Supplementary Materials

1. Prompt Engineering

1.1. Prompt Engineering for Generating MATLAB Scripts that Train, Test and Evaluate Neural Networks

You are an expert at coding. I have 540 pieces of data and I want to train |ANN name| model using these data. Now, generate the MATLAB script for this process. The script should have the functions below:

1. Data preprocessing and dataset dividing. The script should load the data from a .csv file. Column 1 to Column 27 are the input features, and Column 28 to Column 34 are the output features. After loading the data, the script should normalize the dataset to the range of 0 to 1. Then the script should divide the training set and the testing set in a ratio of 8:2 randomly using a certain random seed.
2. Model training, testing and evaluation. After dividing the training set and testing set, the script should train the model. The script should contain a set of network architecture and parameters that you recommend. Then the script should test the model on the testing set, and evaluate the performance of the model using determination coefficient (R^2), mean absolute error (MAE) and root mean square error (RMSE).
3. Data saving. The script should save three MATLAB data files. The first data file contains the training set and testing set. The second data file contains only one variable, which is the trained model. The third data file contains the actual and predicted output features both on the training set and the testing set, and the evaluation parameters of the model on the training set and the testing set.

Should you have any question or find any aspects of this prompt unclear, please include them in your response.

1.2. Prompt Engineering for Generating MATLAB Scripts that Interpret CNN Using the Grad-CAM Method

You are an expert at coding. Previously, I have trained a convolutional neural network (CNN) and now I want to interpret this model using gradient-weighted class activation mapping (Grad-CAM) method. Now, generate the MATLAB script for this process. The script should have the functions below:

1. Load data. The script should load the training set and the CNN model that are stored in MATLAB data file.
2. Execute Grad-CAM. The script should execute Grad-CAM explanation for all 7 output features. Every entry in the training set should be used as a query point.
3. Output data. The script should output the Grad-CAM results in a CSV file.

Should you have any question or find any aspects of this prompt unclear, please include them in your response.

1.3. Prompt Engineering for Interacting with GPT-4 to Perform Debugging Tasks

Based on the above script, I have the following requests.

|Branch 1| The input feature of my CNN is not an image but a matrix. I have uploaded the input features in an Excel file to you.

[Attached file: Input features.xlsx]

|Branch 2| When executing Line [x], an error occurs:

“[error message]”

Revise the script. Should you have any question or find any aspects of this prompt unclear, please include them in your response.

2. Response from GPT-4

Here is a MATLAB script that meets your requirements. This script includes functions for data loading and preprocessing, model training, evaluation, and saving the necessary data:

MATLAB

```
% Load and Preprocess Data
data = readtable('your_data.csv');
inputs = data{:, 1:27};
outputs = data{:, 28:34};
% Normalize data to the range 0 to 1
inputs = (inputs - min(inputs)) ./ (max(inputs) -
min(inputs));
outputs = (outputs - min(outputs)) ./ (max(outputs) -
min(outputs));
% Divide data into training and testing sets
rng(1); % For reproducibility of random numbers
cv = cvpartition(size(data, 1), 'HoldOut', 0.2);
idxTrain = training(cv);
idxTest = test(cv);
XTrain = inputs(idxTrain, :);
YTrain = outputs(idxTrain, :);
XTest = inputs(idxTest, :);
YTest = outputs(idxTest, :);
% Save the datasets
save('datasets.mat', 'XTrain', 'YTrain', 'XTest',
'YTest');
% Create and train the CNN model
layers = [
    imageInputLayer([1 27 1]) % Adjust the size according
to your actual input dimensions
    convolution2dLayer([1 3], 16, 'Padding','same')
    reluLayer
    fullyConnectedLayer(64)
    reluLayer
    fullyConnectedLayer(7)
    regressionLayer
];
options = trainingOptions('adam', ...
    'MaxEpochs',100, ...
    'MiniBatchSize', 16, ...
    'InitialLearnRate', 1e-3, ...
    'Shuffle', 'every-epoch', ...
    'ValidationData', {XTest, YTest}, ...
    'Plots', 'training-progress');

model = trainNetwork(reshape(XTrain,[],27,1), YTrain,
layers, options);
% Save the trained model
save('trainedModel.mat', 'model');
% Model Testing and Evaluation
YPred = predict(model, reshape(XTest,[],27,1));
% Calculate R2, MAE, RMSE
r2 = 1 - sum((YTest - YPred).^2) ./ sum((YTest -
mean(YTest)).^2);
mae = mean(abs(YTest - YPred), 'all');
rmse = sqrt(mean((YTest - YPred).^2, 'all'));
% Save the results
results.actual = YTest;
results.predicted = YPred;
results.R2 = r2;
results.MAE = mae;
results.RMSE = rmse;
save('results.mat', 'results');
fprintf('R2: %f\n', r2);
fprintf('MAE: %f\n', mae);
fprintf('RMSE: %f\n', rmse);
```

Notes:
1. Data Normalization: Ensure that the normalization factors (min and max values used) are saved if you need to apply the same normalization to new incoming data during deployment.
2. Network Architecture: The architecture is a simple example. Depending on your specific problem, the number, size of layers, and types of layers might need adjustment.
3.Evaluation Metrics: R2 is computed as one minus the ratio of the sum of squares of residuals to the total sum of squares. This is one ap proach among many to calculate R2.
4. Saving Data: The script saves three .mat files as per your request containing the datasets, the model, and the performance results.
Feel free to customize the script further according to your specific needs or to align with additional features of your dataset and model requirements.

Figure S1. GPT-4 response for prompt engineering to assist in generating scripts for training and evaluating neural networks.

Since your input features are matrices (not images) and stored in an Excel file, I'll adjust the script to handle this data format. First, I need to inspect the Excel file to understand its structure and how the data is organized. I'll then modify the script to read the matrix data from this Excel file and update the Grad-CAM execution part accordingly.
Let's start by examining the content of the uploaded Excel file to see how the data is structured.
The Excel file contains data structured as a matrix with 539 rows and 27 columns. Each row represents a set of input features for your CNN.
Now, I'll adjust the MATLAB script to:
1. Load the input features from this Excel file.
2. Use the loaded matrix as input for the Grad-CAM analysis.
3. Ensure that the Grad-CAM visualization and data processing aligns with the non-image nature of your data.
Here's the revised MATLAB script:

MATLAB

```
function interpret_cnn_with_gradcam()
    % Load CNN model
    load('model_data.mat'); % Ensure 'model_data.mat'
contains 'trainedModel'

    % Load input features from Excel file
    inputFeatures = readmatrix('Input features.xlsx');

    % Assuming 'trainedModel' is the CNN model
    % Verify that the model is loaded correctly
    disp("Model loaded successfully.");

    % Prepare the CSV file to store Grad-CAM results
    resultsFileName = 'gradcam_results.csv';
    writematrix(["Index", "Feature", "Grad-CAM Heatmap
Data"], resultsFileName);

    % Execute Grad-CAM for each entry and each output
feature
    numSamples = size(inputFeatures, 1);
    numFeatures = 7; % Update if the number of features
changes
    for i = 1:numSamples
        for j = 1:numFeatures
            % Here we need to reshape the input matrix if
needed
            % Adjust the reshape function according to
your CNN input layer requirement
            reshapedInput = reshape(inputFeatures(i, :),
[1, size(inputFeatures, 2), 1]);
            heatmap = gradcam(trainedModel, reshapedIn-
put, "OutputFeature", j);

            % Process heatmap for output
            heatmapData = extractHeatmapData(heatmap);

            % Append results to CSV
            dataToWrite = [i, j, heatmapData];
            writematrix(dataToWrite, resultsFileName,
'WriteMode', 'append');
        end
    end

    disp("Grad-CAM analysis completed and results
saved.");
end

function heatmapData = extractHeatmapData(heatmap)
    % This function should convert heatmap to a suitable
format for CSV
    % Example: Flatten and convert to string
    heatmapData = num2str(heatmap(:)');
end
```

Notes:
- Data Loading: This script loads the input features directly from an Excel file. Ensure the path and filename match your actual file.
- Input Reshaping: Since your data is matrix-based (not typical image input), it might require reshaping to match the input layer of your CNN. You'll need to adjust the `reshape` parameters accordingly based on your model's requirements.
-Grad-CAM Function: If your CNN input isn't typical image data, you may need to ensure that the Grad-CAM method or library you use supports your data type. This might require custom adjustments or alternative methods for visualization.
This revised script should be more aligned with your data and analysis needs.

Figure S2. Using branch 1 of Category III prompt engineering to modify the script for using Grad-CAM to interpret CNN.

3. Formulas of R^2, MAE and RMSE

The formula for coefficient of determination is as follows:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y_i})^2}{\sum_{i=1}^{n}(y_i - \overline{y})^2} \tag{S1}$$

where n is the number of samples, y_i is the actual value of variable y at i, $\hat{y}_i$ is the predicted value of variable y at i, and $\bar{y}$ is the mean value of y samples.

The formula for mean absolute error is as follows:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}\left|y_i - \overline{y}_i\right| \tag{S2}$$

where n is the number of samples, y_i is the actual value of variable y at i, and $\hat{y}_i$ is the predicted value of variable y at i.

The formula for root mean square error is as follows:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y_i})^2} \tag{S3}$$

where n is the number of samples, y_i is the actual value of variable y at i, and $\hat{y}_i$ is the predicted value of variable y at i.

4. Hyperparameters of Neural Networks

Table S1. Hyperparameters of three neural networks.

Methods	Hyperparameters	Value
BP	epochs	1000
	goal	1×10^{-6}
	lr	0.01
	hiddens	[5,5]
LSTM	solverName	‘adam’
	MiniBatchSize	64
	MaxEpochs	1000
	InitialLearnRate	0.01
	LearnRateDropPeriod	800
	LearnRateDropFactor	0.1
	numHiddenUnits	6
CNN	solverName	‘adam’
	MiniBatchSize	32
	MaxEpochs	200
	InitialLearnRate	0.001
	LearnRateDropPeriod	150
	LearnRateDropFactor	0.1

5. Scatter Plot of Neural Networks

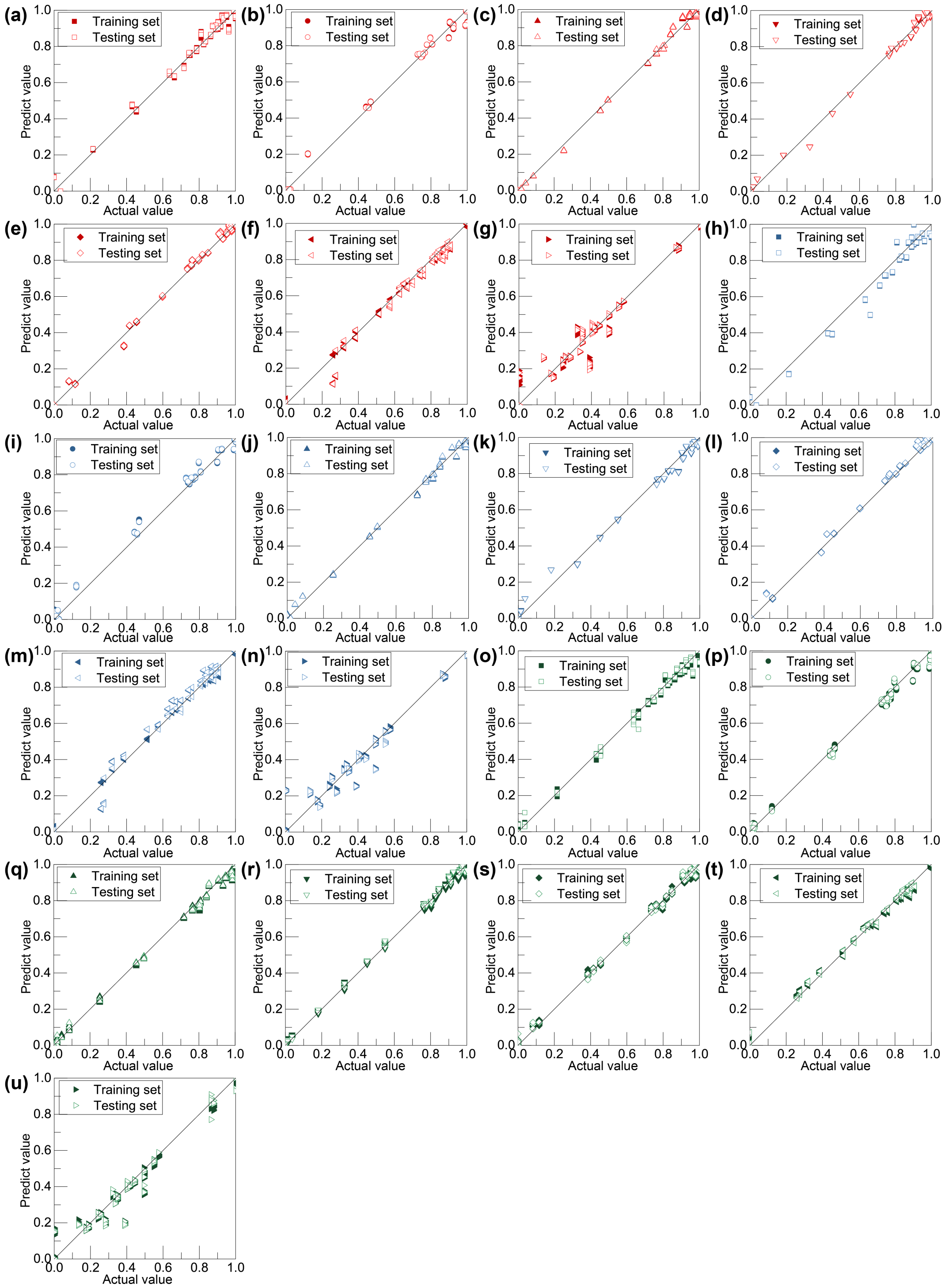

Figure S3. Scatter plot of models: (**a**) BP, ρ_{20}^{TO}; (**b**) BP, $DP_{10\%}^{TO}$; (**c**) BP, $DP_{30\%}^{TO}$; (**d**) BP, $DP_{50\%}^{TO}$; (**e**) BP, $DP_{70\%}^{TO}$; (**f**) BP, $DP_{90\%}^{TO}$; (**g**) BP, $BMCI^{TO}$; (**h**) LSTM, ρ_{20}^{TO}; (**i**) LSTM, $DP_{10\%}^{TO}$; (**j**) LSTM, $DP_{30\%}^{TO}$; (**k**) LSTM, $DP_{50\%}^{TO}$; (**l**) LSTM, $DP_{70\%}^{TO}$; (**m**) LSTM, $DP_{90\%}^{TO}$; (**n**) LSTM, $BMCI^{TO}$; (**o**) CNN, ρ_{20}^{TO}; (**p**) CNN, $DP_{10\%}^{TO}$; (**q**) CNN, $DP_{30\%}^{TO}$; (**r**) CNN, $DP_{50\%}^{TO}$; (**s**) CNN, $DP_{70\%}^{TO}$; (**t**) CNN, $DP_{90\%}^{TO}$; (**u**) CNN, $BMCI^{TO}$.