IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

SPECIAL SECTION ON ADVANCED ENERGY STORAGE TECHNOLOGIES AND THEIR APPLICATIONS

# An Ensemble Learning Approach for Accurate Energy Load Prediction in Residential Buildings

**MABROOK AL-RAKHAMI[1,2], (Student Member, IEEE), ABDU GUMAEI[3], (Student Member, IEEE), AHMED ALSANAD[2], (Member, IEEE), ATIF ALAMRI[1], (Member, IEEE), AND MOHAMMAD MEHEDI HASSAN[1,2], (Senior Member, IEEE)**

[1]Chair of Pervasive and Mobile Computing (CPMC), Department of Information Systems, College of Computer and Information Sciences, King Saud University, Riyadh 11362, Saudi Arabia
[2]Department of Information Systems, King Saud University, Riyadh 11362, Saudi Arabia
[3]Department of Computer Science, King Saud University, Riyadh 11362, Saudi Arabia

Corresponding author: Mohammad Mehedi Hassan (mmhassan@ksu.edu.sa)

This work was supported by the Deanship of Scientific Research and Research Chair of Pervasive and Mobile Computing at King Saud University.

Experimented and Presented by
Md. Istiak Ahammed
ID: 2024100207
Robot and Smart System Engineering

# Outlines

- Introduction
- Proposed Ensemble Learning Approach
- XGBoost (eXtreme Gradient Boosting)
- Experimental Design
- Results and Discussion
- Conclusion

# Introduction

**General Background**

→ Energy efficiency in buildings is crucial for environmental and economic benefits.

→ Accurate energy load prediction (Heating and Cooling) helps in designing energy-efficient buildings.

→ Machine learning methods have been increasingly used for this purpose due to their predictive power and efficiency.

**Problems**

→ Traditional machine learning methods often suffer from overfitting, leading to inaccurate predictions.

→ There is a need for models that can handle high-dimensional data and avoid overfitting while providing accurate predictions.

# Introduction

**Previous Gap**

→ Existing methods lack robustness and fail to generalize well, especially in complex scenarios with high-dimensional data.

→ Overfitting is a common problem that undermines the reliability of predictions in traditional models.

**Research Objectives**

→ Develop an ensemble learning approach using the XGBoost algorithm to improve load prediction accuracy and robustness.

→ Conduct extensive experiments to validate the proposed approach against traditional methods.

→ Demonstrate the effectiveness of XGBoost in predicting energy loads in residential buildings, aiming for high accuracy and generalizability.

# Proposed Ensemble Learning Approach

## Ensemble Learning



# XGBoost (eXtreme Gradient Boosting)

- **XGBoost** (eXtreme Gradient Boosting) is a powerful and efficient implementation of the gradient boosting framework and able to adjust its scale when needed.
- Developed by Tianqi Chen, XGBoost is known for its performance and speed

## Gradient Boost

Step1→ Start by computing the initial prediction, which is often the mean of the target values
Step 2→ Compute Residuals: Calculate the difference between the actual values and the predicted values
Step 3 → Train a new model (typically a decision tree) to predict the residuals.
Step 4 → Update the initial predictions by adding the predictions of the residuals, multiplied by the learning rate.

The new prediction after first iteration: 173 + learning rate (0.1) x Pred Residual (1.5)

| Age | BMI | Height | Pred |
|---|---|---|---|
| 21 | 24 | 174 | 173 |
| 22 | 26 | 176 | 173 |
| 23 | 30 | 169 | 173 |

| Actual Height | Pred Height | Residual | Pred Residual |
|---|---|---|---|
| 174 | 173 | 1 | 1.5 |
| 176 | 173 | 3 | 5 |
| 169 | 173 | -4 | -1 |

Final prediction =Base value + (LR x 1st PredRes by RM1) + (LR X 2nd PredRes by RM2)+....
= 173 + (0.1X1.5) + (0.1 X 1.2) +.....

# XGBoost (eXtreme Gradient Boosting)

## XGBoost Algorithm

**Step 1:** Sort and split each predictor (input variable) to find the best splitting point.

**Step 2:** Optimize the tree depth by choosing the best splitting descriptor.

**Step 3:** Repeat steps 1 and 2 until you reach the maximum tree depth.

**Step 4:** Calculate the prediction score for each tree leaf.

**Step 5:** Prune (remove) nodes with negative scores.

**Step 6:** Repeat the entire process for all trees in the model.

**Algorithm 1** Training Algorithm of the XGBoost Model

**Input:** Vectors of energy load predictors ($V_n$), number of trees ($R$);

**Output:** XGBosst trained model;

1. **For each** predictor ($v_i$),
   1.1. $v_i \leftarrow$ Sort($v_i$);
   1.2. $p_i \leftarrow$ Split_best (*lowest_gain*($v_i$));
   // Compute tree depth by optimizing the objective function and choosing the descriptor of the best splitting point
2. Tree_depth $\leftarrow$ Optimizes (*ObjTraFunc* $\leftarrow$ *Choose* (*descriptor* ($p_i$));
3. Repeat (1 and 2) until *Tree_depth* == *Max_tree_depth*;
4. *Tree_leaves* $\leftarrow$ Prediction_score (*Tree_depth*);
5. Bottom-up_Prune_negative_nodes (*Tree_leaves*);
6. Repeat steps (1-5) until cumulative training covers all trees in R;

# XGBoost (eXtreme Gradient Boosting)

## Mathematical Equation of XGBoost Algorithm

Predicted Score Sum

$$\hat{y}_i = \sum_{r=1}^{R} f_r\,(x_i)\,, f_r \in S$$

Updated Prediction

$$\hat{y}_i^{(t)} = \sum_{r=1}^{R} f_r(x_i) = \hat{y}_i^{(t-1)} + f_t\,(x_i)$$

Objective Function

$$Obj\,(\Theta) = \sum_{i}^{n} l\,(y_i, \hat{y}_i) + \sum_{r}^{R} \Omega\,(f_r)$$

Simplified Objective Function

$$Obj(\Theta)^{(t)} = \sum_{i}^{n} l\left(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\right) + \Omega\,(f_t)$$

Regularization Term

$$\Omega\,(f) = \varphi T + \frac{1}{2}\vartheta \sum_{j=1}^{T} s_j^2$$

$f_r = Independent\ tree\ with\ scores\ of\ each\ leaf$

$l = differentialbe\ loss\ function\ between\ actual\ \&\ predicted$

Ω = regularization term to avoid overfitting

$T = leaf\ count, s = leaf\ score, \varphi\ and\ \vartheta = controlling\ constant\ for\ regularization$

# Experimental Design

## Dataset

**Dataset Source:** The dataset used is from a simulation of various residential building designs created with Ecotec software, which publicly available. **Number of Samples**: 768 samples

Input Attributes

| Name | Unit |
|---|---|
| Overall height | m |
| Relative compactness | NA |
| Wall area | $m^2$ |
| Surface area | $m^2$ |
| Roof area | $m^2$ |
| Glazing area distribution | NA |
| Glazing area | $m^2$ |
| Orientation | Direction |

Output Responses

| Name | Unit |
|---|---|
| Cooling load (CL) | kW |
| Heating load (HL) | kW |

# Experimental Design

## Performance Metrics

The performance metrics used to evaluate the proposed energy prediction model are statistical measures.

$$\mathrm{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}$$

$$\mathrm{R-squared} = 1 - \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y}_i)^2}$$

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$

$$MAPE = 100\% \times \frac{1}{N}\sum_{i=1}^{N}\frac{|y_i - \hat{y}_i|}{y_i}$$

$x_i = Input, y_i = actual\ output,$
$\hat{y}_i = predicted\ output,$
$\bar{y}_i = mean\ of\ actual\ output$

```
from sklearn.metrics import mean_absolute_error,
mean_squared_error, r2_score, mean_absolute_percentage_error

# Calculate performance metrics for HL and CL
mae_hl = mean_absolute_error(y_test_hl, y_pred_hl)
rmse_hl = np.sqrt(mean_squared_error(y_test_hl, y_pred_hl))
mape_hl = mean_absolute_percentage_error(y_test_hl, y_pred_hl) * 100
r2_hl = r2_score(y_test_hl, y_pred_hl)

mae_cl = mean_absolute_error(y_test_cl, y_pred_cl)
rmse_cl = np.sqrt(mean_squared_error(y_test_cl, y_pred_cl))
mape_cl = mean_absolute_percentage_error(y_test_cl, y_pred_cl) * 100
r2_cl = r2_score(y_test_cl, y_pred_cl)
```

# Experimental Design

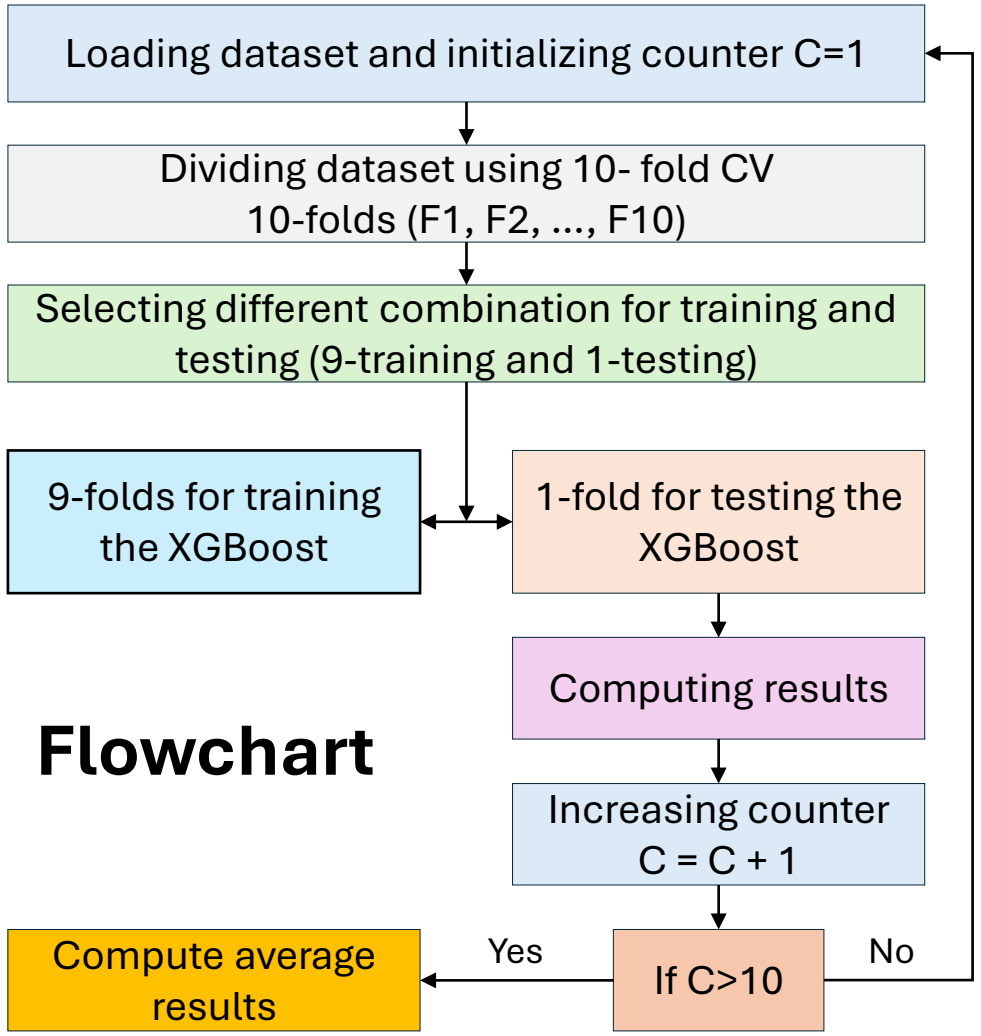
## Parameter Initialization for XGBoost (Grid Search)

| Parameter | Description | Paper's value | Experiment's value |
|---|---|---|---|
| learning_rate | Step size shrinkage used to prevent overfitting | 0.1 | 0.1 |
| gamma | Minimum loss reduction to make a split | 0 | 0 |
| max_depth | Maximum depth of a tree | 150 | 100 |
| n_estimators | Number of boosting rounds | 2000 | 2000 |
| reg_alpha | L1 regularization term on weights | 0.01 | 0.1 |
| reg_lambda | L2 regularization term on weights | 0.8 | 1 |
| subsample | Fraction of samples used for fitting each tree | 0.7/0.75 (HL/CL) | 0.8 |
| colsample_bytree | Fraction of features used for each tree | 0.7 | 0.7 |
| min_child_weight | Whether to print running messages | 1.8 | 1 |

# Experimental Design

## Training and Testing

- 10-fold cross-validation has been conducted
- Energy efficiency dataset is divided into 10 folds
- 9 of them are utilized to train the model, and the remaining fold is applied for testing.
- Repeated this step 10 times, each time a different fold from the dataset is used for testing and the remaining 9 folds for training
- Hyper parameters has defined by using Grid Search techniques



# Experimental Design

## Training and Testing Code

```
# Load dataset
data = pd.read_csv('../data/ENB2012_data_with_columns.csv')

# Check for NaN values and replace or drop them
data = data.replace([np.inf, -np.inf], np.nan)
data = data.dropna()

# Define input and output variables
X = data.iloc[:, :-2]
y_hl = data['Heating_Load']
y_cl = data['Cooling_Load']


# Initialize models with best parameters
best_params_hl = {'colsample_bytree': 0.7, 'gamma': 0, 'learning_rate': 0.1, 'max_depth': 150,
'min_child_weight': 1.8, 'n_estimators': 2000, 'reg_alpha': 0.01, 'reg_lambda': 0.8, 'subsample': 0.7, 'silent':
1, 'nthread': -1}
best_params_cl = {'colsample_bytree': 0.7, 'gamma': 0, 'learning_rate': 0.1, 'max_depth': 150,
'min_child_weight': 1.8, 'n_estimators': 2000, 'reg_alpha': 0.01, 'reg_lambda': 0.8, 'subsample': 0.75, 'silent':
1, 'nthread': -1}

xgb_model_hl = XGBRegressor(**best_params_hl)
xgb_model_cl = XGBRegressor(**best_params_cl)
```

```
# Perform 10-fold cross-validation
kf = KFold(n_splits=10, shuffle=True, random_state=42)

results_hl = []
results_cl = []

actual_vs_predicted_hl = []
actual_vs_predicted_cl = []

fold = 1
for train_index, test_index in kf.split(X):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train_hl, y_test_hl = y_hl.iloc[train_index], y_hl.iloc[test_index]
    y_train_cl, y_test_cl = y_cl.iloc[train_index], y_cl.iloc[test_index]

    # Train the models
    xgb_model_hl.fit(X_train, y_train_hl)
    xgb_model_cl.fit(X_train, y_train_cl)

    # Predict
    y_pred_hl = xgb_model_hl.predict(X_test)
    y_pred_cl = xgb_model_cl.predict(X_test)

    # Collect actual vs predicted values
    actual_vs_predicted_hl.append((y_test_hl, y_pred_hl))
    actual_vs_predicted_cl.append((y_test_cl, y_pred_cl))

    # Calculate performance metrics
    mae_hl = mean_absolute_error(y_test_hl, y_pred_hl)
    rmse_hl = np.sqrt(mean_squared_error(y_test_hl, y_pred_hl))
    mape_hl = mean_absolute_percentage_error(y_test_hl, y_pred_hl) * 100
    r2_hl = r2_score(y_test_hl, y_pred_hl)

    mae_cl = mean_absolute_error(y_test_cl, y_pred_cl)
    rmse_cl = np.sqrt(mean_squared_error(y_test_cl, y_pred_cl))
    mape_cl = mean_absolute_percentage_error(y_test_cl, y_pred_cl) * 100
    r2_cl = r2_score(y_test_cl, y_pred_cl)

    results_hl.append([fold, mae_hl, rmse_hl, mape_hl, r2_hl])
    results_cl.append([fold, mae_cl, rmse_cl, mape_cl, r2_cl])

    fold += 1
```

# Results and Discussion

(a) Original table

| Fold Number | MAE (kW) | RMSE (kW) | MAPE (%) | R-squared |
|---|---|---|---|---|
| 1 | 0.1773 | 0.2697 | 0.9952 | 0.9993 |
| 2 | 0.1261 | 0.1605 | 0.6682 | 0.9997 |
| 3 | 0.1486 | 0.1904 | 0.7957 | 0.9996 |
| 4 | 0.1669 | 0.2328 | 0.8667 | 0.9994 |
| 5 | 0.1514 | 0.2136 | 0.8054 | 0.9996 |
| 6 | 0.1626 | 0.3968 | 0.9503 | 0.9987 |
| 7 | 0.1989 | 0.3295 | 1.0868 | 0.9989 |
| 8 | 0.1594 | 0.2176 | 0.7148 | 0.9995 |
| 9 | 0.1806 | 0.2318 | 0.911 | 0.9995 |
| 10 | 0.1818 | 0.249 | 0.8669 | 0.9994 |
| **Avg.** | **0.165** | **0.249** | **0.866** | **0.9994** |
| **Std.** | **0.01574** | **0.04970** | **0.09606** | **0.00024** |

(b) Experimental table

| Fold Number | MAE (kW) | RMSE (kW) | MAPE (%) | R-squared |
|---|---|---|---|---|
| 1 | 0.194106 | 0.27829 | 0.93398 | 0.999294 |
| 2 | 0.198931 | 0.262215 | 0.952896 | 0.999302 |
| 3 | 0.161475 | 0.29834 | 0.793821 | 0.99907 |
| 4 | 0.192752 | 0.269674 | 1.016883 | 0.999317 |
| 5 | 0.206243 | 0.419884 | 1.188685 | 0.998307 |
| 6 | 0.164589 | 0.230978 | 0.781186 | 0.999518 |
| 7 | 0.206469 | 0.327119 | 0.858542 | 0.998983 |
| 8 | 0.191862 | 0.301785 | 0.976863 | 0.999112 |
| 9 | 0.166564 | 0.213396 | 0.880673 | 0.999546 |
| 10 | 0.19486 | 0.305161 | 1.008449 | 0.99876 |
| **Avg.** | **0.187785** | **0.290684** | **0.939198** | **0.999121** |
| **Std.** | **0.017081** | **0.057111** | **0.120527** | **0.000373** |

**TABLE 2.** Results of performance metrics for energy predicting of HL.

# Results and Discussion

(a) Original table

| Fold Number | MAE (kW) | RMSE (kW) | MAPE (%) | R-squared |
|---|---|---|---|---|
| 1 | 0.321 | 0.4747 | 1.2408 | 0.9974 |
| 2 | 0.3254 | 0.4877 | 1.2957 | 0.9974 |
| 3 | 0.2702 | 0.3844 | 1.2204 | 0.9979 |
| 4 | 0.2987 | 0.4139 | 1.2248 | 0.9983 |
| 5 | 0.3006 | 0.4765 | 1.1471 | 0.997 |
| 6 | 0.3112 | 0.5418 | 1.109 | 0.9968 |
| 7 | 0.3005 | 0.4876 | 1.1448 | 0.9976 |
| 8 | 0.2829 | 0.4053 | 1.1116 | 0.9981 |
| 9 | 0.2022 | 0.2627 | 0.8574 | 0.9992 |
| 10 | 0.3725 | 0.525 | 1.3304 | 0.9972 |
| **Avg.** | **0.299** | **0.446** | **1.168** | **0.9980** |
| **Std.** | **0.02805** | **0.06351** | **0.09422** | **0.00055** |

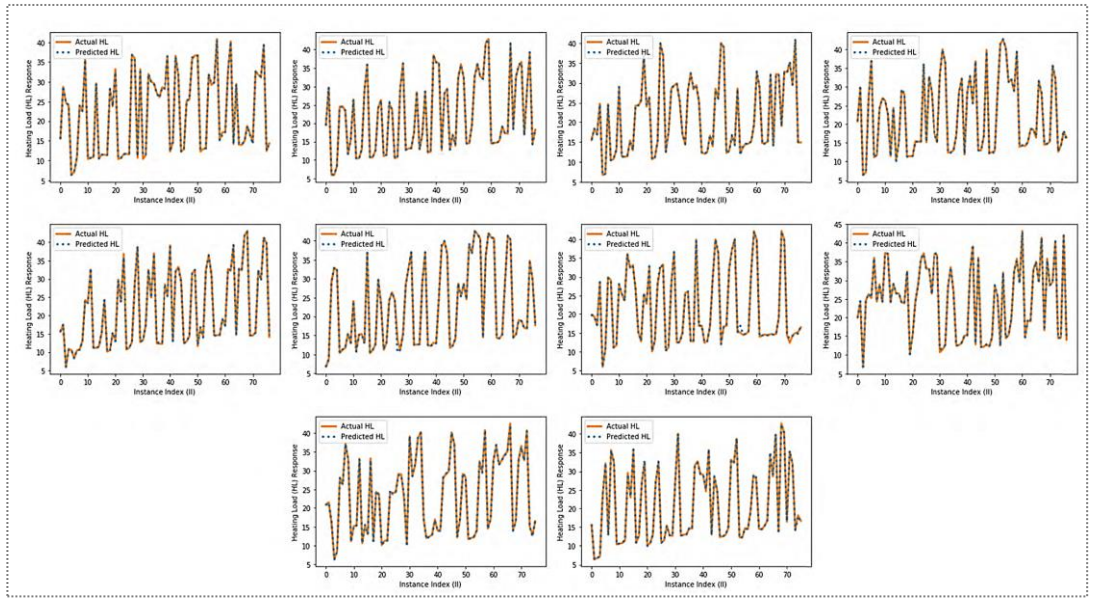
(b) Experimental table

| Fold Number | MAE (kW) | RMSE (kW) | MAPE (%) | R-squared |
|---|---|---|---|---|
| 1 | 0.353798 | 0.537199 | 1.309914 | 0.997079 |
| 2 | 0.297095 | 0.423075 | 1.140242 | 0.997927 |
| 3 | 0.319763 | 0.500917 | 1.221975 | 0.997052 |
| 4 | 0.352545 | 0.496526 | 1.470156 | 0.997224 |
| 5 | 0.374662 | 0.600763 | 1.393107 | 0.996307 |
| 6 | 0.40214 | 0.573561 | 1.566273 | 0.996597 |
| 7 | 0.4109 | 0.589612 | 1.507326 | 0.996185 |
| 8 | 0.32727 | 0.482912 | 1.166158 | 0.997382 |
| 9 | 0.301474 | 0.39535 | 1.219702 | 0.998242 |
| 10 | 0.270597 | 0.426214 | 1.079473 | 0.997528 |
| **Avg.** | **0.341025** | **0.502613** | **1.307433** | **0.997152** |
| **Std.** | **0.046116** | **0.07267** | **0.168542** | **0.000663** |

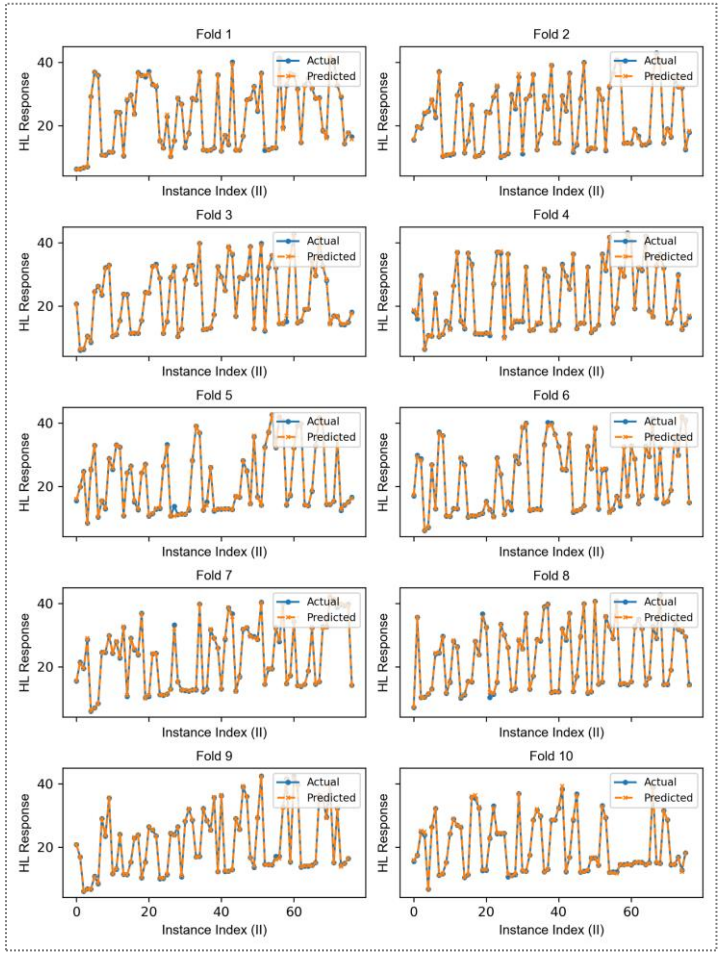
**TABLE 3.** Results of performance metrics for energy predicting of CL.

# Results and Discussion

Actual and predicted energy values of heating load



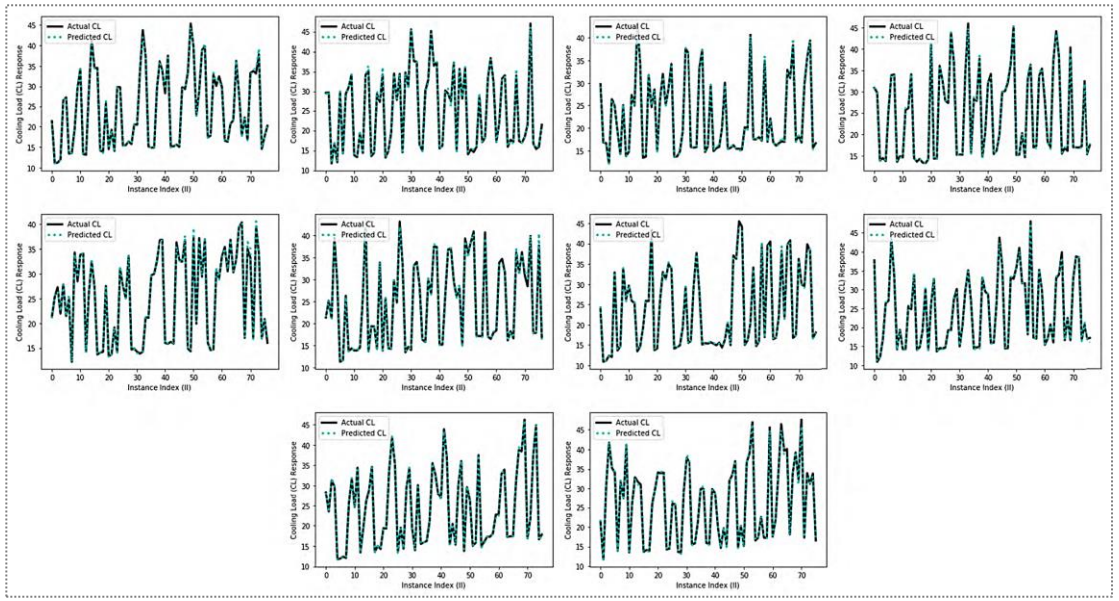

(a) Original figure



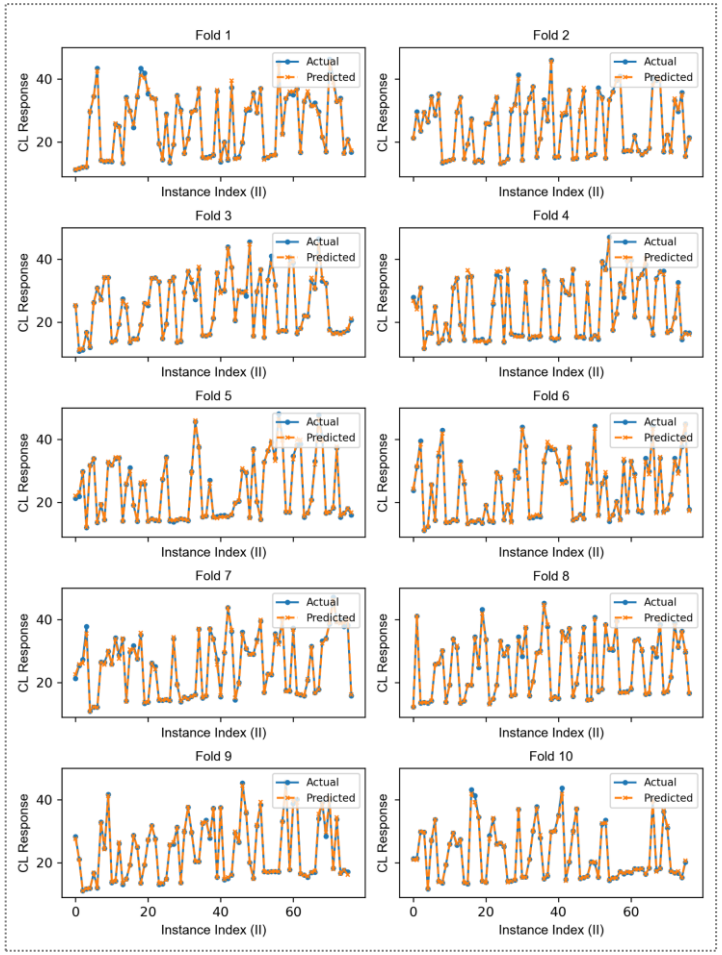

(b) Experimental figure

# Results and Discussion

Actual and predicted energy values of cooling load



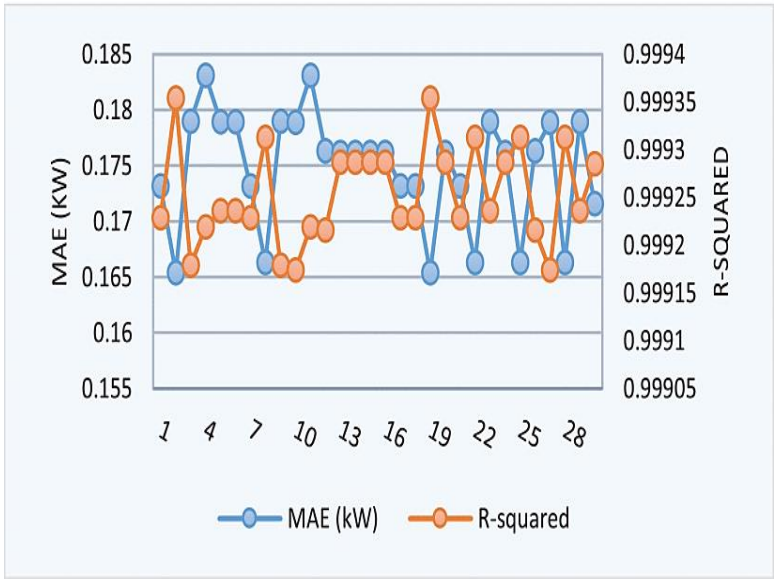

(a) Original figure




(b) Experimental figure

# Results and Discussion

(a) Original figure




(b) Experimental figure

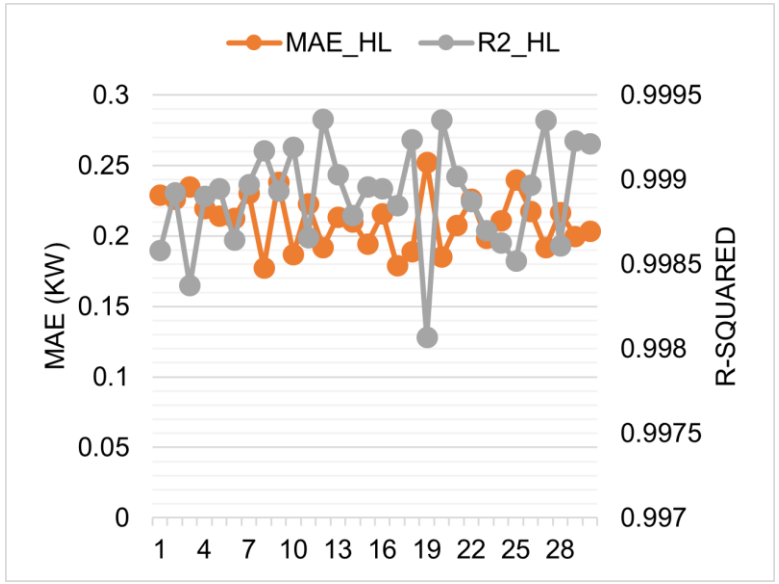
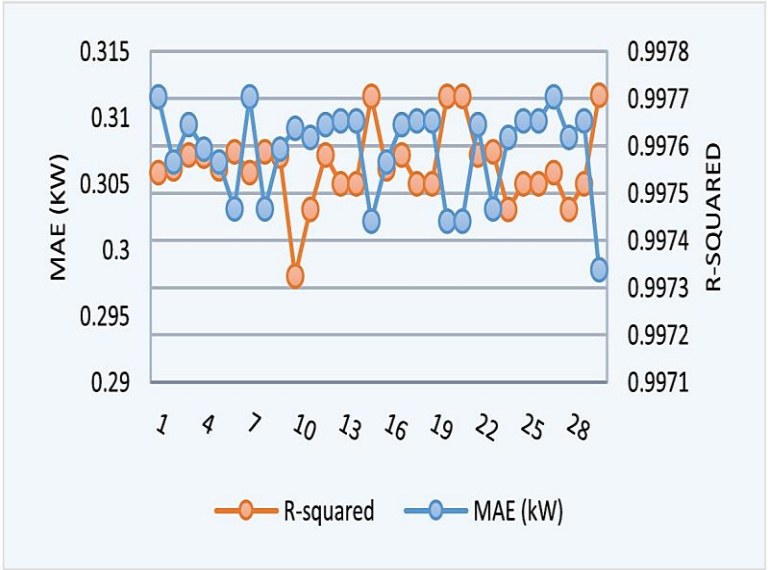



**FIGURE 7.** The relation between R-Squared and MAE values for HL in 30 times run

# Results and Discussion

(a) Original figure




(b) Experimental figure

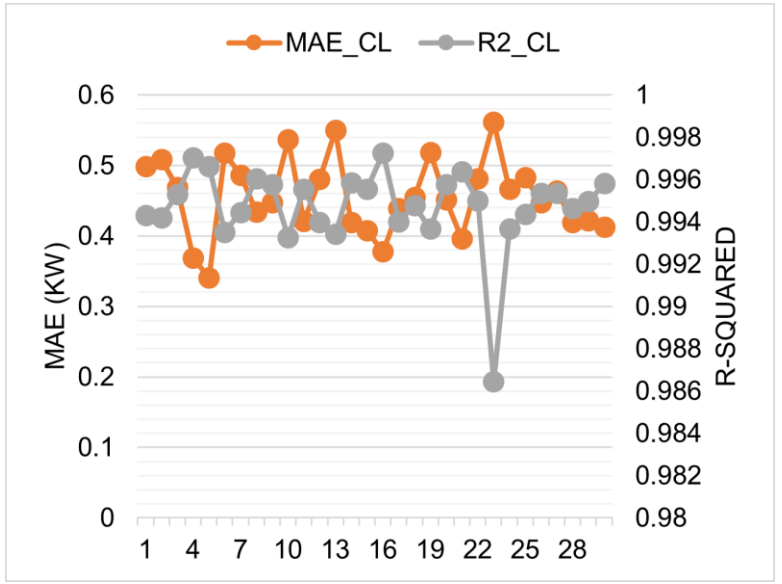
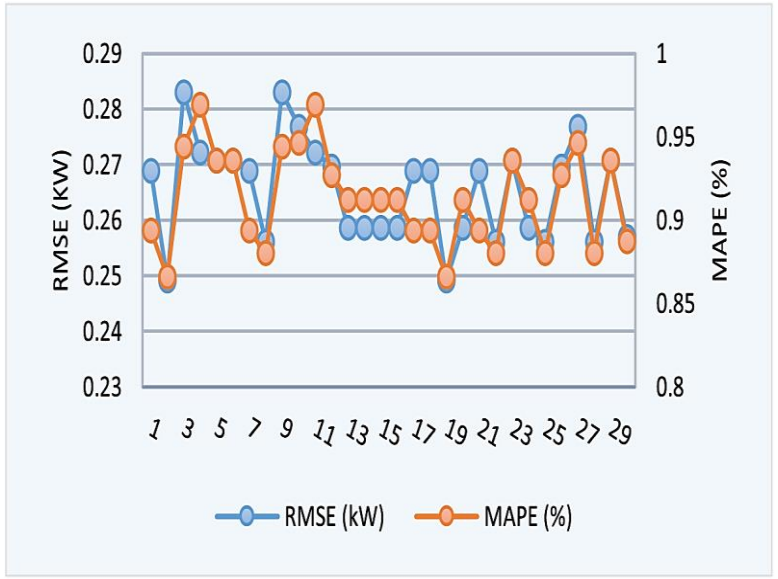



**FIGURE 9.** The relation between R-Squared and MAE values for CL in 30 times run

# Results and Discussion

(a) Original figure




(b) Experimental figure

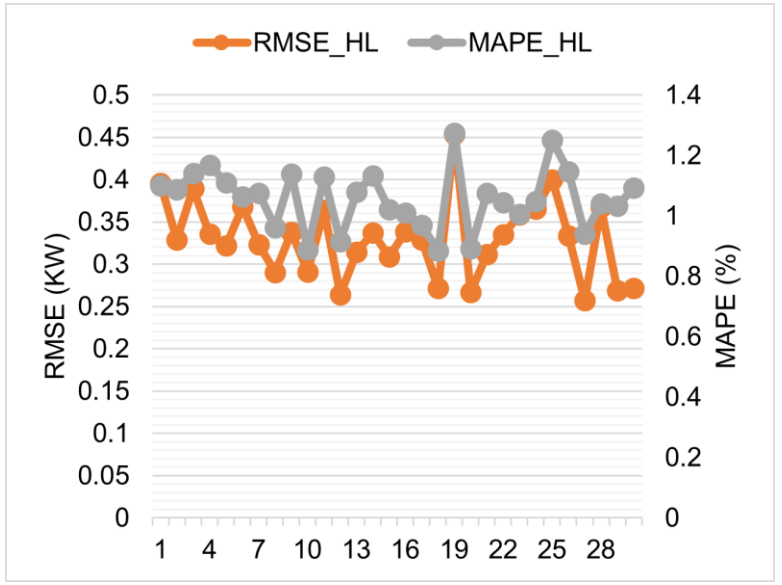
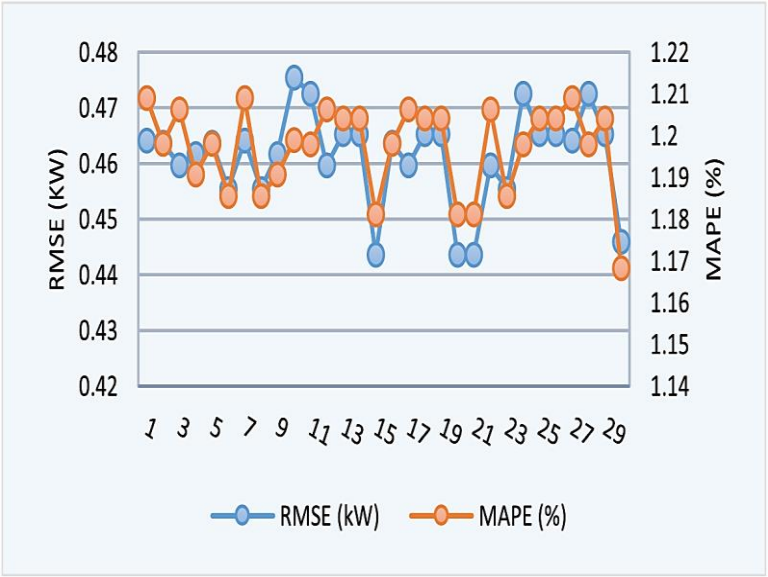



**FIGURE 8.** The relation between RMSE and MAPE values for HL in 30 times run

# Results and Discussion

(a) Original figure




(b) Experimental figure

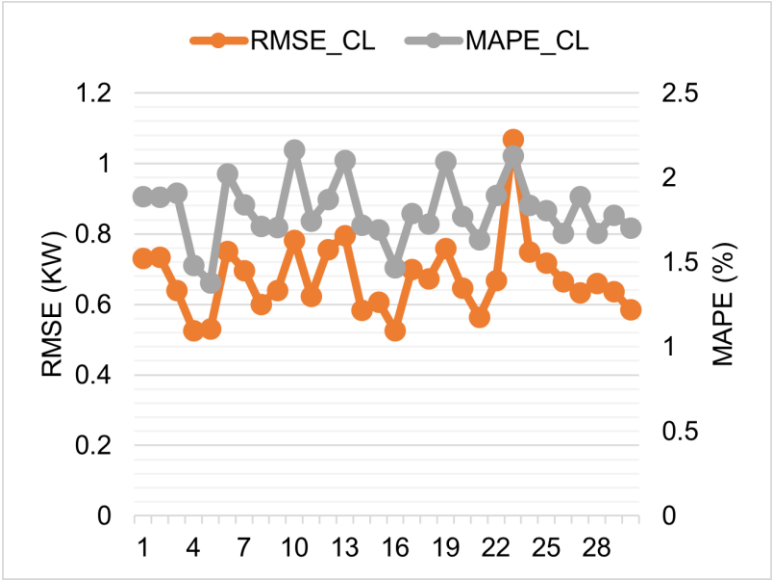
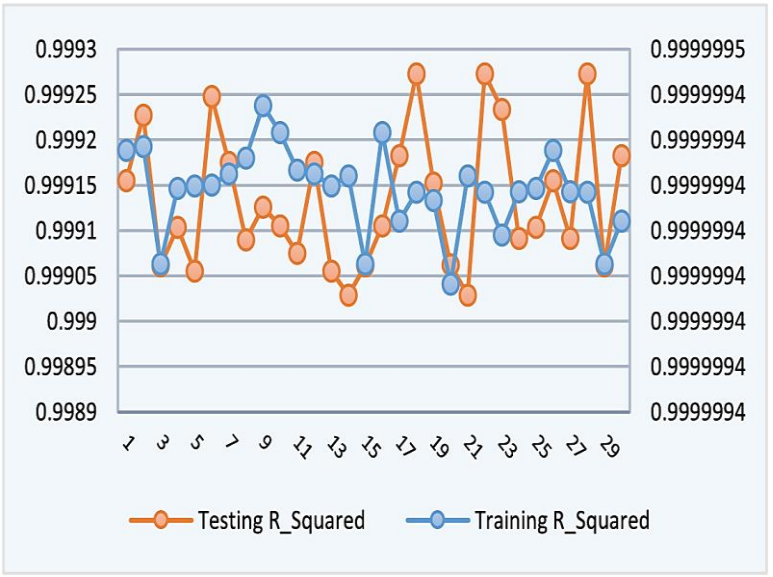



**FIGURE 10.** The relation between RMSE and MAPE values for CL in 30 times run

# Results and Discussion

(a) Original figure




(b) Experimental figure

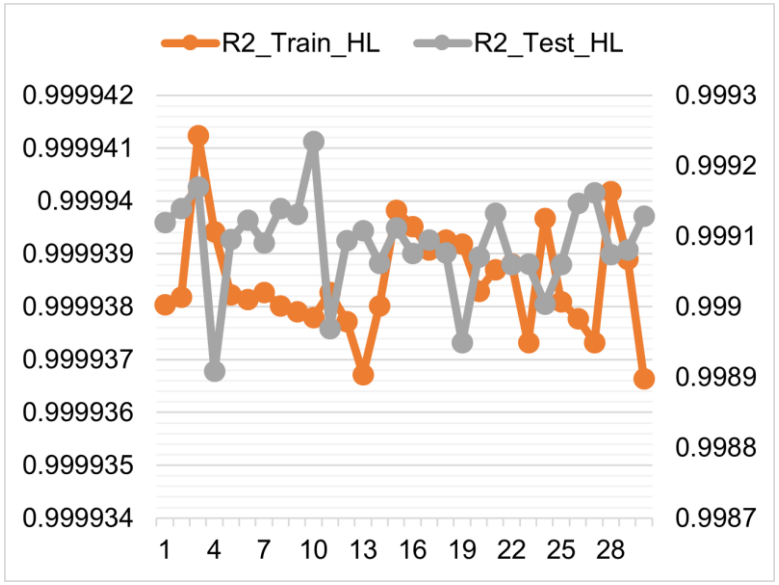
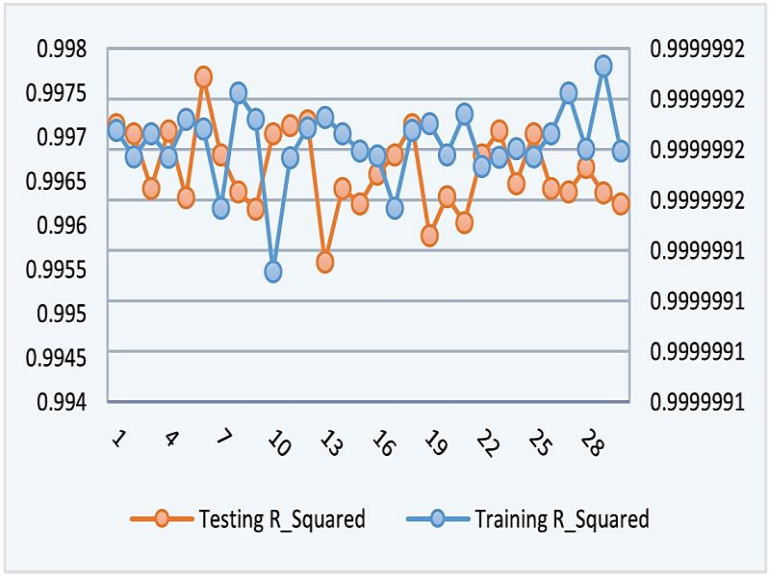



**FIGURE 11.** Comparison results of R-Squared between training and testing set for HL in the 30-times run of 10-cross validation.

# Results and Discussion

(a) Original figure




(b) Experimental figure

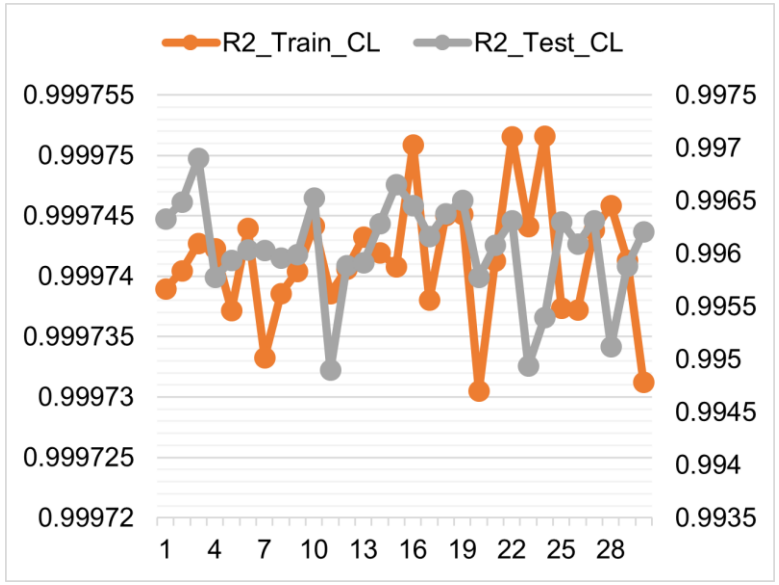



**FIGURE 12.** Comparison results of R-Squared between training and testing set for CL in the 30-times run of 10-cross validation.

# Results and Discussion

## Heating Load

| Ref | Model | MAE (kW) | RMSE (Kw) | MAPE (%) | R-squared |
|---|---|---|---|---|---|
| Tasnas and Xifara (2012) | Random forest | 0.51 | - | 2.2 | - |
| Chen and Cao (2014) | Ensemble model | 0.34 | 0.46 | - | 0.998 |
| Chou and Bui (2014) | Ensemble model | 0.236 | 0.35 | 1.1 | 0.999 |
| Castelli et al. (2015) | Genetic model | 0.38 | - | 0.43 | - |
| Duarte et al. (2017) | Neural network | 0.315 | 0.22 | 1.4 | 0.998 |
| Goliatt et al. (2018) | Gaussian process | 0.251 | 0.38 | 1.3 | 0.999 |
| **Proposed by authors** | **XGBoost ensemble** | **0.175** | **0.265** | **0.913** | **0.9993** |
| **My experimental results** | **XGBoost ensemble** | **0.187** | **0.291** | **0.939** | **0.9991** |

## Cooling Load

| Ref | Model | MAE (kW) | RMSE (Kw) | MAPE (%) | R-squared |
|---|---|---|---|---|---|
| Tasnas and Xifara (2012) | Random forest | 1.42 | - | 4.6 | - |
| Chen and Cao (2014) | Ensemble model | 0.68 | 0.97 | - | 0.99 |
| Chou and Bui (2014) | Ensemble model | 0.89 | 1.57 | 3.5 | 0.986 |
| Castelli et al. (2015) | Genetic model | 0.97 | - | 3.4 | - |
| Duarte et al. (2017) | Neural netwrok | 0.565 | 0.84 | 2.3 | 0.991 |
| Goliatt et al. (2018) | Gaussian process | 0.448 | 0.67 | 1.8 | 0.998 |
| **Proposed by authors** | **XGBoost ensemble** | **0.307** | **0.461** | **1.197** | **0.998** |
| **My experimental results** | **XGBoost ensemble** | **0.341** | **0.502** | **1.307** | **0.997** |

# Conclusion

## Robust Model & Experimental Results

- ✓ Developed an ensemble learning approach using the XGBoost algorithm
- ✓ Achieved lower mean square error (MSE) and higher accuracy compared to state-of-the-art approaches
- ✓ Demonstrated stability in predictions across different test runs
- ✓ Effectively mitigated overfitting issues, providing reliable predictions for heating load (HL) and cooling load (CL)

## Implication & Future Works

- ✓ XGBoost can be a valuable addition for predicting building energy loads
- ✓ Helps building engineers make informed decisions to design energy-efficient buildings
- ✓ Further improve the model by integrating more diverse datasets and refining feature selection techniques
- ✓ Investigate the model's performance in real-world scenarios and its adaptability to different climatic conditions