Electronic Supplementary Material (ESI) for Digital Discovery.

Supporting Information for

Neural networks for a quick access to a digital twin of scanning physical properties measurements

Kensei Terashima,[1,*] Pedro Baptista de Castro,[1,2] Miren Garbiñe Esparza Echevarria,[1,2] Ryo Matsumoto,[1] Takafumi D Yamamoto,[1] Akiko T Saito,[1] Hiroyuki Takeya,[1] and Yoshihiko Takano[1,2]

[1] *National Institute for Materials Science, 1-2-1 Sengen, Tsukuba, Ibaraki, 305-0047 Japan*

[2] *University of Tsukuba, 1-1-1 Tennodai, Tsukuba, Ibaraki, 305-8577, Japan*

TRAPEZOIDAL INTEGRATION AND BILINEAR INTERPORATION

Trapezoidal integration is a numerical method to often used to obtain the an approximation of the area of interest below the curve of a function $f(x)$ and between the a given set of desired points a and b, assuming: $\int_a^b f(x)dx \sim \frac{(b-a)}{2}(f(a)+f(b))$, that corresponds to a summation of linearly-interpolated values of $f(x)$ between starting point a and end point b. Sometimes, for some datasets this linear interpolation method can fail. Figure SM1 shows a typical example of such a failure in linear interpolation. Suppose there are data for $y(x_1, x_2)$ taken in $x_2 = a$ and b, where peak position of y migrates by changing x_2 value, as in Figure SM1(a). If one performs bilinear interpolation as in Figure SM1(b), it does not succeed to reproduce what is desired (as in Figure SM1(a)) but it produces an artificial dip in the obtained interpolated $y(x_1^i, x_2^i)$ .In Section 3.1 of the manuscript, x_1, x_2, y correspond to temperature, magnetic field, and $\left(\frac{\partial M}{\partial T}\right)$, respectively, namely $y(x_1, x_2) = \frac{\partial M(T,H)}{\partial T}$. This causes extrinsic oscillation in $|\Delta S_M|$ derived by integrating $\left(\frac{\partial M}{\partial T}\right)$ along H direction.

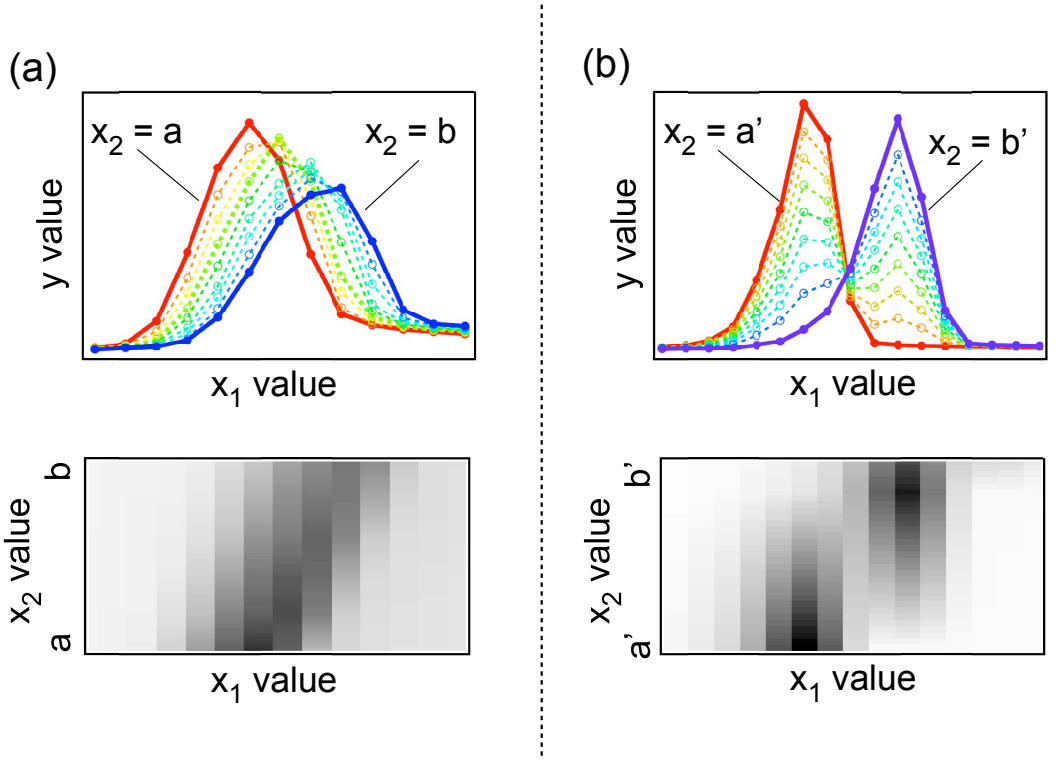

FIG. SM1. Schematic graph showing how (bi)linear interpolation fails to grab characteristic change. Top graphs show spectral lineshapes and bottom graphs are corresponding intensity plots. (a) What is expected (b) What (bi)linear interpolation gives.

We also note here that the situation is also valid for isothermal cuts for $M(T, H)$. In case of isothermal cuts, $|\Delta S_M|$ can be deduced by $\Delta S_M(T_i, H) = \Sigma_i \frac{M_{i+1}-M_i}{T_{i+1}-T_i} \Delta H_i$ (in case of using forward differential), which corresponds to the gray shaded area as depicted in Figure SM2(a). As can be seen in Figure SM2(a) in case of coarse step of H, linear interpolation along H-direction (corresponds to dotted line in the Figure) creates an extrinsic oscillation of area between two subsequent measured temperatures. Figure SM2(b) shows the simulated $M(T, H)$ by neural networks model built by Figure SM2(a) train data, where such extrinsic oscillation of area between measured temperatures is suppressed.

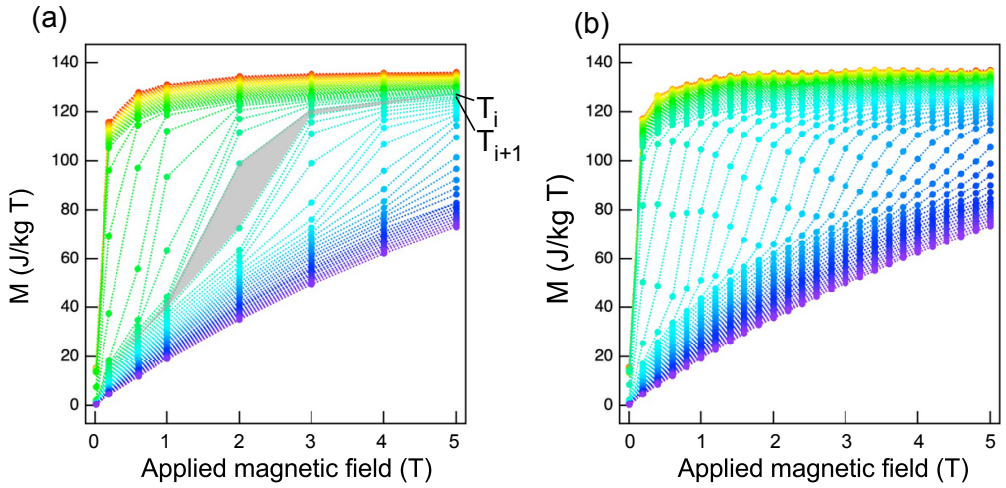

FIG. SM2. Schematic graph showing how linear interpolation along H affects estimated $|\Delta S_M(T_i, H)|$ value, in case of isothermal measurement. (a) Coarse magnetic field step (train data) case. Difference in color stands for different measurement temperatures. Gray area corresponds to estimated $|\Delta S_M(T_i, H)|$. (b) Fine magnetic field step case, simulated by neural networks model.

COMPARISON OF APPROXIMATION METHODS AND RESULTANT ESTIMATED PHYSICAL PROPERTY

Here we compare the results obtained from the same train data for (bi)linear, cubic spline, radial basis function (RBF) and neural networks proposed in this work. Linear and cubic spline interpolation have been performed using scipy package (v1.7.3, https://scipy.org). For RBF, we used a package provided at https://github.com/treverhines/RBF, where func-

tional forms, length scale(eps), and smoothing factor(sigma) are hyperparameters. We used Bayesian optimization package Optuna (https://github.com/optuna/optuna) to obtain the best possible combination of hyperparameters with 5-fold cross validation of train data. The searched range for each hyperparameter was: [thin-plate or gaussian] for functional form, [1e-5, 1e-1] for length scale, and [1e-2, 1e+1] for smoothing factor. As a result of optimization, obtained hyperparameter sets are, functional form: gaussian, length scale = 2.95, and smoothing factor = 0.021.

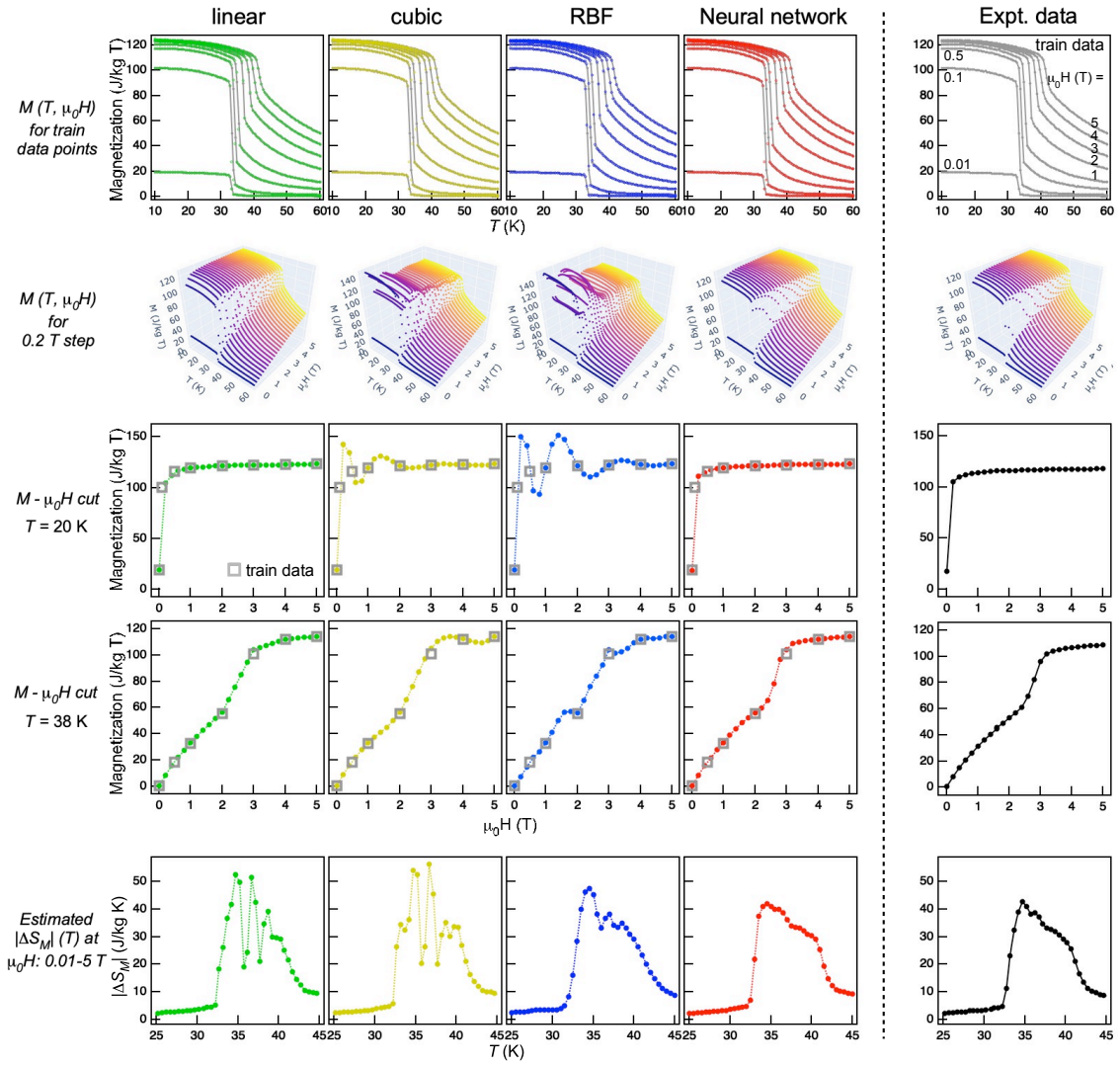

FIG. SM3. Comparison of approximated results among (bi)linear, cubic spline, RBF, and neural networks in the main text. 1^{st} row: predictions for train data points by each model. 2^{nd} row: predictions for finer step of $\mu_0 H$ including unseen points by models. 3^{rd} and 4^{th} rows: $M - \mu_0 H$ cuts of data in 2^{nd} row at fixed $T(K)$. 5^{th} row: estimated $|\Delta S_M|$ by data shown in 2^{nd} row.

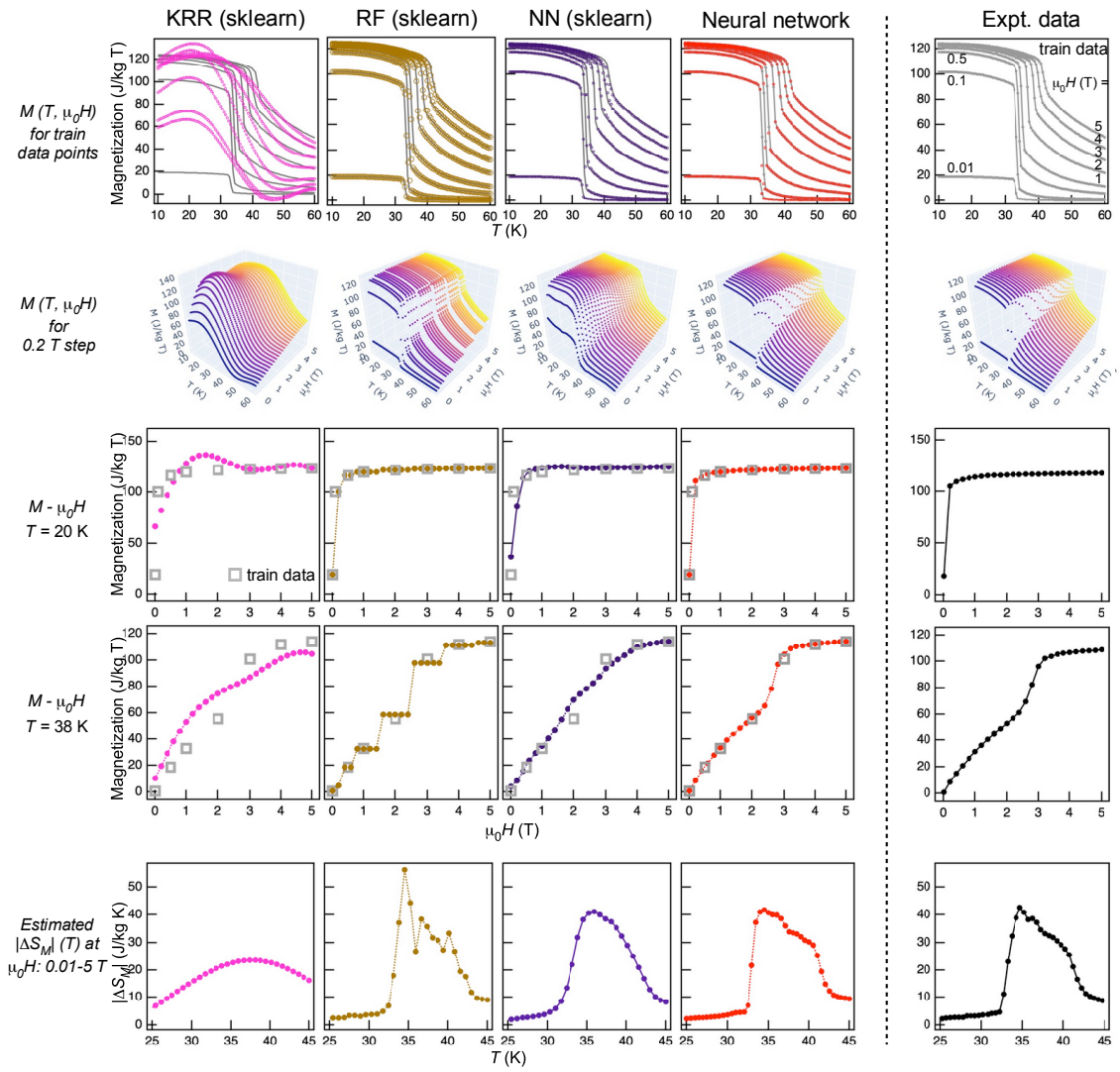

FIG. SM4. Comparison of approximated results between several regressors avalable in Scikit-learn (kernel ridge with RBF kernel, random forest, and neural networks) and neural networks (Tensorflow) in the main text, and experimental verifications for the dataset in Section 3.1 of the main text. The neural networks model in Scikit-learn was constructed using the hyperparameters as the same as possible to those for the ones in the main manuscript Section 3.1. 1^{st} row: predictions for the conditions in the training data points by each model. 2^{nd} row: predictions for finer step of $\mu_0 H$ including unseen points by models. 3^{rd} and 4^{th} rows: $M - \mu_0 H$ cuts of data in 2^{nd} row at fixed $T(K)$. 5^{th} row: estimated $|\Delta S_M|$ by data shown in 2^{nd} row.

In the figure, 1st row shows the predicted values (colored markers) for features in the train data points, compared with the target values of train data (gray markers). All approximation methods trace very well the train data. However, as shown in the 2nd row of the figure,

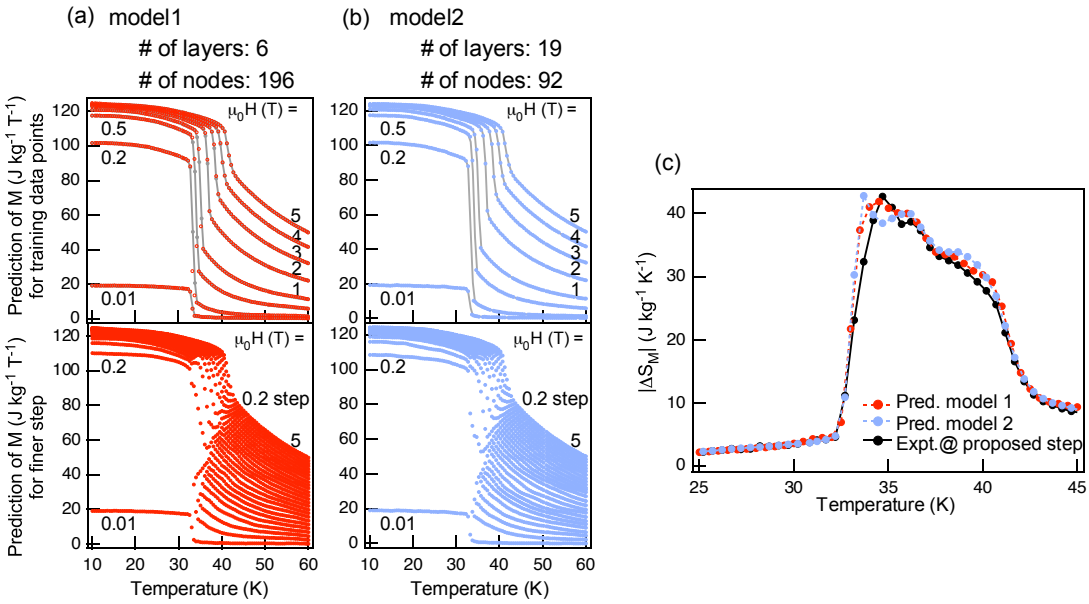

FIG. SM5. (a) and (b) Top panel: Predicted magnetizations for the conditions of training dataset by (a) the model in the main text (Section 3.1) and (b) the model with deeper layers. Bottom panel: The same for finer external field step (0.2 T). (c) $|\Delta S_M|$ by models and experimental data taken on the same step.

predictions for feature points unseen by the models differ each other. Especially, cubic spline and RBF create additional beating behavior, that are clearly visible as well in the 3rd and 4th rows of $M - \mu_0 H$ cuts at fixed T. As a result, none of linear, cubic spline, RBF approximation do not predict precisely the target physical property $|\Delta S_M|$ at 5th row in case of such sparse train dataset, while neural networks proposed in this work predict values well corresponds to those in the experiment taken in the fine step of $\mu_0 H$.

Here we also compare with several regressors available in Scikit-learn, namely kernel ridge with RBF kernel, random forest, and neural networks. Here the models are trained with default hyperparameters, except for the neural networks where the hyperparameters as the same as possible (number of nodes, number of layers, learning rate and batch size) to those in the main text (Section 3.1, where neural networks are built using Tensorflow) were used. Among them, kernel ridge could not learn well the relationship between features and targets, while random forest could not capture the smooth second-axis dependence of target value. The neural network in Scikit-learn has captured an overall trend despite there being a lack of steep structure and additional artificial fluctuations. These behavior might

improve by performing optimization of hyperparameters for each, but it is out of scope of current manuscript.

We also compare with a model with deeper number of layers. In our code, the search area for number of layers during the hyperparameter optimization is 2-10 by default. The number is confined so that even non-state-of-the-art PC can quickly construct a model, but the search area can be changed optionally. In Fig. SM5, we show a comparison of the predicted result between (a) the model in the manuscript and (b) the model with deeper layers. Fig. SM5(c) is a resultant target property $|\Delta S_M|$. At least for the data in Section 3.1, the model found in the default search area seems to be good enough to be used as a simulator.

WORKING OPERATING SYSTEM OF THE CODE AND INSTALLATION GUIDE

A tutorial for how to get ready and perform the neural networks learning and simulation shown in this paper is available at https://doi.org/10.5281/zenodo.7523510 and https://www.github.com/kensei-te/mat_interp.

There are two ways provided to use the code. One is via Jupyter Notebook, which runs both on users own PCs and Google Colab. Another way is additional installation of Streamlit and mySQL for GUI-based use. For the latter, it has been tested only on Linux and MacOS (Intel) systems. In more details, the following OSes were tested by the authors: Ubuntu LTS 20.04 and 18.04, CentOS 7, and MacOS 11 and 12 (both Intel).

EXAMPLE OF DURATION TO PERFORM NEURAL NETWORKS LEARNING

Table SM1 shows the total duration it took for each computer to finish 30 trials of learning. The used computers are the following: (i) Intel i9-9880H 2.3 GHz, (ii) Xeon Silver 4116 2.1 GHz, and (iii) Intel i7 10700K 3.8 GHz. The number of parallel workers are set to be 5 for (i) and (iii), while it was 15 for (ii). Since the result would be influenced by random seeds, we tried 5 times for each, and the averaged values are shown in the table. Standard deviation of 5 times of 30 trials are also presented as numbers in parenthesis. The number of train data points are 808, 2737, 767, 29977 for Section 3.1, 3.2, 3.3, 3.4, respectively.

Data	Intel i9-9880H 2.3 GHz 8 cores, MacOS11(Intel)	Xeon Silver 4116 2.1 GHz 12x2 cores, CentOS7.6	Intel i7 10700K 3.8 GHz 8 cores, Ubuntu20.04
Section3.1	3m 30s (40s)	3m 53s (29s)	2m 36s (45s)
Section3.2	4m 30s (46s)	3m 17s (26s)	2m 31s (12s)
Section3.3	2m 53s (25s)	3m 8s (30s)	2m 5s (35s)
Section3.4	13m 36s (2m 38s)	6m 47s (47s)	11m 14s (1m 46s)

TABLE SM1. Comparison of ellapsed time required for PCs to finish learning through 30 trials of Baysian optimization for each dataset shown in the main text. The results are averaged values of 5 times of trials. Numbers in parenthesis stand for standard deviation.

STABILITY OF LEARNED AND SIMULATED RESULTS

Here we show how much the learning curves and the resultant simulated value would differ among each 30 trials. Two examples are shown, one for the data presented in Section 3.1 in the main text (Figure SM6 top), and another for the data presented in Section 3.3 (Figure SM6 bottom), where the latter is more influenced by noise. Here each trial is allowed to keep running until it reachs a maximum epoch number of 500, unless it is stopped by either pruning or earlystopping. After 30 trials of Bayesian optimization (initial 10 trials are random), both the R^2 score for training and simulated values show a convergence to some extent.

SPECTRAL LINESHAPES OF ARPES IMAGE PLOT (SECTION3.4)

Figure SM7 shows spectral lineshape of ARPES image plot in Section 3.4 of main text, where gray lines are train data and colored lines are simulated ones by neural networks model. The spectra consist a number of peak structures and model can capture the structures as well as sensitivity of the machine such as relatively high intensity region around the edge.

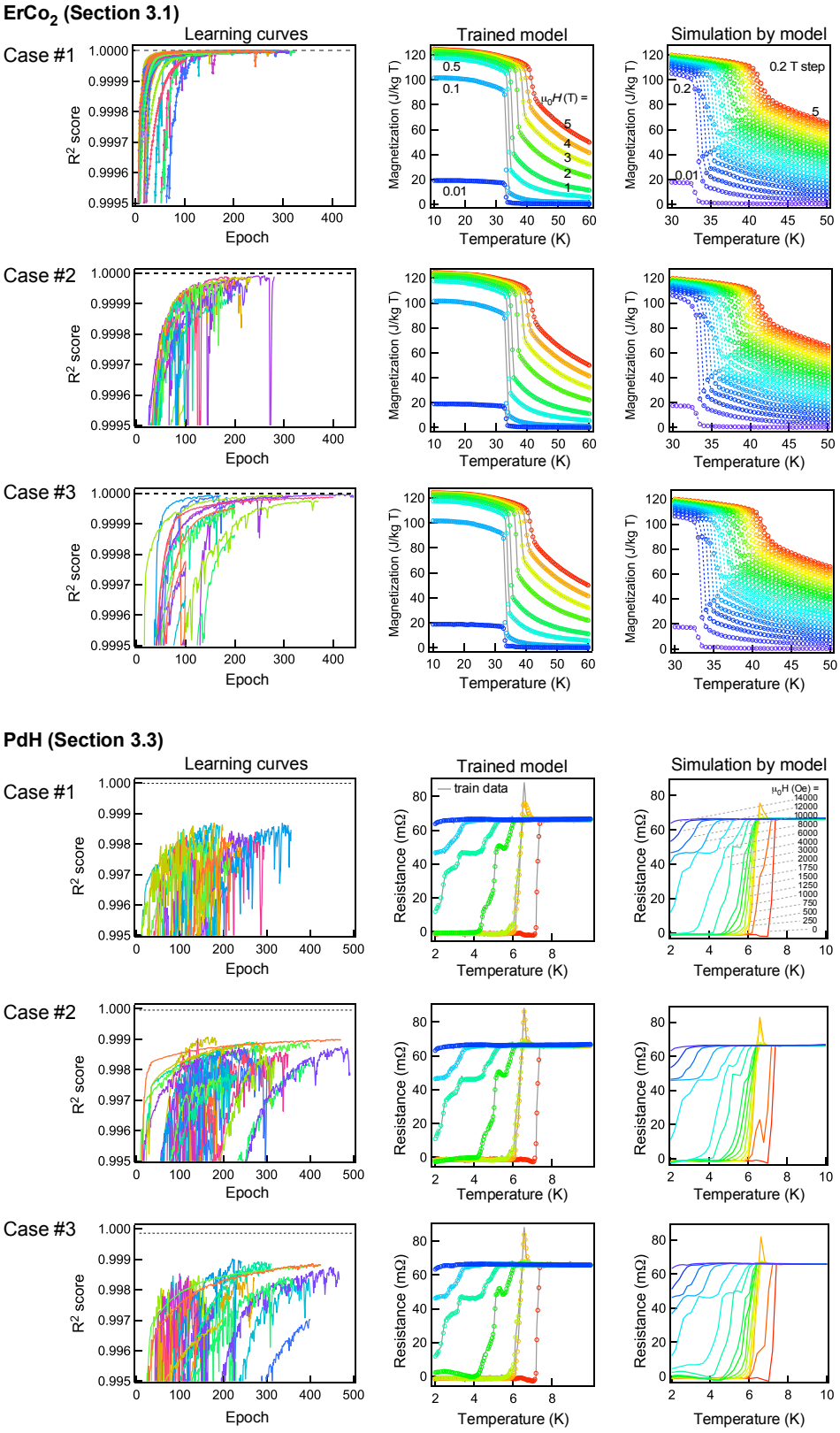

FIG. SM6. Results of learning and simulations in individual cases of 30 trials for $ErCo_2$ (top) and PdH (bottom) data. For each material, case#1 corresponds to the data in the main text. Left: Learning curves. Colors represent different trials. Center: Simulation by learned models that acquired the best R^2 score in each case. Gray lines show train data. Right: Simulation by models.

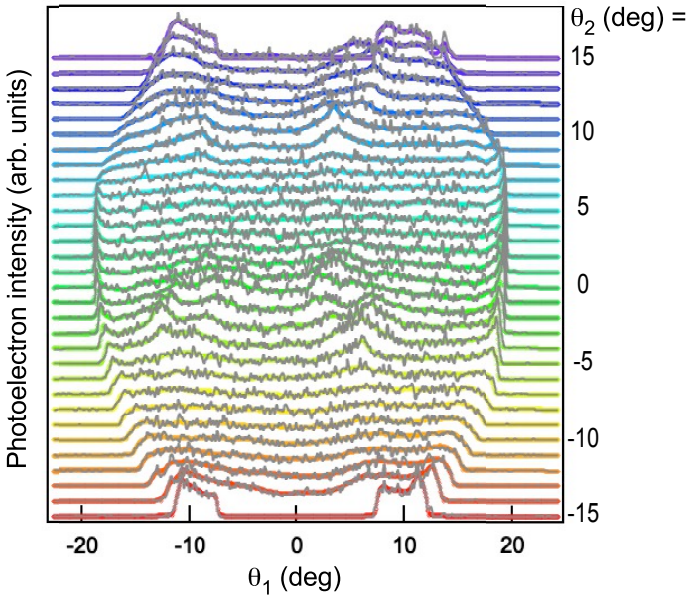

FIG. SM7. Spectral lineshapes of ARPES image plot, shown as Figure 5(a) and (b) in Section 3.4 of main text. Gray lines correspond to train data, while colored lines correspond to simulated ones by learned model.

* TERASHIMA.Kensei@nims.go.jp