\boldsymbol{N_{NP}}$ is the number of nanoparticles present in the polymer matrix, and $\boldsymbol{N_{polymer}}$ is the number of monomers in a polymer chain.

No.	ϵ_{P-NP}	ϵ_{NP-NP}	D	N_{NP}	$N_{polymer}$
1 (Fig. 7a)	0.6	1.5	5	8	25
2 (Fig. 7b)	0.2	0.2	4	8	25
3 (Fig. 7c)	0.6	2	4	8	30
4 (Fig. 7d)	0.6	0.6	4	8	25
5 (Fig. 7e)	0.6	0.2	3	8	35
6 (Fig. 7f)	0.2	2	4	8	30
7 (Fig. 7g)	0.2	1	3	8	25
8 (Fig. 7h)	0.6	0.2	3	12	35
9 (Fig. 7i)	0.05	0.1	5	12	30
10 (Fig. 7j)	0.6	1.5	5	12	25
11 (Fig. 7k)	0.8	0.2	4	12	40
12 (Fig. 7l)	0.4	0.2	2	12	30

Table 1: Compositional parameters of data (***test data***) on which performance of the model has been evaluated. RDF predictions for 12 PNCs are shown in Figure 7 of the main article. These parameters are also used in Figures S1-S6.

No.	ϵ_{P-NP}	ϵ_{NP-NP}	D	N_{NP}	$N_{polymer}$
1	0.4	1.5	2	8	35
2	0.8	1	3	12	40
3	0.4	1	4	8	30
4	0.4	0.2	2	8	30
5	0.8	1.5	3	8	35
6	0.2	0.2	4	12	25
7	1.5	0.1	5	12	30
8	0.6	1	2	12	35
9	0.2	0.4	3	12	30
10	0.8	0.4	3	8	25
11	0.2	0.8	4	12	40
12	0.4	0.8	2	12	25
13	0.6	0.4	2	12	40
14	0.4	0.6	3	8	40
15	1.5	0.1	5	8	30
16	0.8	1.5	3	12	35
17	0.4	0.4	4	12	35
18	0.2	2	4	12	30
19	0.8	1	3	8	40
20	1	0.1	5	12	30
21	0.1	0.1	5	8	30
22	0.8	0.4	3	12	25

23	0.6	0.8	3	8	30
24	0.8	0.6	2	12	30
25	0.2	0.8	4	8	40
26	0.8	0.6	2	8	30
27	1	2.5	4	8	30
28	0.4	0.8	2	8	25
29	1	0.8	2	12	40
30	0.4	0.4	4	8	35
31	0.8	2.5	2	8	40
32	0.8	2.5	2	12	40
33	0.1	0.1	5	12	30
34	0.4	0.6	3	12	40
35	1	1.5	5	12	25
36	0.8	0.2	4	8	40
37	0.05	0.1	5	8	30
38	0.2	0.6	2	12	35
39	1	0.1	5	8	30
40	0.4	1	4	12	30
41	1	1	4	8	25
42	0.2	1	3	12	25
43	0.4	1.5	2	12	35
44	0.8	0.8	4	12	35
45	0.6	0.4	2	8	40
46	0.4	2	2	8	40
47	0.6	2	4	12	30
48	1	0.2	2	12	25
49	1	1	4	12	25
50	0.2	1.5	5	12	25
51	1	1.5	5	8	25
52	0.2	1.5	5	8	25
53	0.2	0.4	3	8	30
54	0.6	0.8	3	12	30
55	1	0.4	4	8	30
56	0.2	0.6	2	8	35
57	1	0.8	2	8	40
58	1	0.2	2	8	25
59	1	0.4	4	12	30
60	0.4	2	2	12	40
61	0.6	1	2	8	35
62	0.6	0.6	4	12	25
63	1	2.5	4	12	30
64	1	0.6	3	12	35
65	1	0.6	3	8	35
66	0.8	0.8	4	8	35
67	0.2	1	3	8	30
68	0.4	0.8	2	12	40

*Table 2: Compositional parameters of data (**training data**) on which the model has been trained.*

5. Comparison of models' performance

For each model framework, we have conducted three cross-validations. The performance of the best model is shown in Figure 7 of the articles. Here we compare the prediction error of all the models. We note that the MSE of the nanoNET is negligible in comparison to other models.

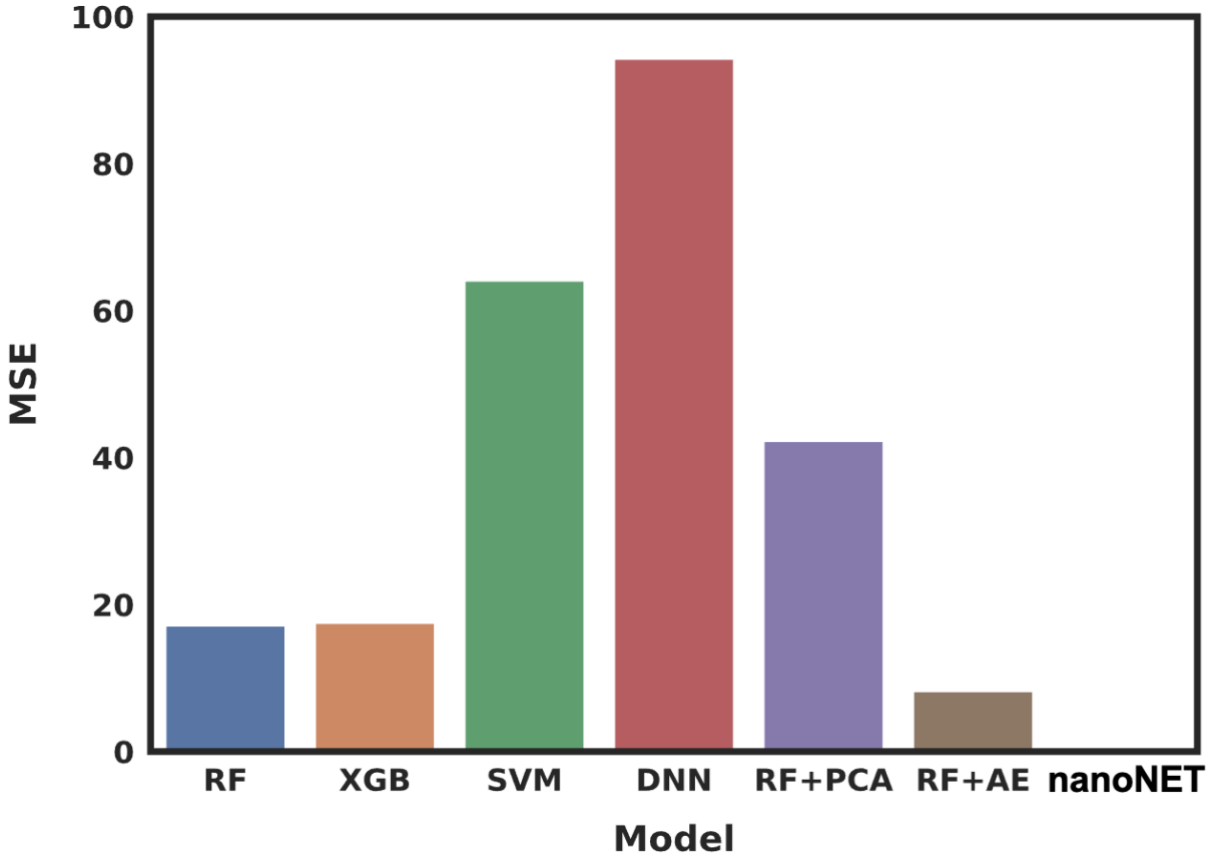

Figure S8: Comparison of models' performance in predicting RDF of nanoparticles in a polymer matrix based on the composition parameters. The MSE of the nanoNET is 0.06.