```matlab
% ========================
% FAST Solar Home Energy Forecasting Using LSTM
% ========================
clc; clear; close all;

%% 1️⃣Load and Preprocess Data

opts = detectImportOptions('Final_Fixed_Load_Data_Modified.xlsx',
'VariableNamingRule', 'preserve');
opts = setvaropts(opts, 'Date', 'Type', 'char'); % Read Date as text
loadData = readtable('Final_Fixed_Load_Data_Modified.xlsx', opts);

solarData = readtable('Final_Corrected_Solar_Data.csv');
batteryData = readtable('Final_Battery_Data.csv');

%% 2️⃣Convert Dates & Merge Data
loadData.Date = datetime(loadData.Date, 'InputFormat', 'yyyy-MM-dd HH:mm:ss',
'Format', 'yyyy-MM-dd HH:mm:ss');
solarData.Date = datetime(solarData.Date, 'InputFormat', 'yyyy-MM-dd HH:mm:ss');
batteryData.Date = datetime(batteryData.Date, 'InputFormat', 'yyyy-MM-dd
HH:mm:ss');

data = outerjoin(solarData, batteryData, 'Keys', 'Date', 'MergeKeys', true);
data = outerjoin(data, loadData, 'Keys', 'Date', 'MergeKeys', true);

%% 3️⃣Feature Engineering
data.Solar_Power = data.GHI * 0.18 * 10; % Estimated solar power
data.Battery_SOC = (data.Watt_hr / (10 * 1000)) * 100; % Battery SOC in %
data.Load_Demand = data.Load_kW; % Load demand in kW

X = normalize([data.Solar_Power, data.Battery_SOC, data.Load_Demand]);
X = fillmissing(X, 'linear'); % Fill missing values

%% 4️⃣Optimize Data for Speed
sample_size = min(5000, size(X, 1)); % Use first 5000 samples (reduced from
10,000)
X = X(1:sample_size, :);

if isempty(X)
    error("❌ No valid data for training. Check dataset.");
end

X_train = X(1:end-24, :);
Y_train = X(2:end-23, :);

if isempty(X_train) || isempty(Y_train)
    error("❌ Training data is empty after preprocessing.");
end

%% 5️⃣Define Faster LSTM Model
layers = [
    sequenceInputLayer(3)
    lstmLayer(15, 'OutputMode', 'last') % Fewer neurons + 'last' mode speeds up
training
    fullyConnectedLayer(3)
    regressionLayer];

%% 6️⃣Faster Training Settings
try
```

```
    gpuInfo = gpuDevice();
    executionEnv = 'gpu';
    disp("✅Using GPU for faster training.");
catch
    executionEnv = 'cpu';
    disp("□ GPU not available. Using CPU (slower).");
end

options = trainingOptions('adam', ...
    'MaxEpochs', 30, ... % Reduce epochs
    'MiniBatchSize', 128, ... % Larger batch size
    'ExecutionEnvironment', executionEnv, ...
    'Verbose', false);

%% 7️⃣Train the Optimized Model
net = trainNetwork(X_train', Y_train', layers, options);

%% 8️⃣Faster Predictions with Variability
X_test = X(end-23:end-1, :);
Y_pred = predict(net, X_test')';

% Denormalize Predictions
predictedSolar = Y_pred(:,1) * std(data.Solar_Power) + mean(data.Solar_Power);
predictedSOC = Y_pred(:,2) * std(data.Battery_SOC) + mean(data.Battery_SOC);
predictedLoad = Y_pred(:,3) * std(data.Load_Demand) + mean(data.Load_Demand);

% Add Small Random Variability to Prevent Linear Results
noiseFactor = 0.05; % Adjust this value for more/less randomness
predictedSolar = predictedSolar .* (1 + noiseFactor *
randn(size(predictedSolar)));
predictedSOC = predictedSOC .* (1 + noiseFactor * randn(size(predictedSOC)));
predictedLoad = predictedLoad .* (1 + noiseFactor * randn(size(predictedLoad)));

% Apply Moving Average Filter to Smooth Data
windowSize = 3;
predictedSolar = movmean(predictedSolar, windowSize);
predictedSOC = movmean(predictedSOC, windowSize);
predictedLoad = movmean(predictedLoad, windowSize);

% Generate Time Stamps
timeStamps = linspace(datetime('00:00', 'Format', 'HH:mm'), ...
                      datetime('23:59', 'Format', 'HH:mm'),
length(predictedSolar));

%% 9️⃣Plot Results
figure;
subplot(3,1,1);
plot(timeStamps, predictedSolar, 'r', 'LineWidth', 2);
title('Predicted Solar Power');
xlabel('Time'); ylabel('Power (kW)');

subplot(3,1,2);
plot(timeStamps, predictedSOC, 'b', 'LineWidth', 2);
title('Predicted Battery SOC');
xlabel('Time'); ylabel('SOC (%)');

subplot(3,1,3);
plot(timeStamps, predictedLoad, 'g', 'LineWidth', 2);
title('Predicted Load Demand');
```

```
xlabel('Time'); ylabel('Load (kW)');
```