Energy Reports 15 (2026) 109071

Contents lists available at ScienceDirect

Energy Reports

journal homepage: www.elsevier.com/locate/egyr

Review article

Comparative analysis of AI-based regression models for solar power forecasting: Techniques, preprocessing, and evaluation

Musa Terkes [a], Alpaslan Demirci [a], Idriss Dagal [b], Erdin Gokalp [a], Umit Cali [c,d,e,*]

[a] Department of Electrical Engineering, Yildiz Technical University, İstanbul, Türkiye
[b] Department of Electrical-Electronics Engineering, Istanbul Beykent University, Istanbul 34396, Türkiye
[c] Department of Electric Energy, Norwegian University of Science and Technology, Trondheim, Norway
[d] School of Physics, Engineering and Technology, University of York, York, United Kingdom
[e] American University of Sharjah, Artificial Intelligence, Smart Infrastructure and Robotics Research Center, Sharjah, United Arab Emirates

ARTICLE INFO

Keywords:
Artificial intelligence
Machine learning
Solar power forecasting
Regression models
Data preprocessing
Hyperparameter optimization

ABSTRACT

Accurate prediction of photovoltaic (PV) power output is crucial for improving the efficiency, planning, and stability of solar energy systems. While numerous machine learning (ML) and deep learning (DL) methods have been proposed for this task, practical, implementation-focused resources that guide researchers in model selection, tuning, and evaluation remain limited. This study presents an implementation-oriented comparative analysis of widely used AI-based regression techniques for solar power forecasting. The analysis encompasses key stages including data preprocessing, model selection, performance assessment, and hyperparameter optimization using a 20-year real-world solar dataset with hourly resolution. A set of widely used supervised regression models, including k-Nearest Neighbors, Linear Regression, Ridge and Lasso Regression, Decision Tree, Random Forest, Support Vector Regression (SVR), Gradient Boosting Regressor (GBR), Bagging, Voting Regressor, Multi-Layer Perceptron (MLP)-based Artificial Neural Network, and basic Deep Learning architectures are evaluated in this study. These models are assessed using Python-based tools such as scikit-learn and TensorFlow, with a focus on reproducibility and ease of adaptation for energy system applications. Rather than proposing novel model architectures, the study aims to serve as a practical guide for researchers and engineers, offering insight into the comparative performance and usability of existing regression techniques. The findings emphasize the importance of data quality, preprocessing, and tuning strategies in determining model performance. By combining theoretical clarity, implementation guidance, and empirical evaluation using open-source Python tools, this work provides a reproducible and practical benchmark for AI-driven solar power forecasting, addressing a critical gap in implementation-focused comparative studies.

1. Introduction

The academic discipline of artificial intelligence (AI), which is closely related to research fields such as statistical algorithms and data science, is known as machine learning (ML). In contrast, deep learning (DL), a more structured and hierarchical form of learning, encompasses artificial neural networks (ANNs) with multiple hidden layers and

Abbreviation: VR, Voting regressor; PCA, Principal component analysis; VMD, Improved variational mode decomposition; RSS, The residual sum of squares; L-BFGS-B, Limited-memory Broyden–Fletcher–Goldfarb–Shanno; SGD, Stochastic gradient descent; RLS, Regularized least squares; SAGA, Stochastic average gradient descent; MSE, Mean-squared error; OOB, Out-of-bag; XGBoost, Extreme gradient boosting; LOR, Logistic regression; RL, Reinforcement learning; RMSprop, Root mean square propagation; RBM, The restricted Boltzmann machine; MBE, Mean bias error; LAS, Lasso regression; GHI, Global horizontal irradiation; BR, Bagging regressor; MSLE, Mean squared logarithmic error; DT, Decision tree; GBR, Gradient boosting regressor; OLS, The ordinary least squares; RMSD, Root mean square deviation; RF, Random forest; AI, Artificial intelligence; ML, Machine learning; ANN, Artificial neural network; PV, Photovoltaic; DL, Deep learning; GA, Genetic slgorithm; SVM, Support vector machine; SVR, Support vector regressor; PSO, Particle swarm optimization; MAPE, Mean absolute percentage error; MAE, Mean absolute error; RMSE, Root mean squared error; R^2, The coefficient of determination; LSTM, Long short-term memory; CNN, Convolutional neural network; DBN, Deep belief network; RNN, Recurrent neural network; GRU, Gated recurrent units; DNN, Deep neural network; GSA, Grid search algorithm; MLP, Multi-layer perceptron; ELM, Extreme learning machine; K-NN, k-nearest neighbor; LR, Linear regression; RDG, Ridge regression.

* Corresponding author at: Department of Electric Energy, Norwegian University of Science and Technology, Trondheim, Norway.
E-mail addresses: musa.terkes@yildiz.edu.tr (M. Terkes), ademirci@yildiz.edu.tr (A. Demirci), idrissdagal@beykent.edu.tr (I. Dagal), gokalp@yildiz.edu.tr (E. Gokalp), umit.cali@ntnu.no (U. Cali).

https://doi.org/10.1016/j.egyr.2026.109071
Received 7 September 2025; Received in revised form 30 November 2025; Accepted 24 January 2026
2352-4847/© 2026 The Author(s). Published by Elsevier Ltd. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

Nomenclature

x	Query data point
y	Training data point (target label)
s	Predicted value or output of a neighbor
y_j	Actual output (label) of the j-th sample
s_j	Output/prediction of the j-th neighbor
$d(x,y)$	Distance between points x and y
D_H	Hamming distance
p	Parameter of the Minkowski distance metric
k	Number of nearest neighbors
m	Number of attributes/features
d	Dimensionality of the output space
w_{i,y_j}	Weight assigned to the neighbor y_j for query instance i
$s_i^{'}$	Weighted k-NN prediction for data point i
x_i	The i-th explanatory (independent) variable
y_i	Observed (actual) dependent variable for sample i
$\overline{y}$	Mean of dependent variable
α	Intercept term of the regression model
β	Regression coefficient vector
β_0	Intercept term in polynomial regression
$\beta_1,\beta_2,..\beta_h$	Polynomial regression coefficients for order h
$\widehat{\beta}$	Estimated regression coefficient vector
ε_i	Random error term (zero mean, constant variance)
$\|w\|_2^2$	$\ell 2$ penalty term
s	Sparsity index: number of non-zero coefficients
$y_i \in \mathbb{R}^n$	Training label/target vector
Q_m	Set of training samples at node m
n_m	Number of samples at node m
$Q_m^{left}(\theta)$	Left child subset after split θ
$Q_m^{right}(\theta)$	Right child subset after split θ
θ^*	Optimal split rule
p_i	Probability of class i
Y_{left}, Y_{right}	Mean values of targets for left/right subsets
r_{mi}	Residual at iteration m
β_m	Weight of weak learner m
a_m	Parameters of weak learner h
M	Number of boosting iterations / number of weak learners
S	Entropy of dataset
$Gain(S,A)$	Information gain for attribute A
R_{RF}^2	Coefficient of determination for RF
$\widehat{h}(x)$	Bagging prediction function
$h_A(x)$	Output of bagged tree ensemble
P_{voting}	Final prediction of the Voting Regressor
$f(x) = wx + b$	Linear SVR function
w	Weight vector
ζ_i, ζ_i^*	Slack variables measuring deviation outside the ε-tube
$\varnothing(x)$	Mapping to higher-dimensional feature space
σ	Free parameter in RBF kernel representing width
d	Degree of polynomial kernel
ε	Epsilon-insensitive loss parameter
$Q(i,j)$	Kernel matrix element $K(x_i, x_j)$
w_i, W	Weights of neurons or weight matrices
α^L	Output vector of layer L
τ_{jo}, τ_{ko}	Biases of hidden and output neurons
e_i	Model error for the i-th training sample
n	Number of observations (samples)
p	Number of predictors (regressors), with intercept included
X	Design matrix of predictors
y	Vector of dependent variable observations
b	Candidate coefficient vector in optimization
$x_i^T b$	Predicted output using coefficient vector b
$S(b)$	Residual sum of squares (RSS) as a function of coefficients
RSS	Residual Sum of Squares
U	Matrix of left singular vectors
V	Matrix of right singular vectors
L	Lower triangular matrix in Cholesky decomposition
A	Matrix being decomposed (e.g., X^TX)
$Cov[x,y]$	Covariance between variables x and y
$Var[x]$	Variance of x
$\overline{x}$	Mean of independent variable
S^2	Residual variance estimator
h	Degree of the polynomial regression model
w	Coefficient vector used in optimization form of Ridge/ Lasso
Σ	Diagonal matrix from singular value decomposition
λ	Regularization hyperparameter (Ridge and Lasso)
$L2$	$\ell 2$ norm (sum of squared coefficients)
$L1$	$\ell 1$ norm (sum of absolute coefficient values)
$\|w\|_1$	$\ell 1$ penalty term
$x_i \in \mathbb{R}^n$	Training feature vector (input sample)
$X = (x_1, x_2, x_n)$	Set of explanatory variables (features)
$Y = f(x)$	Target/response variable
t_m	Threshold value for candidate split
j	Feature index for candidate split
$\theta = (j, t_m)$	Candidate split rule
$p(c)$	Probability of class c in dataset
L, R	Left and right subsets after split
$g_m(x_i)$	Negative gradient (pseudo-residual)
$h(x; a_m)$	Weak learner (usually a small regression tree)
$F_m(x)$	Boosted model at iteration m
$F^*(x)$	Optimal prediction function
$Gini$	Gini impurity
$G(Q_m, \theta)$	Split quality function
MSE_{OOB}	Out-of-bag mean squared error
T	Number of learners/estimators in the ensemble
$\widehat{f}_t(x)$	Prediction of the t-th base learner
$h_i(x)$	Output of the i-th regression tree
P_n	Prediction of the n-th model in the ensemble
w_n	Weight assigned to the n-th model
b	Bias term
$K(x_i, x_j)$	Kernel function
γ	Kernel coefficient controlling the RBF decay rate
κ	Scaling parameter for sigmoid kernel
r	Coefficient in inhomogeneous polynomial kernel
$\alpha_i,\ \alpha_i^*$	Lagrange multipliers in SVR dual formulation
x_i	Input variable or neuron input
b, θ	Bias terms / thresholds
$\mathscr{X}_{ij}, \mathscr{X}_{kj}$	Weight connections of hidden and output layers
M	Number of hidden neurons
N	Number of input features
nc	Number of training samples
dW, db	Gradients of weights and biases
β	Decay parameter in RMSprop
ε	Small constant to avoid division by zero
$f(\bullet)$	Activation or kernel function
z_l^2	Net input to neuron l in second layer
$z_l^{(k)}$	Net input to neuron l in layer k
p_{k-1}, p_k	Number of units in layers $k-1$ and k
λ	Regularization coefficient
$\sum w_{ij}^2$	L2 penalty
$\dot{g}$	Derivative of activation function
α, n	Step size in optimization
η_0	Initial learning rate
$coef$	Averaged coefficients over iterations
b	Intercept (bias term) in linear model

$f(x)$	Scoring function or model output
$\dot{g}^{(k)}$	Derivative of activation function g at layer k
i	Index of training sample
k	Index of a layer in a deep network
l	Index referring to a neuron in current layer
m	Number of model features
p	Number of inputs in the first hidden layer
p_k	Number of units in layer k
$R(w)$	Regularization term
t	Time step (iteration)
w_j	Weight of feature j
ρ	Mixing coefficient in elastic net regularization
E	Error (loss) function
S_{dw}, S_{db}	Squared gradient accumulations (RMSprop/Adam)
V_{dw}, V_{db}	Momentum terms (Adam)
out	Output of the network
X_r	Input vector for generalized regression networks
Y_r	Predicted output vector
$a_l^{(k)}$	Activation of unit l in layer k
$F\left(x_1, x_2, \ldots, x_{p^{(0)}}\right)$	Final deep network model output
$J(w)$	Weight penalty term
δ_l^K	Error term at output layer
$\delta_l^{(k)}$	Error term at hidden layer k
$w := w - \eta.\nabla Q(w)$	Gradient update rule
$power_t$	Exponent in learning rate schedule
$a_j^{(k-1)}$	Activation of neuron j in layer $k-1$
$E(w, b)$	Regularized loss function
$g^{(k)}$	Activation function at layer k
h	Output activation function
j	Index referring to a neuron in a previous layer
K	Total number of layers
$L\left[y_i, f(x_w)\right]$	Loss function
n	Number of total training samples
p_{k-1}	Number of units in layer $k-1$
$Q(w)$	Objective function in SGD
$Q_i(w)$	Loss of the i^{th} example
t_0	Offset parameter in learning rate schedule
$\eta^{(t)}$	Time-dependent learning rate schedule

similar ML techniques, and can be carried out in supervised, semi-supervised, or unsupervised settings. Both approaches contribute to computational statistical methods and adapt to mathematical optimization problems. Based on data analysis, these techniques can serve many fields, such as pattern recognition, natural language processing, computer vision, speech recognition, email filtering, and more. Global temperature concerns have increased interest in clean energy pathways, particularly promoting the installation of photovoltaic (PV) panels, which are now more prominent in the market. Due to the inherent unpredictability of solar radiation, energy production from PV plants, and consequently the planning of sensitive and accurate energy models, becomes challenging. The clarification of PV plant participation in the day-ahead electricity market, the accurate planning of hybrid renewable energy system management in hourly intervals, and the determination of storage processes according to the electricity market, among other reasons, have brought renewable energy production estimation to the forefront, much like demand estimation. Especially, to effectively serve the purposes of multiple stakeholders with PV panel output power, short- or long-term estimates are essential, and ML and DL techniques provide adaptable approaches that align with data science goals. Additionally, the correlation relationships among factors affecting production estimation must be considered. The confirmed correlation between radiation and PV power production in correlation analyses highlights the importance of weather classification and cloud tracking for high estimating accuracy (Ahmed et al., 2020). The continuous changes in cloud positions, shapes, and densities, the difficulty of distinguishing between cloud edges and clear skies, and their impact on solar radiation are among the major challenges in estimating. The tracking of cloudy and overcast images with cameras directed at the sky and using cloud segmentations for ML prediction under various sky conditions can enhance model robustness (Park et al., 2021). While the U-Net deep neural network (DNN) architecture appropriately segments cloud pixels, it may not guarantee high accuracy in radiation and, therefore, production estimating. ANN-based predictions combined with genetic algorithms (GAs) or particle swarm optimization (PSO) stand out for their accuracy among ML techniques. For large datasets with high-quality preprocessing where outliers and missing values are eliminated, models that consider cloud cover and external inputs, in collaboration with physical models, should be developed using DNNs (Mellit et al., 2020). However, necessary precautions must be taken as prediction errors will inevitably rise in cloudy conditions. Another perspective suggests that weather classification may be the most effective factor in improving prediction models (Kumari and Toshniwal, 2021). The forecast horizon, weather classification, and performance metric criteria for solar radiation are affected by various factors. Additionally, aspects such as data types (comparative, real-time, and simulation), dataset characteristics (data origin, indoor and outdoor weather conditions), temporal resolution levels of the prediction, and dataset sizes (total forecast time) should be discussed in relation to training and testing prediction models. Data cleaning processes, such as normalization, wavelet transforms, and augmentation using generative adversarial networks, are recommended in ML techniques (Ahmed et al., 2020). Decomposition methods are commonly used in preprocessing techniques for renewable energy production forecasting. Another perspective is that dealing with large datasets requires significant data storage and high computational power units. When considering the advantages and disadvantages of each ML and DL-based technique, prediction performance varies depending on the dataset (Aslam et al., 2021). Optimization of network parameters is a necessity in this field, and the use of GAs and PSO is widespread (Ahmed et al., 2020). The most common approaches for applying metaheuristics in parameter selection are extreme ML and support vector machine (SVM). Techniques that guarantee higher R^2 scores also support performance in terms of mean absolute percentage error (MAPE) (Lai et al., 2020).

1.1. Literature review

To reduce the limitations of traditional ML techniques and improve estimating accuracy, DL models (long short-term memory (LSTM), deep belief network (DBN), echo state network, convolutional neural network (CNN), etc.) have gained prominence. These models are the most effective way to achieve invariant structures and nonlinear features, and they are known for their competitive estimating performance (Wang et al., 2020). However, performing a prediction at a single location is not a proper approach due to the memorization problem in data-driven learning processes. Energy production forecasts for short time horizons, up to a few days, are of particular interest to researchers. Although PV power can be accurately predicted a few steps ahead, high prediction accuracy may not be achievable as the number of steps increases. At each prediction step, applying a resampling process to create a new representation of the original time series and integrating it with ML algorithms can serve the relevant objective for more steps (Rana and Rahman, 2020). From this perspective, the use of ANN for short-term predictions in PV energy production is recommended, while extreme

ML is useful in online sequential adaptive network predictions, the Bootstrap technique is effective in uncertainty forecasting, and CNNs, which can appropriately extract nonlinear input-output relationships, stand out for their performance (Ahmed et al., 2020).

Although hybrid strategies require more training time, they exhibit high estimation performance for many models. Hybrid estimation strategies, which combine several techniques including LSTM, gated recurrent units (GRUs), recurrent neural network (RNN), and CNN, are expected to be the top three models of interest for researchers in PV energy production estimation (Alkhayat and Mehmood, 2021). Additionally, the combination of backpropagation and wavelet neural networks in estimation models is anticipated to enhance estimation performance (Zhang et al., 2021). Hybrid methods that include the most physically calculated techniques (such as separation and transposition) for operational day-ahead estimation of PV plants can reduce mean absolute error (MAE) by 5.2 % and 10.4 % compared to models that neglect the physical computational process and classical ML strategies (Mayer, 2022). Furthermore, in hybrid models, optimizing physical techniques plays a critical role in performance. The comparison of physical and ML models in PV power modeling, including global horizontal radiation, direct normal radiation, and global tilted radiation, is essential. RNN, which can utilize the time-series properties of the connections between units, can be adapted to measured weather data and PV power production data. By performing optimization processes through the adjustment of classification and layer options, and carrying out normalization processes, root mean square error (RMSE) can be reduced to 13.8 % and 13.2 % in energy supply and demand matching using single-layer and multi-layer RNN models (Park et al., 2021). However, in single model situations, LSTM and GRU are more effective than RNN in all conditions as they reduce the vanishing gradient problem. It is predicted that CNN-LSTM can improve estimation accuracy by 3.62 %, 25.29 %, 34.66 %, 37.37 %, and 26.20 % compared to CNN, LSTM, GRU, RNN, and DNN, respectively (Kumari and Toshniwal, 2021). While the CNN-LSTM model challenges this perception, complex data inputs such as images still need to be managed. Due to their ability to reduce mean deviation and root mean square deviation (RMSD), LSTM and GRU are recommended for better performance compared to physical models (Ramadhan et al., 2021). In statistical computations, GRU outperforms LSTM in terms of computation time (Rajagukguk et al., 2020). Based on accuracy, input data, estimation horizon, season, weather type, and training time, the most valuable approach to minimize RMSE for independent models is expected to be LSTM (Rajagukguk et al., 2020). Considering data preprocessing, model fitting, cross-validation, metric analysis, and hyperparameter tuning, LSTM models perform better than multi-layer perceptron (MLP) in short-term production estimation under different weather conditions, with 200 hidden nodes and a tanh activation function (average 0.497 RMSE) (Liu et al., 2021). While LSTM's RMSE may vary depending on the estimation horizon, it can guarantee a range between 30 W/m^2 and 76 W/m^2. However, as the estimation horizon increases, solar radiation and, consequently, PV energy production estimation performance decreases (Kumari and Toshniwal, 2021). In the research presented in (Konstantinou et al., 2021), the authors used LSTM deep networks for PV output power estimation in Nicosia, Cyprus, obtaining an average RMSE metric of 0.09394 along with k-fold cross-validation. If fixed parameters are considered as input, physical models are more effective in reducing RMSD for solar energy estimation.

Although the literature offers a broad range of ML- and DL-based approaches for solar power forecasting, several structural research gaps remain evident and continue to limit the comparability and generalizability of existing models. A key issue is the lack of standardized experimental settings: many studies rely on different datasets, sampling resolutions, preprocessing workflows, and climatic characteristics, making it difficult to draw uniform conclusions across conventional ML methods and more recent DL architectures. This heterogeneity is particularly apparent in the evaluation of hybrid models, such as CNN–LSTM, GRU–CNN, attention-based structures, and transformer-inspired architectures, whose reported performances are often based on isolated or non-equivalent experimental designs, preventing meaningful cross-study benchmarking. Another notable gap concerns geographic variability. Despite the strong dependence of solar irradiance on local climate and seasonal dynamics, only a limited number of studies conduct multi-climate or multi-region benchmarking, which restricts insights into the regional transferability and robustness of forecasting methods. Moreover, a substantial portion of the literature emphasizes accuracy metrics alone while providing limited analysis of computational cost, training time, hyperparameter sensitivity, or model generalizability, factors that are essential for the practical deployment of forecasting systems. Finally, the field still lacks transparent, accessible, and fully replicable tutorial-style implementations that guide early-career researchers through the complete workflow, from preprocessing and feature construction to training, hyperparameter tuning, and evaluation, using consistent datasets and unified metrics. Collectively, these gaps underscore the need for comparative frameworks that operate under harmonized conditions and offer reproducible, pedagogical insights. Addressing these shortcomings constitutes one of the central motivations for the comprehensive and practical comparative roadmap presented in this study.

1.2. Motivation and contributions

In recent years, the prediction of power output from PV panels, plants, and parks using a wide range of ML and DL techniques has attracted significant attention in the research community (also discussed in the Discussion section). Also, recent developments in the literature, particularly in studies published between 2022 and 2025, highlight a growing interest in hybrid DL architectures for solar forecasting. Many contemporary works increasingly explore CNN–LSTM configurations, attention-based temporal models, and transformer-inspired architectures, noting their effectiveness in capturing spatial–temporal dependencies present in irradiance and PV output data. While these approaches often report notable performance gains, they also underscore the persistent need for more standardized evaluation conditions, consistent preprocessing protocols, and transparent documentation of hyperparameter choices, considerations that align with the comparative and methodological perspective adopted in this study. In light of the research gaps identified in the recent literature, such as the absence of unified experimental environments, the scarcity of cross-model comparisons under identical conditions, and the limited benchmarking of emerging hybrid and attention-based DL architectures, this study provides a consistent and replicable baseline for PV power forecasting. By analyzing a long-term real-world dataset using a broad set of widely adopted ML and DL regressors, the study offers a controlled environment that enables fair comparison, thus addressing the fragmentation and methodological heterogeneity observed in prior works. In addition this study distinguishes itself from prior surveys and tutorial-style papers by providing a full empirical implementation of multiple widely used ML and DL regression models for PV power forecasting. The models are evaluated on a real-world solar dataset covering 20 years of hourly observations collected from NASA Power and susdesign, spanning Antalya (Türkiye). The dataset includes key variables such as solar irradiance, ambient temperature, wind speed, and historical PV output. Temporal integrity is maintained through continuous hourly recordings, and preprocessing steps, including missing data handling and normalization, ensure dataset consistency. Quantitative benchmarking is performed using RMSE, MAE, R^2, and MAPE metrics, with and without hyperparameter optimization. Additionally, the study investigates model-specific hyperparameter effects using GA-based TPOTRegressor and Grid Search methods, providing insights into how optimization strategies influence predictive performance across large-scale datasets. This article is designed as a pedagogical, tutorial-style comparative study aimed especially at early-career researchers seeking to understand

and apply widely used AI-based regression models for solar power forecasting. Rather than proposing a novel algorithm or striving for state-of-the-art accuracy, the study focuses on providing a hands-on, practical roadmap using the Python platform, with particular emphasis on interpretability, implementation, and comparative model analysis. The goal is to bridge the gap between theoretical knowledge and practical application in the context of data-driven energy forecasting.

The roadmap covers essential stages such as data preprocessing, regression model selection, performance evaluation, and hyperparameter tuning. It introduces and compares multiple supervised ML and DL techniques including k-NN, LR, Ridge (RDG) and Lasso regression (LAS), Decision Tree (DT), RF, SVR, Gradient Boosting Regressor (GBR), Bagging Regressor (BR), Voting Regressor (VR), MLP-based ANN, and basic DL architectures. These models are applied to a real-world solar dataset spanning up to 20 years of hourly observations, where the target variable is PV power output per square meter, and independent variables include atmospheric irradiance, solar altitude, azimuth, angle of incidence, wind speed, temperature, declination, and more. A supervised learning framework is employed, and optimal hyperparameter configurations are explored for each model.

The novelty of this study lies in its comprehensive and replicable comparative structure, combining mathematical clarity, implementation guidance, and performance analysis within a single platform. While not a systematic literature review, the work provides a tutorial-style investigation that illustrates how various models perform under comparable conditions using openly accessible tools such as scikit-learn, pandas, TensorFlow, and other Python libraries. The study also discusses classical optimization approaches like Grid Search (GSA) and GAs, helping readers understand their roles and limitations in practical AI workflows. By integrating model theory with implementation and evaluation, this article offers a valuable resource for researchers developing data-driven solutions in energy systems.

This study is organized as follows: Section 1 presents the Introduction, including the literature review and originality. Section 2 explains the mathematical equations and hyperparameters of ML and DL architectures. Section 3 also discusses data preprocessing and hyperparameter tuning that may affect model prediction performance, while introducing performance metrics related to regression. In Section 4, the prediction performances of the selected architectures are evaluated comparatively, while Section 5 discusses the model prediction performance and the results obtained, based on the publications considered in the Introduction. Finally, the conclusions and recommendations are presented.

2. Methodology

2.1. Data collection & preprocessing

2.1.1. Detailed dataset preparation pipeline

The 20-year solar power forecasting dataset was assembled from multiple authoritative sources to ensure comprehensive coverage of both meteorological and astronomical variables:

Data Sources and Variables:

1. NASA MERRA-2 Reanalysis (Temporal Coverage: 2003–2023, Spatial Resolution: 0.5° × 0.625°):

- Global Horizontal Irradiance (GHI): Derived from surface incoming shortwave flux with corrections for local terrain and atmospheric conditions
- Ambient Temperature (T2M): 2-meter air temperature [°C]
- Wind Speed (U10M, V10M): Combined 10-meter zonal and meridional components [m/s]
- Data acquired via NASA's Goddard Earth Sciences Data and Information Services Center

2. ***Solargis Satellite-Based Database (Temporal Coverage: 2003–2023, Spatial*** Resolution***: 250 m):***

- Out-of-atmosphere irradiance components
- Cloud modification factors derived from Meteosat s Generation imagery.
- Data validated against 15 + ground stations in the Mediterranean region (mean bias error (MBE) < 2 %)

3. ***CM SAF SARAH-2 Dataset (Temporal Coverage: 2003–2023, Temporal*** Resolution***: Hourly):***

- Surface solar radiation components with sub-hourly variability captured through 15-minute satellite acquisitions aggregated to hourly means.
- Quality flags indicating clear-sky conditions, partial cloud cover, and overcast periods.

4. Astronomical ***Calculations (Susdesign Solar Position Algorithm):***

- Solar Altitude (α)*:* Computed using NREL's solar position algorithm (SPA) with ±0.0003° accuracy.
- Solar Azimuth (ψ)*:* Cardinal direction of the sun from true north.
- Declination (δ): Earth's axial tilt contribution [-23.45°, + 23.45°].
- Hour Angle (ω): Angular displacement from solar noon.
- Angle of Incidence (θ): Angle between incoming solar radiation and panel surface normal.

Preprocessing Workflow:

Step 1: Temporal Alignment and Quality Control

- All data sources were aligned to coordinated universal time (UTC) timestamps rounded to the nearest hour.
- Missing values (<0.3 % of total samples) were flagged and excluded from analysis.
- Sensor malfunction periods (identified via MERRA-2 quality flags QA=3) were removed.
- Nighttime periods (GHI < 5 W/m^2) were retained to capture the full diurnal cycle.

Step 2: Feature Engineering

- Incident Solar Radiation*:* Calculated as GHI × cos(θ) to account for panel tilt (35° for Antalya).
- Clear-Sky Index (k_t)*:* Ratio of observed GHI to theoretical clear-sky GHI (Ineichen-Perez model).
- Temperature-Efficiency Adjustment*:* PV output corrected for cell temperature effects using the temperature coefficient (-0.45 %/°C for monocrystalline silicon).

Step 3: Normalization Strategy

- All features scaled to [0, 1] range using MinMaxScaler, fitted only on training data.
- Target variable (PV power output per m^2) normalized separately to preserve physical interpretation.
- Scaler parameters saved and applied identically to test data to prevent information leakage.

Step 4: Temporal Splitting for Model Validation

- Chronological ordering enforced: samples sorted by timestamp.
- Training set: First 122,640 samples (70 %, covering 2003–01–01–2017–01–15).
- Test set: Last 52,560 samples (30 %, covering 2017–01–16–2023–12–31).

Table 1
Comparison of PV output power forecasting performance.

Ref.	ML Techniques	Resolution	Dataset Size	Metrics			
				RMSE	*MAE*	R^2	*MAPE*
(Zazoum, 2022)	Matern 5/2 Gauss process regression	1-hr	1-yr	7.967	5.3025	0.98	-
(Ağbulut et al., 2020)	Support vector regressor (SVR)	1-hr	1 yr	0.7082	0.3357	0.9921	0.6238
(Zhou et al., 2020)	Similar day analysis & GA & Extreme learning machine (ELM)	1-hr	1.83 yr	21.07	0.2367	0.9225	-
(Markovics and Mayer, 2022)	Core ridge and MLP regression (MLPR)	15-min.	2 yr	52.1	36.5	-	-
(Mahmud et al., 2021)	Random forest (RF)	5-min.	1 yr	0.02	0.0098	0.9919	-
(Kabilan et al., 2021)	Neural network (NN)	5-min.	1-yr	4.42	3.00	0.8833	0.55
(Li et al., 2020)	CNN & LSTM	15 min. steps for 15–180 min. range	1 yr	2.095	1.028	-	-
(Theocharides et al., 2020)	ANN & k-nearest neighbors (k-NN) & linear regression (LR)	1-hr	1-yr	6.11	-	-	4.7
(Feng et al., 2020)	PSO & ELM	1-day	55-yr	0.11	1.23	-	-
(Mellit et al., 2021)	LSTM	1, 5, 30, and 60 min.	164,171 1-min samples	0.16	0.05	0.992	12.45
(Mishra et al., 2020)	Wavelet transform & LSTM	1-hr, 1-day, 15-days, 30 days, and 60 days	2-yr	0.0043	0.0365	0.95053	0.02914
(Khan et al., 2022)	Stacked ensemble learning algorithm	15-min and 1-hr	1-yr	0.74	0.47	0.98	-
(Elsaraiti and Merabet, 2022)	LSTM	30-min	1-yr	317.4	236.35	0.9381	0.275
(Akhter et al., 2022a)	Salp swarm algorithm & RNN & LSTM	5-min and 1-hr	4-yr	26.74	14.34	0.9961	-
(Jebli et al., 2021)	MLP-based ANN	30-min.	3-yr	0.08	0.03	0.9999	-
(Kumar et al., 2022)	ELMAN	5-min.	1-yr	0.298	0.186	0.95	-
(Li et al., 2020)	Wavelet packet decomposition & LSTM	5-min. and 1-hr	(1)+ (1)-yr	0.2357	0.0067	-	2.4002
(Munawar and Wang, 2020)	Principal component analysis (PCA) & RF & Extreme gradient boosting (XGBoost)	5-min.	3-month	2.49082	-	0.9994	-
(Agga et al., 2022)	CNN & LSTM	1-hr	1.25-yr	6.65	4.97	-	19.85
(Abdel-Basset et al., 2021)	PV-Net	5-min.	5-yr	0.8124	0.3987	-	6.614
(Luo et al., 2021)	Physically constrained LSTM	1-hr	2-yr	2.064	2.95	0.910	-
(Pan et al., 2020)	Improved ant colony algorithm & SVM	5-min.	Almost 2-yr	0.1868	0.1569	0.997	-
(Wang et al., 2020)	Partial daily pattern prediction & Time correlation modification & RNN & LSTM	15-min.	5-yr	6.29	2.78	0.9619	-
(Lin et al., 2020)	Cauchy mutation operator & Improved moth-flame optimization & SVM	30-min.	1-yr	0.0333	-	0.9962	2.8559
(Qu et al., 2021)	An attention-based long-term and short-term temporal neural network prediction model (ALSM)	5-min.	3-yr	6.21	3.95	0.9792	-
(Zhen et al., 2021)	Bidirectional improved LSTM optimized with a GA	5-min.	315,705 samples	0.438	0.242	0.983	-
(Zhang et al., 2020)	Improved variational mode decomposition (VMD) & Autoregressive integrated moving average & Improved DBN	5-min.	1-yr	7.54	4.84	-	0.67
(Chen and Chang, 2021)	LSTM based on Pearson feature selection	20-min.	1-yr	0.121	0.189	-	-
(Dairi et al., 2020)	DL with variational autoencoder	1-min. 15-min.	3-yr 1-yr	199.645	99.838	0.995	-
(Hossain and Mahmood, 2020)	LSTM	1-hr	6-yr	0.71	0.36	-	22.31
(Gutiérrez et al., 2021)	ANN	5-min.	6-month	86.466	8.409	-	-
(Zhen et al., 2020)	Convolutional autoencoder & K-means clustering algorithm	15-min.	25,000 samples	45.1149	30.0172	0.9874	-
(Zang et al., 2020)	ResNet & DenseNet	1-hr	47,600 samples	0.2846	0.152	0.990	-
(Ağbulut et al., 2021)	ANN	1-hr	2-yr	2.218	-0.0251	0.936	17.82
(Sharadga et al., 2020)	Bidirectional LSTM	15-min.	10-month	0.791	-	0.98	-
(Liu et al., 2020)	Improved chicken swarm optimization & ELM	5-min.	1-day	2.84	-	0.9984	0.85
(Wang et al., 2022)	VMD & Multi-objective intelligent optimization algorithm & CNN	1-hr	745, 1246, 1174, and 669 samples	141.80	112.06	0.98	-
(Akhter et al., 2022b)	RNN & LSTM	5-min.	4-yr	19.78	-	0.9943	-
(Barrera et al., 2020)	ANN with open data	5-min.	3-yr	0.20	0.161	-	-
(Lima et al., 2020)	MLP & Radial basis function & SVR & DL-based portfolio theory	1-hr 5-min.	10-yr 2-yr	-	-	-	4.52
(Narvaez et al., 2021)	RF	30-min.	12-yr	66	33	0.96	-
(Khan et al., 2020)	MLPR & SVR & CatBoost	1-hr	8.5-yr	21.7477	15.7275	-	-
(Kim et al., 2021)	Multi-scale LSTM	1-hr	1-yr	9.871	-	-	34.125
(Qu et al., 2021)	GRU pool	5-min.	3-yr	6.83	4.12	-	-

- No random shuffling applied to preserve temporal dependencies.

Dataset Statistics:

The final curated dataset consists of 175,200 hourly observations with 10 input predictors and one target variable (PV power output). A summary of the statistical properties of the main variables is presented in Table 2. Since the dataset contains a large number of observations and the descriptive statistics follow expected physical ranges, only the most relevant distributional indicators are reported.

The distribution statistics confirm the dataset's integrity and physical consistency. The strong skewness in power output (median = 0, 75th percentile = 0.27552) reflects the inherent diurnal cycle, where nighttime periods (power = 0) constitute approximately half of all observations. The extraterrestrial radiation shows the expected seasonal variation (mean = 683.64 W/m^2, std = 661.57 W/m^2), with values peaking near the theoretical maximum of 1415 W/m^2, consistent with annual solar geometry.

The solar altitude values range from 0° to 75.26° and exhibit a broad dispersion (mean = 16.11°, std = 21.25°), aligning well with Antalya's geographic latitude (36.9°N). Temperature spans from –11.79°C to 44.64°C, capturing both winter and summer extremes typical of the Mediterranean climate. Wind speed remains relatively moderate (mean = 3.73 m/s, std = 1.98 m/s), with occasional high-wind events reaching 15.64 m/s.

The normalized global solar radiation and incident solar radiation variables also follow expected solar patterns (mean = 0.18934 and 0.20773, respectively) and show strong correlation with the power output, supporting their inclusion as primary predictors.

The distribution of the angle of incidence reveals a clustering near 90° (median = 90°, 75th percentile = 90°), corresponding to nighttime and low-sun-angle periods when the PV module surface receives minimal direct irradiance. This behavior is characteristic of fixed-tilt PV systems and does not indicate any data anomalies.

Feature Correlation Analysis

A comprehensive correlation analysis was conducted to investigate the linear relationships between all predictor variables and PV power output. The full correlation matrix is presented in Fig. 1, and the strongest pairwise relationships obtained from the Python output are incorporated into the discussion. Incident solar irradiance shows an almost perfect linear association with PV power output (r = 0.99998), reaffirming its role as the dominant physical driver of PV energy generation. Global solar irradiance also exhibits a strong positive correlation (r = 0.966), in line with established PV performance theory, as global horizontal irradiance represents one of the primary determinants of DC power production.

Solar altitude demonstrates a substantial positive relationship with PV output (r = 0.852), capturing both diurnal and seasonal variations in the sun's apparent position. In contrast, the angle of incidence presents a strong negative correlation (r = –0.838), reflecting the geometric losses that occur when the incoming solar beam deviates from normal incidence on the module plane. Temperature displays a weak positive correlation (r = 0.331). Although PV efficiency conventionally decreases with increasing temperature, this sign reversal is likely attributable to its collinearity with irradiance, as higher temperatures tend to co-occur with periods of high solar input. Wind speed remains only marginally correlated with PV power output (r = 0.187), indicating that its influence in this dataset is minimal and largely indirect. Azimuth is essentially uncorrelated with PV output (r ≈ –0.01).

Fig. 1 provides a clear visual depiction of these relationships. The matrix reveals a pronounced cluster of highly inter-correlated irradiance-related variables, including incident solar irradiance, global solar irradiance, and solar altitude, all of which represent the primary physical mechanisms governing PV generation. The sharp negative correlation associated with the angle of incidence further emphasizes the geometric dependency inherent in PV systems. Meanwhile, environmental parameters such as temperature and wind speed form a weakly correlated subgroup, consistent with their secondary roles in short-term PV performance. Collectively, these observations validate theoretical expectations and offer a rigorous basis for the subsequent model selection and feature engineering stages of the study.

2.1.2. *Alternative choices and theory*

Preprocessing is used to transform feature vectors into a more suitable representation for downstream predictors. Preprocessing encompasses many task-specific methods, but here, a Python-based workflow will be followed. A common requirement for many ML predictors is to perform standardization on datasets. Specifically, by applying the standardization function on a dataset, data with zero mean and unit variance, following a Gaussian normal distribution, can be obtained. In this process, the distribution shapes in the dataset are ignored, and the mean of each feature is subtracted. The data is centered, and variable features are scaled by dividing by their standard deviations, ensuring that their variances are in the same order. If the variance of one feature is significantly larger than others, it can modify the objective function in its favor, preventing the proper learning of other features. To repeatedly apply the same operation and compute the mean and standard deviation of the training set, the Transformer API-based StandardScaler feature is used. Another perspective focuses on scaling features according to a range defined by the user. Transformations such as scaling within a range of 0–1 (MinMaxScaler) or scaling each feature by its maximum absolute value to a unit size (MaxAbsScaler) can serve this purpose. MinMaxScaler applies the rule $std.(\max. - \min.) + \min.$ to the dataset, while MaxAbsScaler scales the training data within the range of [-1,1] by dividing each feature by its largest maximum value. This approach ensures that sparse data maintains zero entries, and the negative impact of small standard deviations is reduced. When features are on different scales, it is recommended to use the MaxAbsScaler for scaling sparse data. If there are many outliers in the data, using the mean and variance for scaling can lead to incorrect scaling, so a more robust method that considers the data center and range, such as the RobustScaler, should be

Table 2
Summary of key predictors and target variable.

Variable	Min.	Max.	Mean	Std. Dev.	Median		
					25th Percentile	50th Percentile	75th Percentile
Extraterrestrial Radiation	0	1415	683.64	661.57	0	727	1349
Global Solar Radiation	0	1.12	0.18934	0.27385	0	0.01183	0.31661
Solar Altitude	0	75.26	16.11	21.25	0	0.21184	30.16
Azimuth	-172.67	173.46	0.70408	100.0	-89.60	1.22	90.46
Angle of Incidence	4.99	98.73	69.64	26.67	48.08	90	90
Solar Incidence Radiation	0	1.23	0.20773	0.30562	0	0.00185	0.34707
Declination	-23.44	23.44	0.39436	16.45	-15.91	0.765	16.72
Hour Angle	-197.85	154.82	-21.79	103.86	-109.68	-21.52	66.66
Wind Speed	0.05112	15.64	3.73	1.98	2.26	3.46	4.90
Temperature	-.11.79	44.64	18.88	8.92	13.20	19.77	25.74
Solar Power Output	0	0.98576	0.16461	0.2422	0	0.00147	0.27552

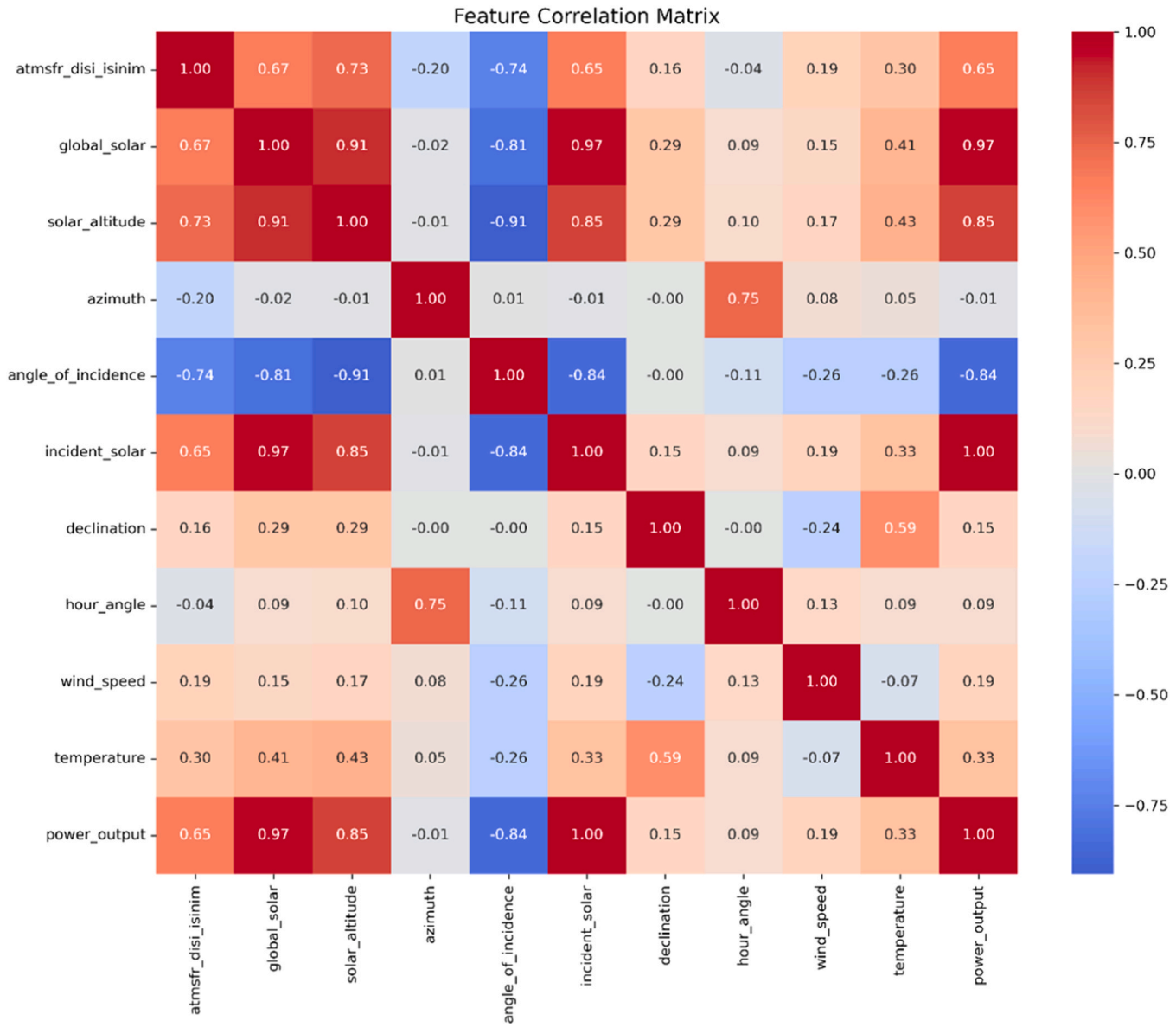


Fig. 1. Feature correlation matrix of the dataset.

used. Centering the kernel matrices is another preprocessing technique, while quantile transformation, where the value order across each feature is mapped to a cumulative distribution function and unusual distributions are smoothed, is an alternative. Since quantile transformations distort correlations and distances within and between features, power transformations can be preferred to achieve a distribution close to a Gaussian distribution. In quantile transformations, a uniform distribution between 0 and 1 is performed, while in power transformations, variance is stabilized and skewness is minimized through a Gaussian distribution matching. Another perspective involves the normalization process, which scales the samples according to a unit norm. In applications such as dot product or kernel methods that measure the similarity of sample pairs, data normalization is beneficial. For this purpose, $l1$, $l2$ (spatial signal preprocessing), or max norms can be used. When datasets include categorical attributes, these categorical features need to be encoded numerically. The OrdinalEncoder can be used to transform each categorical feature into an integer between 0 and the number of categories minus one. For categorical features, OneHotEncoder is recommended as it assigns binary values (1 or 0) based on whether a given condition is true or not for each category, as it expects continuous input and interprets categories in an ordered manner. However, if missing categorical features exist, even with the handle_unknown= 'infrequent_if_exist' command, the columns for these features may be turned to zero, causing infrequent categories. To avoid creating additional columns in the dataframe, the drop command should be used, ensuring only a single category is represented for each feature. Another situation involves encoding unordered categories, and for this, TargetEncoder, which uses the target mean for categorical features, stands out for nominal categories. To avoid causing a more expensive data processing process by expanding the feature space, it is particularly useful in high cardinality categorical features (such as postal codes or location-based features). Due to the negative impacts of non-informative target variables on time representations and false correlation events in downstream models, the cross fitting scheme is used, where the encoding learned for each fold is not saved for the attribute, utilizing the fit_transform function. Another perspective is related to dividing continuous features into discrete values or transforming them into nominal features only. This can be achieved using quantization or binning, a discretization preprocessing function. The output values are one-hot encoded into a sparse matrix and defined with known ranges for

four features, with bin edges calculated during the fitting process for each feature. Instead of counting features in specific bins for continuous data, the feature values are assigned to the bins, and the strategy varies depending on the task. If numerical features need to be thresholded to extract boolean values, the feature binarization function can be used. In particular, probabilistic predictors that fit multivariate input data and use a Bernoulli distribution have found applications in text processing communities and in normalized counts (such as term frequencies). If nonlinear features are desired for input data, possibilities based on polynomials, which are divided into PolynomialFeatures and SplineTransformer, can be considered. To add complexity to data sets, high-order features and interaction terms are evaluated in PolynomialFeatures, with a common application being the implicit use of kernel functions in support vector machines. In SplineTransformer, spline-based piecewise functions are created that parametrize polynomial degrees and knot positions for each feature. A low polynomial degree and a parsimoniously adapted number of knots ensure a flexible and robust operation. Unlike PolynomialFeatures, it does not exhibit oscillatory behavior at the boundaries and is a suitable option for extrapolation beyond the boundaries. It produces a feature matrix in a banded structure containing only positive elements in each row, sorted sequentially, and reduces the number of conditions. This makes it easily adaptable to stable algorithms applied to linear models. If the existing Python functions in data processing need to be converted into a transformer class, the FunctionTransformer preprocessing function can be used.

In this study, 20 years of out-of-atmosphere radiation data for Antalya province was obtained from Solargis, while global solar radiation, solar altitude, azimuth, angle of incidence, incident solar radiation, wind speed, and temperature data have been derived using NASA's MERRA reanalysis and the CM SAF SARAH. Declination and hour angle data were acquired from the Susdesign website (Susdesign, n.d.). After creating the dataset, data cleaning and preprocessing were performed using features such as Python pandas dataframe. To preserve temporal integrity and prevent data leakage, a time-aware validation strategy was employed. The dataset was chronologically ordered, with the first 70 % (122,640 samples) allocated to training and the final 30 % (52,560 samples) reserved for testing.This approach ensures that the model is evaluated on genuinely unseen future data. Furthermore, rolling-origin cross-validation with an expanding window was implemented during hyperparameter tuning. In this scheme, the training set grows incrementally while maintaining the temporal sequence, and validation is performed on the subsequent time window. This mimics realistic operational deployment where models are retrained periodically with accumulated historical data. All preprocessing transformations, including StandardScaler normalization, were fitted exclusively on the training partition and then applied to the test set. This strict separation prevents any statistical properties of future data from influencing model training, thereby eliminating look-ahead bias.

2.2. Machine and deep learning architectures

This section explains the theoretical foundations and mathematical models of ML and DL architectures, introducing the potential hyperparameters for each technique. Following this, topics such as data preprocessing processes, performance metrics, and hyperparameter tuning and optimization are discussed.

2.2.1. k-nearest neighbor (k-NN)

In k-NN regression analysis, which is a non-parametric (real-time) and supervised learning strategy that groups individual data points (Modaresi et al., 2018), the prediction about classification utilizes the average of the nearest k neighbors. For regression, continuous values are used in classification, and it is necessary to define distance to perform the operations. The input consists of the k nearest training examples from the dataset, while the output represents a class membership. Classification occurs through the majority vote of the neighbors, and the function is calculated locally, only approximating the result. Assigning weights to the contributions of the neighbors is a useful technique for both classification and regression analysis. This way, the contribution of closer neighbors to the average can be greater. A commonly used weighting approach assigns a weight of $1/d$ to each neighbor, where d is the distance to the neighbor. Instead of applying the training phase, it performs regression analysis by storing the training dataset and performing a classification or prediction in a single step. For this reason, it is considered a lazy learning model. Due to its simplicity, accuracy, and adaptability, it is one of the first algorithms learned in data science and is easy to apply. However, as the dataset grows, the model performance efficiency decreases because more memory is required. It will lead to high runtime and a costly computation process. In high-dimensional data inputs, it may also encounter dimensionality issues. For this reason, it is more prone to overfitting, which can be caused by the value of k. The number of neighbors to be considered in regression analysis is determined by the k parameter of the query point. Low values of k lead to high variance and low bias, while high values of k cause the opposite. The relevant parameter is closely related to the input data, particularly regarding outliers and noise. Typically, higher values of k reduce the impact of noise in both classification and regression.

The most commonly used distance definitions to determine the nearest neighbors of a given query point today are: Euclidean, Manhattan, Minkowski, and Hamming. In the Euclidean approach, a straight line between the query point and the other point is assumed. Another approach, known as taxi distance or city block distance, is Manhattan, which is visualized with a grid. It determines how to travel between addresses by measuring the absolute value between two points. The generalized form of the Euclidean and Manhattan distances is known as the Minkowski approach, where if p equals one, the Manhattan distance is obtained, and if p equals two, the Euclidean distance is obtained. The Hamming approach is used for Boolean or string vectors to identify the points where the vectors do not match. All distance definitions are shown in Eqs. (1)-(4). After calculating the distances from the labeled examples in the query instance, the labeled examples are sorted by increasing distance. The k points with the smallest distance from the query point are selected. The frequency of the category in which the k points are located is determined, and the category with the highest probability is returned.

$$d(x,y) = \sqrt{\sum_{j=1}^{n}\left(s_j - y_j\right)^2} \quad \textit{Euclidean distance} \tag{1}$$

$$d(x,y) = \sum_{j=1}^{m}\left|y_j - s_j\right| \quad \textit{Manhattan distance} \tag{2}$$

$$d(x,y) = \left(\sum_{j=1}^{n}\left|y_j - s_j\right|\right)^{1/p} \quad \textit{Minkowski distance} \tag{3}$$

$$D_H = \left(\sum_{j=1}^{k}\left|y_j - s_j\right|\right) \rightarrow \begin{cases} D = 0 & \text{if } s = y \\ D \neq 1 & \text{if } s \neq y \end{cases} \quad \textit{Hamming distance} \tag{4}$$

On the other hand, for the weight of each predicted data point, the inverse of the distance is considered as shown in Eq. (5), which can also be referred to as the distance approach.

$$w_{i,y_j} = \frac{1}{d_{i,y_j}} \tag{5}$$

However, in contrast to this, there is also the uniform method, which uses uniform weights, meaning all points in the neighborhood are equally weighted. In a dataset represented by m attributes, $x_j = \left(x_{j1}, x_{j2}, \ldots, x_{jm}\right)$, with output y_j and predicted s_j, the average of the k nearest neighbor outputs in regression is calculated as $\widehat{y} = \frac{1}{k}.\sum_{i=1}^{k} y_i(x)$

(Song et al., 2017). It should be noted that functional regression models map patterns to continuous labels, specifically to a subspace of $\mathbb{R}^d$. The regression problem is to predict the labels $x' \in \mathbb{R}^d$ for new patterns $x' \in \mathbb{R}^q$, based on a set of N labeled observations, i.e., labeled patterns $\{(x_1, y_1), \ldots, (x_N, y_N)\}$. Therefore, the statistical learning process must be carried out (Kramer, 2013). The BallTree technique is one such method. In multidimensional search operations, it uses binary trees to create a subset of data points enclosed in a hyper-sphere at each node. It recursively selects the dimensions by dividing the data into two subsets and creating a binary tree between them. Another approach is the KDTree strategy. Unlike BallTree, it recursively divides the data along the axes and groups the data into binary subsets at each level, based on the median of the chosen dimension. The tree is constructed with each node representing a region in the multidimensional space. It is particularly important to evaluate the leaf size for these two tree types. The relevant parameter affects the query speed and the memory required to store the tree, with the optimal value depending on the nature of the problem. In contrast, the brute force approach calculates the squared distances from the feature vector of each query to the reference feature vectors for each query. The closest objects from the query feature vector to the training dataset's query feature vectors are selected. The final predicted value for each data point (s_i') is determined by Eq. (6) (Fan et al., 2019).

$$s_i' = \frac{\sum_{j=1}^{k} \left(w_{i,y_j} . s_{i,y_j} \right)}{\sum_{j=1}^{k} w_{i,y_j}} \tag{6}$$

Ultimately, for the prediction of PV power output using the k-NN regression method, hyperparameters such as the number of neighbors, weights option, statistical learning algorithm, leaf size, p value, and distance metric need to be optimized to achieve high performance metrics. To this end, GA and GSA have been employed, and the optimal hyperparameter space and performance metrics are discussed in Section 3.

2.2.2. Linear regression (LR)

The simplest way to obtain the value of one variable (dependent, predicted) from another (independent) variable is LR. It is a statistical approach that expresses the scalar response in terms of explanatory variables and determines the linear relationship. In the case of multiple explanatory variables, the analysis is referred to as multiple LR, while for a single explanatory variable, this is termed simple LR. One of the main reasons it is one of the first regression analyses used in practice is that it adapts more easily compared to models with nonlinear relationships. Thus, the coefficients of one or more independent variables that best predict the dependent variable can be easily determined through regression analysis. It finds the optimal line or surface that minimizes inconsistencies between predicted and actual outputs. The first method that comes to mind for this task is the ordinary least squares (OLS) method, which typically uses the R^2 metric (James et al., 2023). The OLS regression model is formulated with respect to the explanatory variable x_i, as shown in Eq. (7).

$$y_i = \alpha + \beta . x_i + \varepsilon_i \tag{7}$$

This model is used to calculate the loss function, which includes the sum of squared residuals when only one variable is considered. Here, y_i represents the dependent variable, α is the intercept of the model, x_i is the i^{th} explanatory variable of the model, and ε is the random error, evaluated according to variance with an expected value of 0. OLS minimizes the sum of squared differences between observed and predicted values, consider the issue of multicollinearity (Modaresi et al., 2018). This approach calculates the coefficient vector, β, to serve this purpose (Xu et al., 2023). The subcalculations used to compute the relevant equation are provided in Eqs. (8) and (9).

$$\widehat{\beta} = \frac{\sum (x_i . y_i) - \frac{\sum (x_i) . \sum (y_i)}{n}}{\sum (x_i^2) - \frac{\sum (x_i^2)}{n}} = \frac{Cov\,[x, y]}{Var\,[x]} \tag{8}$$

$$\widehat{\alpha} = \overline{y} - \widehat{\beta} . \overline{x} \tag{9}$$

Here, $\widehat{\beta}$ is the coefficient vector of the least squares hyperplane. $\{x_i, y_i\}_{i=1}^{n}$ refers to the dataset with n observations, and y_i represents the scalar response. However, real-world applications often rely on multivariate equations. In this case, Eq. (10) is used.

$$S(b) = \sum_{i=1}^{n} \left(y_i - x_i^T b \right)^2 = (y - Xb)^T . (y - Xb) \tag{10}$$

Here, the matrix x_i contains all the ones corresponding to the intercept of the response variable equation. y_i is the actual response variable, and $x_i^T b$ is the predicted value of the explanatory variable, meaning that $S(b)$ refers to the residual sum of squares (RSS). The β value obtained by minimizing $S(b)$ is calculated in Eq. (11) through the optimization process.

$$\widehat{\beta} = argmin_{b \in \mathbb{R}^p} S(b) = \left(X^T X \right)^{-1} . X^T y \tag{11}$$

Here, p is equal to one more than the number of regressors. The error term, which is typically neglected, must be predicted since it has a constant variance, and this task is handled by Eq. (12). Beyond simple and multivariate LR, there is also polynomial regression (Maulud and Abdulazeez, 2020). The relationship between dependent and independent variables is based on a polynomial model of degree n. The polynomial regression formula can be found in Eq. (13), where h represents the polynomial degree.

$$S^2 = \frac{\sum_{i=1}^{n} (y_i - \widehat{y}_i)^2}{n - p - 1} \tag{12}$$

$$y = \beta_0 + \beta_1 . x + \beta_2 . x^2 + \ldots + \beta_h . x^h + \varepsilon \tag{13}$$

Ultimately, it has a number of advantages, such as its simplicity and the ease of interpreting the predictions. It is an established statistical procedure that guarantees fast training. It manages reality based on the collected data, without relying on experience or intuition. It offers the ability to transform large amounts of data into actionable information. Since it optimizes its parameters through OLS, there is no need for hyperparameter tuning.

2.2.3. Ridge (RDG) and Lasso (LAS) regression

Another approach used in multiple regression models is RDG. The main issue with LR in models with numerous parameters is the problem of multicollinearity. RDG is considered as a possible solution to the OLS uncertainty in models with many parameters and high weights. It has improved efficiency in parameter estimation in exchange for tolerable bias. It corrects overfitting caused by the training data and is more commonly known as $\ell 2$ regularization.

Multicollinearity refers to the situation where two or more predictors have nearly a linear relationship. Methods such as increasing the sample size or reducing the number of independent variables can be solutions to the multicollinearity problem. RDG is typically the first approach considered to eliminate this issue. The coefficients that are not explicitly stated in the training data are calculated using OLS. The RSS is computed, which allows the determination of how well the LR model fits the training data. A value of zero for this indicates that the dependent variable is predicted very accurately, though it may lead to overfitting in small training datasets. The high correlation between two or more variables increases sensitivity to small changes in the input data. This causes stability problems by compromising the generalization ability of OLS. Considering potential related predictors, OLS can be replaced with

RDG. By adding a regularization or penalty term, which includes the sum of the squares of the model coefficients, high-value coefficients are corrected (see Eqs. (14) and (15)).

$$L2 = \|\widehat{\beta}\|^2 = \widehat{\beta}_1^2 + \widehat{\beta}_2^2 + \ldots + \widehat{\beta}_p^2 \tag{14}$$

$$RSS_{L2} = \sum_{i=1}^{n} \left(y_i - x_i^T b\right)^2 + \lambda . \sum_{j=1}^{p} \widehat{\beta}_j^2 \tag{15}$$

Here, $\widehat{\beta}$ is known as the OLS matrix coefficient estimator, and after calculating the $\ell 2$ penalty term, it is included in the *RSS* (or $S(b)$) function using the hyperparameter λ. This models the effect on the model and nullifies high coefficients by reducing all coefficient values (though not proportionally). More precisely, it computes the reduced new regression coefficients and prevents overfitting. As λ increases, high-value coefficients are reduced more, resulting in greater penalization. Consequently, the model's bias increases while its variance decreases (Ahrens et al., 2020). In contrast, instead of overfitting, underfitting occurs. If λ equals zero, it reverts to the OLS method. The $\widehat{\beta}$ parameter is estimated by applying in Eq. (16); and the RDG regression model's pending coefficient is calculated using Eq. (17) (Fan et al., 2021).

$$\frac{\partial L(\widehat{\beta})}{\partial \widehat{\beta}} = 0 = \frac{\partial \left[(y - X\widehat{\beta})^T . (y - X\widehat{\beta}) + \gamma . (\widehat{\beta})^T . \widehat{\beta}\right]}{\partial \widehat{\beta}} \tag{16}$$

$$\widehat{\beta} = \left(X^T . X + \lambda . I\right)^{-1} . X^T . y \tag{17}$$

The penalized sum of squares is then minimized (Eq. (18)) (Wang et al., 2018). The amount of shrinkage can be controlled as long as the complexity parameter α is greater than zero. As the value of α increases, shrinkage becomes more pronounced, making the model more resistant to nonlinearity.

$$\min_w \|Xw - y\|_2^2 + \alpha \|w\|_2^2 \tag{18}$$

The average difference between the predicted and actual values is measured by bias, while variance quantifies the difference between predictions across different actions of the model. As either bias or variance increases, the accuracy of the model decreases, creating a trade-off when trying to achieve high accuracy by minimizing both parameters. RDG adds bias to the model to reduce variance. As a result, while the model may be less accurate on training data, it achieves higher accuracy on test data. RDG differentiates from LAS because, although it reduces the coefficients and feature weights towards zero, it does not shrink them to exactly zero. In LAS, on the other hand, the predictor's effect on the model is eliminated as the coefficients are reduced to zero, and feature selection is performed. In RDG, due to severe multicollinearity, the predictor effects cannot be separated, making LAS more prominent. LAS applies $\ell 1$ regularization and seeks a balance between simplicity and accuracy in the model. It eliminates irrelevant or unnecessary variables and is a regression method that regularizes the $\hat{\beta}$ parameter under the sparse assumption. Data values shrink towards a central point as the mean of the values. It encourages sparser models with fewer parameters. By considering the absolute value of the coefficients and the adjustment parameter λ, the $\ell 1$ penalty is included in the RSS function (see Eqs. (19) and (20)).

$$L1 = \lambda . \left(|\widehat{\beta}_1| + |\widehat{\beta}_2| + \ldots + |\widehat{\beta}_p|\right) \tag{19}$$

$$RSS_{L1} = \sum_{i=1}^{n} \left(y_i - x_i^T b\right)^2 + L1 \tag{20}$$

When λ is sufficiently large, some coefficients are zeroed out, so coordinate descent is typically used to predict the coefficients and fit the model. On the other hand, if p predictors and n sample size satisfy $s \ll n$, a subset is included in the model, and s, the sparsity index, is determined by Eq. (21) (Ahrens et al., 2020).

$$s = \sum_{j=1}^{p} 1\{\widehat{\beta}_j \neq 0\} = \|\widehat{\beta}\|_0 \tag{21}$$

In LAS, a basic regression analysis for compressed sensing, the formulation to minimize RSS is expressed by Eq. (22). In $\alpha \|w\|_1$, α is any constant, and $\|w\|_1$ is the $\ell 1$ norm of the coefficient vector.

$$\min_w \frac{1}{2n_{samples}} \|Xw - y\|_2^2 + \alpha \|w\|_1 \tag{22}$$

To manipulate OLS, optimization is essential for RDG and LAS, which have hyperparameters. In the case of RDG, the α hyperparameter and the evaluation of maximum iterations and solver options are crucial. Similarly, in LAS, instead of a solver, a selection criterion is used. There are several alternatives for solving RDG regression, such as singular value decomposition (SVD), Cholesky, sparse conjugate (sprc), regularized least squares (RLS), stochastic average gradient descent (SAGA), and limited-memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS-B) algorithms. The stochastic gradient descent (SGD) method, which is determined by the direction of the highest variance (V^T), works with matrix decomposition. It involves the pointwise product of three additional matrices with unique characteristics, ultimately reaching the compressed representation of the original matrix. The decomposed dataset A follows a process based on the intensity of directions (Σ), orthogonal unit vectors (V), and the example length in the dataset (U) (see Eq. (23)) (Storrs, n.d.). In Cholesky decomposition, the Hermitian positive-definite A matrix is considered, and it follows a procedure such as $A = [L].[L]^T$. Here, L is a lower triangular matrix containing real and positive diagonal entries. The final result is computed using Eq. (24).

$$A = U.\Sigma.V^T \tag{23}$$

$$L_{j,j} = \sqrt{A_{j,j} - \sum_{k=0}^{j-1} \left(L_{j,k}\right)^2} \tag{24}$$

The processes for algorithms such as sprc, RLS, SAGA, and L-BFGS-B are detailed, and the necessary mathematical formulations can be found on Wikipedia. On the other hand, in LAS, the user is provided with random and cyclic options. In random mode, unlike sequential loops, a random coefficient is updated in each iteration. In contrast, the cyclic method is followed in sequential loops. The optimal results regarding the hyperparameter tuning and optimization metrics for predicting PV output power are presented in the Section 3.

2.2.4. Decision tree (DT)

DT are a non-parametric supervised learning algorithm. In the DT method with a decision support hierarchy, the option with the highest probability of reaching the goal is selected. Each internal node corresponds to various tests, while each branch represents the results of those tests. Once all the attributes are calculated, each leaf node represents the class label of the node. The tree structure consists of a root node, branches, internal nodes, and leaf nodes. Starting from the root node, which has no branches, the hierarchy continues with decision nodes (internal nodes) and branches extending from the root. Internal nodes form leaf nodes, which are members of homogeneous subsets, and terminal nodes represent possible outcomes in the dataset. This journey from root to leaf is commonly associated with classification, but it is also suitable for regression analysis. Here, a divide-and-conquer strategy with a greedy search is applied. The goal is to build a model that predicts the value of the target variable by learning simple DT rules extracted from the data attributes. This process is iteratively repeated in a hierarchical order until regression is finalized. The best attribute at each node can be determined using methods frequently employed in datasets such as Gini impurity and entropy. In this way, the quality of each test condition is measured, and the best information gain can be obtained.

The entropy method, which measures the impurity of sample values, uses Eq. (25).

$$S = -\sum_{c\epsilon C} p(c).\log_2 p(c) \tag{25}$$

Here, S represents the entropy calculated for the dataset, and c represents the classes in the dataset. The proportion of data belonging to class c in relation to the total data is calculated as $p(c)$. If all examples in the dataset belong to one class, the entropy is zero, and if half of the examples belong to one class, the entropy equals one. The attribute that results in the lowest entropy is selected for the optimal DT. Information gain refers to the difference in entropy before and after a split for a particular attribute, and it is calculated cooperatively with entropy using Eq. (26).

$$Gain(S,A) = S - \sum_{v\epsilon Values(A)} \frac{|S_v|}{|S|}.S_v \tag{26}$$

Here, A represents a specific attribute or class label, and $|S_v|/|S|$ indicates the proportion of values in S_v to the total dataset. The Gini impurity measures the probability of mislabeling or misclassifying a random data point according to the class distribution in the dataset. If the dataset belongs to one class, the Gini coefficient is zero, and it is calculated using Eq. (27).

$$Gini\ Impurity = 1 - \sum_i (p_i)^2 \tag{27}$$

The selection of attributes shapes the DT structure. However, a complex DT structure is influential in determining the final classes, and as the tree grows, certain subtrees receive fewer data. This can lead to overfitting, but if DTs are necessary, complexity should be added to the structure. Problems such as complexity and overfitting can be reduced through pruning, which removes branches of lower significance. Pruning is evaluated in two ways: pre-pruning, which stops tree growth when missing data occurs, and post-pruning, which removes subtrees after the tree formation. Pruning is typically carried out by adjusting the minimum number of samples required at leaf nodes or setting the maximum depth.

Given training vectors $x_i \in R^n$ and label vectors $y \in R^l$, the DT recursively partitions the feature space, grouping together examples with similar target values. Assuming that the data at node m is represented by the set Q_m containing n_m examples, for each candidate split $\theta = (j, t_m)$, consisting of feature j and threshold t_m, the data is divided into two subsets: $Q_m^{left}(\theta)$ and $Q_m^{right}(\theta)$ (see Eqs. (28) and (29)). The quality of the candidate split at node m is computed using an impurity or loss function, which depends on the regression (see Eq. (30)). The appropriate parameters that minimize impurity are determined by Eq. (31).

$$Q_m^{left}(\theta) = \{(x,y)|x_j \le t_m\} \tag{28}$$

$$Q_m^{right}(\theta) = Q_m \backslash Q_m^{left}(\theta) \tag{29}$$

$$G(Q_m,\theta) = \frac{n_m^{left}}{n_m}.H\left(Q_m^{left}(\theta)\right) + \frac{n_m^{right}}{n_m}.H\left(Q_m^{right}(\theta)\right) \tag{30}$$

$$\theta^* = -argmin_\theta G(Q_m,\theta) \tag{31}$$

The process of splitting continues until the maximum allowed depth is reached (either $n_m < \min_{samples}$ or $n_m = 1$). If the target is continuous, future split positions are determined by minimizing criteria such as half Poisson deviation, MAE or $\ell 1$ error, and mean-squared error (MSE) or $\ell 2$ error. If the target is a count or frequency, half Poisson is generally selected as the criterion. After the calculations, the target values of all examples in the subspace of the input space are compared. If the target or input values of the examples are identical, if the tree depth exceeds the predefined maximum limit, or if the number of examples is smaller than the minimum leaf size limit, the main nodes are obtained in the iterative process (Ao et al., 2019).

In the presence of missing information, it is an effective method for making online decisions without recall. Due to its use of Boolean logic and visual representations, it is easy to understand and interpret, and valuable information can be extracted even from a small amount of data by considering probabilities and costs. It does not require data normalization, creation of dummy variables, or removal of missing values, and it supports missing values. The actions of multiple decision-makers, which do not require data preprocessing, can be considered in the relevant regression analysis; however, due to high variance, small changes in the data can cause unstable DT structures. It cannot generalize well to new data and its training is more costly compared to other algorithms. The cost is logarithmically related to the number of data points used for training. Since it is the simplest among tree schemes, its prediction accuracy is lower compared to others. Due to value uncertainty and associated outcomes, there can be computational complexity. Additionally, the predictions of DTs may not be smooth and continuous. Heuristic algorithms are needed to make the best decisions at each node (Jumin et al., 2021). It does not guarantee returning the globally best DT and may struggle to learn exclusive OR, parity, or multiplexing problems.

In this study, hyperparameter tuning for criterion, splitter, maximum depth, minimum sample split, and leaf has been considered. As criterion, alternatives such as "squared_error", "friedman_mse", "absolute_error", and "poisson" are evaluated. These represent the function used to measure the quality of the split. Among feature criteria, the "squared_error" approach, which is equivalent to variance reduction, uses the mean of each terminal node and minimizes $\ell 2$ loss. The "friedman_mse" method, integrating Friedman improvement score and mean squared error for potential splits, is another strategy. "absolute_error", which minimizes $\ell 1$ loss and absolute error based on the medians of each terminal node, offers a different perspective. "poisson", which determines splits based on reduction in Poisson deviation, is the final functional approach. The splitter, which is used to choose the split at each node, has two alternatives: best and random. Other hyperparameters utilize the numerical data defined and selected later. For detailed hyperparameter definitions, the Python sklearn library can be consulted.

2.2.5. Random forest (RF)

RF is a collaborative learning algorithm that uses the outputs of multiple DTs to reach a single result. It is easy to use, and due to its flexibility, it can perform regression analysis. The average or mean prediction of individual trees is used in regression analysis. In DTs, a series of questions are asked at the decision nodes of the tree to split the data, and the final decision is made at the leaf nodes. The classification and regression tree algorithm is frequently used for splitting the data into subsets. As explained in DTs, metrics such as Gini impurity, information gain, and entropy are valid here as well. However, due to the formation of a community of multiple DTs and the lack of correlation between individual trees, bias and overfitting problems are reduced, leading to more accurate predictions. In RFs, both bagging and feature bagging, also known as random subspace method, are used to introduce randomness. This feature distinguishes it from DTs, as it does not use the entire feature space while creating and selecting a subset of features with low correlation.

There are three hyperparameters considered before training in the RF algorithm to solve regression problems: node size, number of trees, and the number of sampled features. In this community of DTs, data samples taken from the bootstrap dataset are used to create the trees. A portion of the training samples is then considered as test data, which are referred to as out-of-bag (OOB) samples. Through feature bagging, additional random examples are included in the dataset to reduce the correlation between DTs, and the average of individual DT is used for regression tasks. The selected portion of the data is then used in cross-validation, and prediction operations continue. On the other hand, in RFs, each tree grows with examples from the bootstrap of the original

data, and a randomly selected set of m variables is searched for each tree. Data inconsistency is measured based on the placement in the same leaf (final subspace). A variable known as $s(i,j)$, which expresses the similarity between i and j, is a measure of when two data points are in the same leaf. A variable related to the similarity matrix of the random, symmetric, and positive forest can be used to obtain a non-similar matrix transformation (see Eq. (32)) (Shabani et al., 2020).

$$d(i,j) = \sqrt{1 - s(i,j)} \tag{32}$$

In this bagging-based structure, training is performed on a bootstrap sample (*Db*) taken from the original dataset (D). A random sample of k samples is taken from the N-sample dataset to create *Db*. After sample selection, the data returns to the pool, and a new sample is selected. After several iterations, *Db* should contain approximately 2/3 of the samples from D (Čeh et al., 2018). Furthermore, through splitting, randomization is incorporated into the selection of data nodes, improving the classification/regression performance of a single DT (Dewi and Chen, 2019). The number of trees determines how many regression trees are grown based on the bootstrap sample derived from the initial training dataset. A final forest regression model is created for *k* regression trees, where each node performs binary testing based on the selected predictor variable. A variable is selected to minimize the RSS for the samples flowing from left to right. By explaining the dataset and threshold value, the entire computation process is detailed from Eqs. (33)-(38).

$$\{(X,Y)|X = (x_1, x_2, \ldots, x_n), Y = f(x)\} \quad \rightarrow x_i > t \tag{33}$$

$$L = \{(X,Y)|x_i > t, Y = f(x)\} \tag{34}$$

$$R = \{(X,Y)|x_i \leq t, Y = f(x)\} \tag{35}$$

$$Y_{left} = \frac{1}{|L|} \cdot \sum_{(X,Y)\in L} Y \tag{36}$$

$$Y_{right} = \frac{1}{|R|} \cdot \sum_{(X,Y)\in R} Y \tag{37}$$

$$RSS = \sum_{(X,Y)\in L} \left(Y - Y_{left}\right)^2 + \sum_{(X,Y)\in R} \left(Y - Y_{right}\right)^2 \tag{38}$$

Another perspective focuses on measuring the model's accuracy. If the probability of a sample not being selected is expressed by $P = ((N-1)/N)^N$, as N approaches infinity, P approaches $\frac{1}{e} \approx 0.368$. Since 36.8 % of the samples are not selected, this part of the sample is referred to as the OOB data. The model's accuracy can be evaluated by calculating the OOB error (Liu et al., 2020). The model's prediction performance can be assessed using the OOB mean squared error MSE_{OOB} and the coefficient of determination (R^2_{RF}) metrics (see Eqs. (39) and (40)) (Zhao and Wu, 2018).

$$MSE_{OOB} = \frac{\sum_{i=1}^{n} (y_i - \widehat{y}_i)^2}{n} \tag{39}$$

$$R^2_{RF} = 1 - \frac{MSE_{OOB}}{\widehat{\sigma}^2_Y} \tag{40}$$

Here, n represents the total number of OOB samples; y_i represents the observed output value; $\widehat{y}_i$ is the predicted output obtained from the regression model; and $\widehat{\sigma}^2_Y$ is the predicted variance of the OOB output.

After the explanation of the computational mathematical model, it is important to understand the advantages and disadvantages. Specifically, the average of unrelated trees reduces variance and prediction error through RF, thereby decreasing the risk of overfitting. It is flexible because it maintains accuracy even with missing data and can be adapted to various strategies based on regression and classification. The importance of variables can easily be determined by excluding variables using Gini importance and mean decrease in impurity coefficients, and assessing how much the model accuracy decreases. The average decrease in accuracy measures the reduction in model performance caused by permuting feature values and is therefore used to evaluate feature importance. However, it is also important to be aware of its disadvantages. Although it can handle large datasets, it is a time-consuming process. Additionally, it requires more resources and interpreting a forest is more complex than interpreting a single DT. It may not perform well in fields where the information is valuable, as only a small portion of numerous features is used. This problem can be addressed by eliminating noisy features, using weighted random sampling at each node of every tree, and assigning higher weights to trees with better accuracy.

In this study, the hyperparameters that need to be adjusted include the number of estimators, criterion, maximum depth, minimum sample split and leaf, minimum leaf weight fraction, maximum features, minimum impurity decrease, verbose, ccp_alpha, and maximum samples. The methods discussed and explained in the DT are considered for the criterion, while the number of features to be considered when searching for the best split, i.e., maximum features, is evaluated with alternatives such as "sqrt", "log2", and None. The complexity parameter ccp_alpha is used for minimal cost-complexity pruning and is reflected in the model. The subtree with the largest cost complexity smaller than ccp_alpha is selected. The decision to split is made with a decrease equal to or greater than this value using minimum impurity decrease. For other hyperparameters, the selected numerical values will be explained later. For detailed hyperparameter definitions, the Python sklearn library can be examined.

2.2.6. Gradient boosting regressor (GBR)

GBR, a functional ML technique, uses residuals of words instead of typical residuals in traditional boosting. This technique emerged from the idea that optimization of the cost function for boosting can be interpreted. It is based on the principle of boosting. Subsequently, boosting algorithms shifted toward the iterative functional gradient descent approach. By iteratively selecting weak hypotheses pointing in the negative gradient direction, the cost function is optimized over the function space (Fafalios et al., 2020). In this method, which is constructed step-by-step, the differentiable loss function that can be arbitrarily considered is optimized, ensuring the generalization ability of the model. Like DTs, in boosting tree algorithms, a weak learner is generated at each step. The newly generated weak learners are weighted and accumulated in the model. The weak learners are iterated until reaching the minimum error and a strong learner is obtained (Xu et al., 2023). If, at each step, a weak learner is produced based on the gradient direction of the loss function, the gradient boosting goal is achieved.

Particularly in regression problems, the basic architecture of gradient boosting is as shown in Fig. 2.

In each iteration, the weak learner is assigned a weight indicating the prediction error rate, and each training example is re-weighted as it is misclassified. The regression tree is fitted according to the difference between the observed response in each step and the collective prediction of all previous learners (Dabiri and Abbas, 2018).

Considering the response (y_i) and explanatory (x_i) vector variables in the training dataset, along with N examples, the expected value of the loss function ($L(y, F(x))$) should be minimized (see Eq. (41)).

$$F^* = arg_F \min E_{y,x} L(y, F(x)) = arg_F \min E_{y,x}[y - F(x)]^2 \tag{41}$$

The goal here is to learn a model $F^*(x)$, which is the best approximation of the true function $F(x)$ that maps x to y. Although the loss function may vary, the least-square algorithm can be utilized to minimize the squared-error loss function. The function $F(x)$ is determined by considering the weighted sum of weak learners (see Eq. (42)).

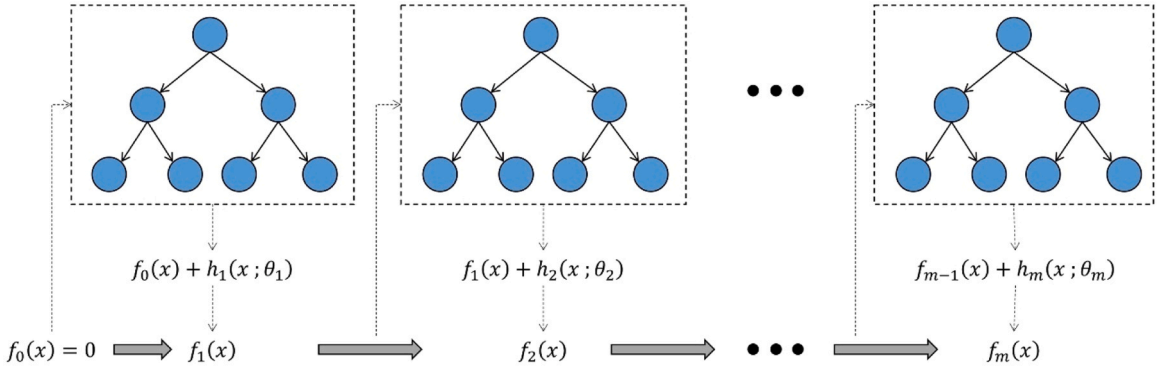


Fig. 2. Basic architecture of GBR tree.

$$F(x;\{\beta_m, a_m\}_1^M) = \sum_{m=1}^{M} \beta_m . h(x; a_m) \tag{42}$$

Here, M represents the number of weak learners, and β is the weight corresponding to each learner. The small regression tree is defined by a finite set of parameters a_m, including split variables, split locations, and leaf nodes, denoted as $h(x; a_m)$. In the final iteration, when $F_m(x)$ is equal to $F(x)$, a contributory model is obtained as shown in Eq. (43).

$$F_m(x) = F_{m-1}(x) + \beta_m . h(x; a_m) \tag{43}$$

For better contribution, the flawed model from the previous step is added to $F_{m-1}(x)$. Here, the reduction of the interaction between the DT weight β_m and $h(x; a_m)$ is the primary goal (see Eq. (44)).

$$[h(x; a_m), \beta_m] = arg_{h,\beta} \min \sum_{i=1}^{N} L(y_i, F_{m-1}(x_i) + \beta . h(x_i; a_m)) \tag{44}$$

To make the problem easier to solve, the gradient descent algorithm is utilized. For gradient descent, $h(x_i; a_m)$ is trained using the training set $\{x_i, g_m(x_i)\}_i^N$ (see Eq. (45)), where $g_m(x_i)$ is the negative gradient of the loss function calculated with the new equation. The step size is determined by solving the one-dimensional optimization problem in Eq. (46). For details, refer to (Dabiri and Abbas, 2018). The loss function is expressed as in Eq. (47), where the residual of the model fit for the minimum value of the previous loss function is calculated by Eq. (48) and used to approximate the current residuals.

$$g_m(x_i) = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} = [y_i - F_{m-1}(x_i)] \tag{45}$$

$$\beta_m = \arg_\beta \min \sum_{i=1}^{N} L(y_i, F_{m-1}(x_i) + \beta . h(x_i; a_m)) \tag{46}$$

$$L(y_i, f_m(x_i)) = [y_i - f_{m-1}(x_i) - h_m(x_i; \theta_m)]^2 = [r_{mi} - h_m(x_i; \theta_m)]^2 \tag{47}$$

$$r_{mi} = -\left[\frac{\partial L(y_i, f_{m-1}(x_i))}{\partial f_{m-1}(x_i)}\right] \tag{48}$$

Since overfitting may occur due to the close fitting of the training set, regularization techniques are employed. In regularization, as the number of gradient boosting iterations or the number of trees in the model, known as M, increases, the errors on the training set decrease, leading to the risk of overfitting. Therefore, optimization is necessary, and it is selected by monitoring the prediction error on the validation dataset. The depth of the trees is another regularization parameter. As the depth increases, the model's fit to the training data improves, and optimization is required to avoid overfitting issues. Another approach in regularization is focusing on the minimum number of observations. By limiting the minimum number of observations in terminal nodes, splits that lead to fewer than this number of observations are ignored, reducing variance. Penalizing the complexity of trees is another important aspect. The number of leaves in the learned trees determines the model complexity. Branches that cannot reduce the loss to a threshold are pruned through the collaborative optimization of the loss function and model complexity. An $\ell 2$ penalty can also be added to the leaf values to eliminate overfitting.

On the other hand, while a simple DT is a weak learner, applying gradient boosting is more effective in performance than RFs. Although it can increase accuracy in regression, it becomes more difficult to interpret and understand. Additionally, due to its high computational demand, it leads to an increase in cost. To improve performance and interpretability, model compression techniques, such as extreme gradient boosting (XGBoost considered in this study), which transforms the model into a "re-born" DT, can be used. Particularly for XGBoost, hyperparameter optimization is crucial. In this study, the hyperparameter space considered includes loss, learning rate (for gradient descent), number of estimators, criterion, minimum sample split and leaf, minimum weight fraction in the leaf, maximum depth, minimum impurity decrease, maximum features, alpha, maximum leaf nodes, validation fraction, tolerance, ccp_alpha, iteration count until no change, and tolerance. The options for the loss function are 'squared_error', 'absolute_error', 'huber', and 'quantile'. For regression, 'squared_error' and 'absolute_error' are preferred for squared and absolute errors, respectively. 'Huber' is a combination of both errors, while the alpha coefficient is suggested for the error in quantile regression. As for the criterion, 'friedman_mse' and 'squared_error' are considered, while for maximum features, 'sqrt' and 'log2' are considered. For other hyperparameters, numerical values chosen and explained later have been used. For detailed hyperparameter definitions, the Python sklearn library can be referred to.

2.2.7. *Bagging regressor (BR)*

In the technique known as bootstrap aggregation (bagging), variance is reduced in noisy datasets. It is a meta-algorithm in ML techniques that improves stability and accuracy and is commonly associated with its applicability to DTs, although it is useful with any method. Based on random selections of data samples with replacement from the training set, weak models are trained independently. In random selection, subalgorithms such as pasting, bagging (which is considered here), random subspace, and patches can be used (Sharafati et al., 2021). When random subsets of the dataset are drawn as random subsets of samples, the term 'Pasting' is used. When samples are drawn with replacement, the term 'Bagging' applies, and when random subsets of features are drawn, the term 'Random Subspaces' is used. Another method involves building base predictors on both sample and feature subsets, known as 'Random Patches.' On the other hand, averaging predictions is more commonly considered in regression tasks. This ensemble approach is based on the concept of the "wisdom of crowds." A combination of weak learners or bagging or boosting approaches can address high variance or

bias effects in weak learner models. In bagging and boosting methods, which differ based on variations in the training process, bagging contributes to the ensemble by training weak learners in parallel, while boosting contributes by training weak learners sequentially. However, both methods share a common characteristic of increasing the weights of misclassified data from the previous model. Since both methods are referred to as bootstrap aggregation, both concepts should be well understood. In bagging techniques, the ability of bootstrap sampling is utilized. The related resampling method involves the random selection and replacement of data points, creating different subsets of the training set. As a result, the same sample may be selected multiple times, and aggregation refers to applying the method that will ensure prediction accuracy specific to the task. In regression, the average of the predictions from individual classifiers is taken, known as soft voting. In classification, the class with the most votes is selected, and this procedure is known as hard voting or majority voting. BR, which is easy to apply and reduces variance, has good generalization ability in noisy and high-dimensional datasets. However, since the predictions are averaged, there is a loss in interpretability. The individual classifier output may be more precise for the relevant data point, but a definitive insight cannot be obtained. Without clustered systems or many processing cores, bagging becomes computationally expensive as the number of iterations increases, leading to slower processing. It is more compatible with less stable algorithms. In highly stable models, the diversity within the dataset is smaller due to high bias, which limits flexibility in terms of adaptability. Ultimately, in Table 3, the advantages and disadvantages of bagging (although separated during the training process, it shares the logic with boosting) and RF are explained within a common pool.

A new sample set of size T is randomly generated using the bootstrapping strategy, with the same size as the training sets. A regression tree corresponding to this method is created for each training set, and the corresponding trees are applied to the test sample. The predicted values are obtained by taking the average of the values of each tree and use Eq. (49) (Dong et al., 2021).

Table 3
Pros and cons of bagging and RF regressions.

Pros	Cons
It requires less normalization and scaling.	When bootstrapping is performed, a small change in the data causes individual tree structures to change significantly due to their dependence on the data sets.
Data preparation is easy. To use feature selection, it is sufficient to create a bootstrap set and a certain number of DTs.	It is applied in a more complex way compared to DTs due to the extra steps taken for bagging and the iterative requirements for producing the entire forest, which increases computational power and resources.
Higher accuracy is guaranteed compared to individual trees.	Even though smaller sample and feature sets are used for each tree, the training requires much more time compared to DT because more data will be processed.
Because it uses linear splits and a tree ensemble, it performs well with nonlinear data.	Compared to DT, its interpretability and understandability are more difficult due to the large number of trees.
With the use of random feature selection and bagging techniques, the risk of overfitting remains low, even in large datasets.	Bagging partially improves prediction performance beyond the training data range, but it is not sufficient and cannot predict the entire dataset.
Since they use a smaller dataset, they work with high speed and accuracy.	It requires tracking a completely random seed to regenerate specific outputs and determine bootstrap sets, which results in slower performance.
It is not sensitive to missing data and outliers. In such datasets, it can be preferred in many applications because it avoids grouping from values that are very far away.	

$$\widehat{h}(x) = \frac{1}{T}\sum_{t=1}^{T}\widehat{f}_t(x) \tag{49}$$

Theoretically, the predicted variance is reduced to the variance of the original single learner * 1/number of learners. The average of the output prediction values of all regression tree models forms the final value of the bagging ensemble regression tree algorithm, and the mathematical model of the algorithm is as shown in Eq. (50) (Peng et al., 2023).

$$h_A(x) = sign\left(\sum h_i(x)\right) \tag{50}$$

Here, $h_A(x)$ is the output value of the bagged tree ensemble regression, and $h_i(x)$ is the output value of each regression tree. In such regression analyses, the optimization of the number of regression trees is a key approach. If the number of regression trees is small, accuracy is lower. An excessive number of trees results in a high computational load.

In this study, the hyperparameter space considers the number of estimators, maximum features, random state, and verbose. The numerical values selected for these parameters are explained in later sections, and detailed hyperparameter definitions can be found by examining the Python sklearn library.

2.2.8. Voting regressor (VR)

The performance deficiencies in predictors are evaluated in the ensemble feature of the VR. The voting classifier and VR are divided into two types. If only classification tasks are involved, the use of the voting classifier is valid. Otherwise, the VR is referred to for regression functions. Based on the principle of the wisdom of crowds, the average or weighted average of the component models is considered for the final prediction. It combines predictions from multiple regression models, providing a more accurate and robust operation. The mathematical analysis of the VR is quite similar to that of the voting classifier and involves the ML crowd (M_1, M_2, …, M_x). Each model generates a prediction (P_n) for the input data I. The final prediction from the VR is expressed as P_{voting}. In the simple averaging mode, the total weight determined is then distributed equally across all models. On the contrary, for better performance in specific data subsets, certain model predictions may need to be weighted more heavily. Simple and weighted weights are determined using Eqs. (51) and (52). Here, ωt_n represents the special weights assigned during the training process.

$$P_{voting} = \frac{1}{2}\sum_{n=1}^{x} P_n \quad (for\ simple\ average) \tag{51}$$

$$P_{voting} = \frac{1}{2}\sum_{n=1}^{x} (P_n.\omega t_n) \quad (for\ weighted\ average) \tag{52}$$

By including various regression models in the ensemble, the strengths and weaknesses of individual ML techniques are complementarily executed with the VR. Each model, trained on its respective dataset, generates its own prediction. These predictions are then combined using a summation method (averaging) to form the final ensemble prediction. This process helps to smooth out the noise introduced by each model, improving the overall prediction accuracy and ensuring more stable performance for large datasets. To obtain optimized outputs by reducing the weaknesses in the models, it is crucial to carefully select complementary features. In this study, the hyperparameter space is not considered for the VR. Instead, individual ML techniques such as RF, SVR, and k-NN are introduced to the VR. The goal is to achieve a more applicable and higher accuracy regression model.

2.2.9. Support vector regression (SVR)

SVR aims to find the hyperplane that best separates the classes in the input data and minimizes the losses. The number of features in the input

data determines whether the hyperplane is a line in a two-dimensional space or a plane in an n-dimensional space. In general, in SVM, the fewer the input features, the simpler the hyperplane that is created in that feature space. The hyperplane is chosen to maximize the margin between the nearest data points in each class, and these closest data points are known as support vectors. A set of hyperplanes in a high-dimensional space is created, which can be used for classification, regression, or outlier detection. As the margin increases, the generalization error decreases, thus preventing overfitting. In SVR, there is a slight difference in how the margin is defined. The concept of margin here considers the tolerance for error and is often referred to as the ε-insensitive tube. SVR is a nonlinear method developed from traditional statistics, where a small deviation of data points from the hyperplane is allowed (Xu et al., 2020). In SVR, a function $f(x)$ is fitted to approximate the observed target values, capable of predicting numerous engineering problems (Sharafati et al., 2021). SVR maps input data to a higher-dimensional feature space in a nonlinear manner and solves the LR problem (Zhang et al., 2018). The objective is to reduce complexity to prevent overfitting while finding the best fit that can accurately predict the target variable (Rasifaghihi, 2023).

The ε-insensitive tube considers the deviation between the real and predicted target values and improves the model's generalization ability according to model complexity. Mathematically, this is based on a convex optimization problem where a function $f(x)$ is sought that allows a maximum ε deviation from the targets for all training data. Finding a flatter function is a viable approach as it is less sensitive to small changes in the input data, reducing the risk of overfitting. Especially in univariate linear SVR, the computational functional representation that allows or does not allow tolerance for error is explained in Fig. 3.

The best fitting function, when error tolerance is not considered, corresponds to a small w value in $f(x) = wx + b$. In cases where the optimization problem cannot be applied, slack variables outside the ε-insensitive tube may be allowed to incur some error. A balance between the slack variables' distance from the tube and the model's complexity is achieved by using a non-zero C regularization parameter. Data points within the ε-insensitive tube are assigned zero prediction error, while slack variables are penalized proportionally to their distance from the margin (ζ_i, ζ_i^*). This approach reduces the risk of overfitting while addressing nonlinear regression through kernel functions. In the case of linearly inseparable data, the goal is to transform the data into a higher-dimensional space where it becomes linearly separable, allowing methods like logistic regression (LOR) or other classifiers to perform better. The kernel function facilitates the transition in the feature space, and common types include linear (Eq. (53)), polynomial (homogeneous) (Eq. (54)), polynomial (inhomogeneous) (Eq. (55)), Gaussian radial basis function (Eq. (56)), and sigmoid function (Hyperbolic tangent) (Eq. (57)).

$$K(x_i, x_j) = x_i^T x_j \tag{53}$$

$$K(x_i, x_j) = \left(x_i^T x_j\right)^d \quad (d = 1 \quad \textit{linear kernel}) \tag{54}$$

$$K(x_i, x_j) = \left(x_i^T x_j + r\right)^d \tag{55}$$

$$K(x_i, x_j) = \exp\left(-\gamma \left\|x_i - x_j\right\|^2\right) \quad \rightarrow \textit{for } \gamma > 0 \tag{56}$$

$$K(x_i, x_j) = \tanh\left(\kappa . x_i^T x_j + c\right) \rightarrow \textit{for some (not every) } \kappa > 0 \quad \textit{and } x > 0 \tag{57}$$

Here, the parameter γ, which controls the rate of decay of the kernel, can be adjusted as $\gamma = 1/(2.\sigma^2)$ based on a free parameter σ (Lin et al., 2022). By defining the dot products of the input data vector pairs to the kernel function, the variables in the original space can be easily computed. The hyperplane in the higher-dimensional space corresponds to a set of data points where the dot product with the vector in that space is constant and orthogonal. This process is known as the kernel trick, where only a series of similarity comparisons between pairs of original data points are performed. In this way, inputs are transformed into coordinates in a higher-dimensional feature space, and implicit matching of feature spaces is facilitated by the kernel trick. However, in SVR, the training is conducted with a subset of the training data. The original SVR training is performed according to the procedure outlined in Eq. (58), where x_i is the training sample and y_i is the target value. $\langle w, x_i \rangle + b$ is the prediction for the respective sample, and ε is a free parameter serving as a threshold.

$$\textit{minimize } \frac{1}{2}\|w\|^2$$

$$\textit{subject to } |y_i - \langle w, x_i \rangle - b| \leq \varepsilon \tag{58}$$

A common approach to manage regression is based on the least squares method. It minimizes structural and experimental risks to recognize the patterns between predictor and predicted variables (Modaresi et al., 2018). However, the presence of outliers in regression problems can reduce the performance of least squares estimators. A robust predictor can be developed, which is not sensitive to small changes in the model (Shabani et al., 2020). SVR, due to its flexibility in allowing maximum error tolerance and cost penalty, can be optimized for better model development based on these two parameters (Dewi and Chen, 2019). Moreover, the distance from any point on the hyperplane indicates the error for that particular point, making the least squares method an ideal approach for LR problems. However, when outliers are present in regression problems, using the least squares estimator may become impossible; as a result, the processor will show poor performance. Hence, a robust predictor that is not sensitive to small changes in

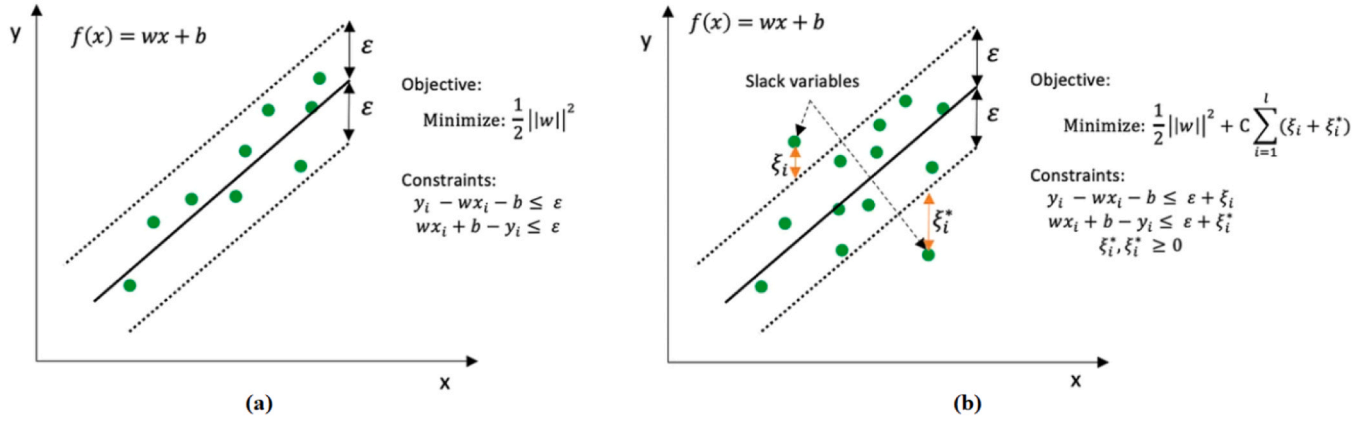


Fig. 3. Univariate linear SVR logic: (a) Univariate linear SVR (with zero tolerance for error), (b) Univariate linear SVR (allowing for errors).

the model should be developed.

Given the training vectors $x_i \in \mathbb{R}^p$ for $i = 1, \ldots, n$ and the vector $y \in \mathbb{R}^n$, SVR primarily solves the problem as shown in Eq. (59).

$$\underset{w,b,\zeta_i,\zeta_i^*}{\min} \frac{1}{2} w^T w + C. \sum_{i=1}^{n} (\zeta_i + \zeta_i^*)$$

$$\text{subject to } y_i - w^T . \varnothing(x_i) - b \leq \varepsilon + \zeta_i$$

$$w^T . \varnothing(x_i) + b - y_i \leq \varepsilon + \zeta_i^*$$

$$\zeta_i, \zeta_i^* \geq 0, \quad i = 1, \ldots, n \tag{59}$$

The predicted value is penalized for being at least ε away from the actual values. Depending on whether the tube is above or below the target, the penalty is applied with ζ_i or ζ_i^*. The binary problem is addressed as in Eq. (60), where e represents the vector of all ones. Q is an n positive semi-definite matrix, and $Q_{ij} = K(x_i, x_j) = \varnothing(x_i)^T . \varnothing(x_j)$ is the kernel. The implicit mapping of the training vectors into a higher-dimensional space is consider through the ∅ function. Ultimately, the prediction of the targets is made using Eq. (61), and the schematic diagram of SVR is shown in Fig. 4 (Zhang et al., 2018).

$$\underset{\alpha,\alpha^*}{\min} \frac{1}{2} (\alpha - \alpha^*)^T . Q(\alpha - \alpha^*) + \varepsilon . e^T (\alpha + \alpha^*) - y^T (\alpha - \alpha^*)$$

$$\text{subject to } e^T (\alpha - \alpha^*) = 0$$

$$0 \leq \alpha_i, \quad \alpha_i^* \leq C, \quad i = 1, \ldots, n \tag{60}$$

$$\sum_{i \in SV} (\alpha_i - \quad \alpha_i^*) . K(x_i, x) + b \tag{61}$$

In this study, the hyperparameter space includes kernel, degree, gamma, coef0, the coefficient C, ε, cache size, maximum iterations, and verbose. The kernel function methods evaluated are 'linear', 'poly', 'rbf', 'sigmoid', and 'precomputed'. When the polynomial kernel is used, the degree parameter, which will later explain the numerical range, is considered. Gamma is the kernel coefficient, and in the "scale" case, if the gamma scaling data exceeds, it is set as 1 / (n_features * X.var ()). In the "auto" case, it is set as 1 / n_features. In the kernel function, the independent term is expressed with coef0, while the strength of regularization is inversely proportional to the coefficient C. The ε parameter is used to evaluate the distance of the predicted data point from the actual value, and cache size represents the size of the kernel cache. The numerical values chosen for these parameters are explained in later sections, and detailed hyperparameter definitions can be found in the Python sklearn library documentation.

2.2.10. *Artificial neural network: Multi-layer perceptron regressor (MLPR)*

The ways in which biological neurons collaborate to recognize patterns, make decisions, and reach conclusions are emulated by neural networks, which manage a process similar to that of the human brain. Each neural network is composed of nodes or artificial neurons loosely modeling the neurons in the brain, and includes an input layer, one or more hidden layers, and an output layer. Here, artificial neurons are connected by edges that model the synapses in the brain. Each artificial neuron processes the signal it receives and sends it to other neurons, and the neuron outputs are evaluated in the nonlinear function of the sum of the inputs. The strength of each signal changes according to the weight in the learning process. Neurons gather in layers and are transformed according to their inputs in different layers. The node activity is determined based on the threshold value at the nodes, which influences the data transmission to the next layer of the network. The layer that receives external data is considered the input layer, while the layer that produces the final result is the output layer. While single-layer and non-layered networks can exist, a pattern with two or more hidden layers corresponds to the term "fully connected." The input data is consider through weighted connections in the hidden layer and is compared to threshold values (Xu et al., 2023). Regression predictions are then obtained from the output layer via the activation function. The neuron output in regression predictions, based on the connection weight (ω_i) and the threshold (θ), is determined by Eq. (62). The accuracy of such a network is heavily dependent on the training data during the learning and development phases. The accuracy of such a network is highly dependent on the training data during the learning and development phases. A simple neural network's LR model is explained by Eq. (63).

$$y = f\left(\sum_{i=1}^{n} w_i . x_i - \theta \right) \tag{62}$$

$$f(x) = \sum_{i=1}^{n} w_i . x_i + bias \tag{63}$$

After the input layer is determined, weights are assigned to identify which variables might have a greater contribution to the outputs. Once all inputs are multiplied by their respective weights, they are sent to the activation function to determine the output. If any output exceeds a certain threshold, the node is activated, and the data is passed to the next layer of the network. Since the output of one node serves as the input for another, this type of neural network is referred to as a feedforward neural network. A neural network makes increasingly complex decisions based on the outputs of previous layers. To minimize the difference between the predicted and target output, also known as experimental risk, parameter optimization of the network is essential during the training process. The goal is to minimize the cost/loss function, and therefore, as the model weights and biases are adjusted, cost functions and reinforcement learning (RL) approaches can be used to reach a local minimum. During the training phase, the network's weights are iteratively updated to reduce the defined loss function, and the neurons continue to learn from the labeled training data. The cost is a statistical phenomenon whose value can only be approximately calculated, and gradient descent, which helps determine the direction of the weights to reduce model errors, may be effective in this task. The backpropagation algorithm is also preferred, as it moves in the opposite direction from output to input. The error for each neuron is calculated, and the model parameters and connection weights can be appropriately adjusted. The

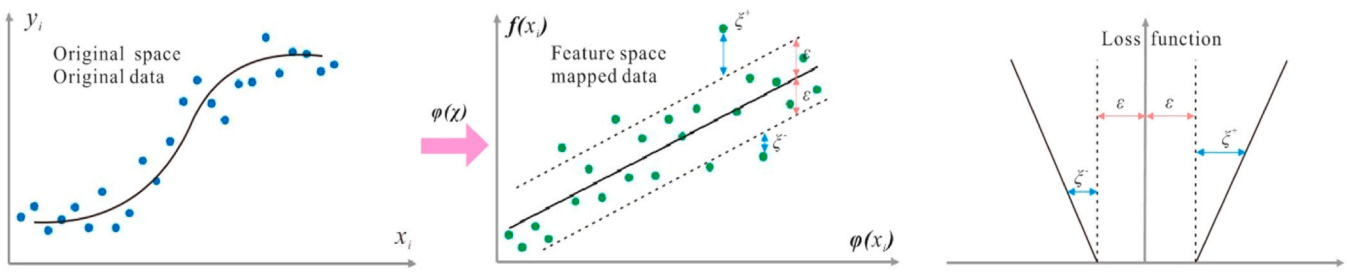


Fig. 4. SVR schematic diagram.

gradient of the task-specific cost function is computed with respect to the weights. Ultimately, learning requires the network to adapt in order to perform a task properly. In this process, directly related to the adjustment of the weights, if additional observations indicate that the error rate is not effectively decreasing, it can be concluded that the learning period has been completed. While it is expected that the error will never reach zero, if the error is high, the network needs to be redesigned (Darbor et al., 2019).

If the labels are known during the learning process to produce the desired output for each input, a supervised training process can be carried out based on the concept of learning with a teacher, and it can be applied to tasks such as pattern recognition, regression, etc., and it may also be valid for sequential data. Input data, the cost function, and the network's output are known in the unsupervised learning process, while labels are unknown prior to training, so it is commonly used in prediction tasks, clustering, statistical distributions, compression, and filtering. In RL, which aims to perform the action that will minimize the expected cumulative cost, the process is conducted based on the agent and action activities. The agent performs actions and generates instantaneous costs. At each milestone, the agent decides whether new actions should be discovered or previous learning should be continued. A reward-penalty system is applied for self-learning. ANNs, with their ability to model nonlinear and complex relationships, can perform tasks that traditional algorithms may fail at. The ability to model any task is directly related to the capacity feature, which evaluates the amount of information that can be stored in the network and the trade-off in complexity. In terms of information capacity, the functions modeled by the network for any data input are captured, and the maximum capacity under optimal conditions is found using the principles of measurement theory, with the VC dimension (memory capacity) representing this maximum. Beyond the capacity feature of ANNs, the convergence property, which evaluates local minima depending on the cost function and the model, comes to the forefront. If the training process starts far from a local minimum, depending on the optimization method, it may fail to converge. In large datasets, the convergence process becomes more complex. Once the saddle point is passed, convergence can lead to wrong directions. Another important aspect that must be carefully addressed is the generalization ability. This concept, which is directly related to overfitting, requires techniques such as cross-validation in hyperparameter selection to control overfitting and improve generalization ability. However, for this, the hyperparameters must be correctly determined before the learning process. Learning rate, momentum, number of hidden layers, batch size, and epoch are some of these parameters. The size of corrective steps taken to adjust the errors in each observation is known as the learning rate. This causes a trade-off between training time and final model accuracy. While a high learning rate can shorten the training time, it may lead to unavoidable low accuracy. Additionally, a high learning rate can cause the gradient descent to overshoot, resulting in the network missing the minimum point when updating the network's connection weights and thresholds. Dividing the training set into several mini-batches is possible with mini batch size. As the batch size increases, individual characteristics stored by the samples may be ignored. On the other hand, with lower batch sizes, efficiency in learning and model training is sacrificed. Since weight adjustment depends to some extent on previous changes, the concept of momentum is introduced to balance the training and previous changes. Momentum values close to zero represent the slope, while values close to one indicate the most recent change. In addition to cross-validation, for regularization, $\ell 2$ regularization can be used to prevent overfitting ($R = k.\sum w_i^2$). Such a training process requires a large number of training examples in real-world applications and many task-specific representative examples to perform good generalization for new situations.

In this study, it is known that MLP serve as an approximator to the universal approximation theorem. Because they can model nonlinear and complex processes in the real world, they are a commonly used supervised ML technique for knowledge acquisition and inference. The hidden/medium layer is the central part that analyzes data using activation functions and passes the outputs to the output layer. The MLP model can be expressed as in Eq. (64), where *out* is the output, *act* is the activation function of the output, act_h is the activation of the hidden neuron, τ_{jo} and τ_{ko} are the biases of the hidden and output neurons, $\mathscr{X}_{kj}$ are the output layer's weight connections, and $\mathscr{X}_{ij}$ are the hidden layer's weight connections, with M representing the number of hidden neurons and N representing the number of inputs (Ehteram et al., 2023).

$$out = act\left[\sum_{i=1}^{M}\mathscr{X}_{kj} \quad . \quad act_h\left(\sum_{i=1}^{N}\mathscr{X}_{ij}.x_i + \tau_{jo}\right) + \tau_{ko}\right] \tag{64}$$

The hidden layer structure is determined based on the type of problem, its size, and the solver speed, and Eq. (65) is used to compute the output in each case (Ramezanian et al., 2019).

$$\alpha^{L+1} = f^{L+1}\left(W^{L+1}.\alpha^{L} + b^{L+1}\right) \tag{65}$$

Here, L is the number of layers in the network, and α is the output vector in each layer. W represents the weight of each synapse, and b represents the error entered from the previous stage. In this structure with input, hidden, and output layers, if the number of neurons in the hidden layer is low, the neural network may lead to underfitting, while if it is too high, it may cause overfitting. Therefore, Heaton's (Heaton, 2015) recommendation can be used for the number of neurons in the hidden layer. The number of neurons here is determined according to the number of influencing factors (D) and the number of outputs (No), and is expressed as $2D/3 + No$. Additionally, the number of neurons in the hidden layers should not exceed $1.5D$ (Tien Bui et al., 2018). On the other hand, the number of neurons in the input and output layers is determined based on the number of predictors, and to train neural networks and achieve the best predictions, the error function should be minimized at each iteration (see Eq. (66)).

$$E = \frac{1}{nc}.\sum_{i=1}^{nc} e_i^2 \tag{66}$$

Here, E represents the error function, e_i is the model error for the i^{th} training pair, and nc represents the number of training pairs (Modaresi et al., 2018). Specifically, in generalized regression networks, the computational procedure in Eq.(67) is performed based on the kernel function $f(X_r, t)$ and the input X_r to determine the predicted vector output Y_r. MLP training can be divided into two phases: forward and backward propagation according to the backpropagation algorithm.

$$Y_r = \frac{1}{\sum_{t=1}^{n} f(X_r, t)} \sum_{t=1}^{n} [f(X_r, t).T_t] \tag{67}$$

In forward propagation, the sum of the products of input neurons and connection weights is used as the starting point. Output values are then propagated forward, considering hidden layer values and connection weights. In contrast, in backward propagation, output responses are compared with target values, and the connection weights are updated to minimize the difference. Unlike propagation, another perspective is gradient descent. Especially in ANN individual learners, two types of gradient descent can be used: root mean square propagation (RMSprop) and adaptive moment estimation (Adam). After iterations in the cost function, the RMSprop principle, which depends on the accumulated gradient momentum (S_{dw};S_{db}) and gradient accumulation (β), is explained by Eqs. (68)-(71).

$$S_{dw} = \beta.S_{dw} + (1-\beta).dW^2 \tag{68}$$

$$S_{db} = \beta.S_{db} + (1-\beta).db^2 \tag{69}$$

$$W = W - \alpha.\frac{dW}{\sqrt{S_{dw}} + \varepsilon} \tag{70}$$

$$b = b - \alpha.\frac{db}{\sqrt{S_{db}} + \varepsilon} \tag{71}$$

Here, α is the learning rate, while W and b are the weights and thresholds in the networks. ε is a very small value. Compared to RMSprop, which accelerates the function by suppressing the oscillatory behavior of the cost function, the Adam algorithm includes momentum terms (see Eqs. (72)-(77)).

$$V_{dw} = \beta_1.V_{dw} + (1-\beta_1).dW \tag{72}$$

$$V_{db} = \beta_1.V_{db} + (1-\beta_1).db \tag{73}$$

$$S_{dw} = \beta_2.S_{dw} + (1-\beta_2).dW^2 \tag{74}$$

$$S_{db} = \beta_2.S_{db} + (1-\beta_2).db^2 \tag{75}$$

$$W = W - \alpha.\frac{V_{dw}}{\sqrt{S_{dw}} + \varepsilon} \tag{76}$$

$$b = b - \alpha.\frac{V_{db}}{\sqrt{S_{db}} + \varepsilon} \tag{77}$$

The inertia of gradient descent is considered with the momentum terms (V_{dw} and V_{db}), improving the stability of the cost function and increasing the convergence speed. A feedforward MLP schematic with an input layer, one or more hidden layers, and an output layer is shown in Fig. 5 (Pham et al., 2019).

In this study, within the hyperparameter space of the MLPR, the following are considered: hidden layer size, activation, solver, learning rate, batch size, alpha coefficient, maximum iterations, momentum, verbose, validation fraction, beta coefficients, and the number of iterations with no change. The number of neurons in the i^{th} hidden layer is determined by the hidden layer size, while activation functions are evaluated with options such as 'identity', 'logistic', 'tanh', and 'relu'. 'Identity' represents a no-operation activation and applies a linear bottleneck ($f(x) = x$). The logistic function with a sigmoid form computes the calculations using $f(x) = 1/(1+\exp(-x))$. Another activation operation is performed using $f(x) = \tanh(x)$, while 'relu' refers to the rectified linear unit function and uses $f(x) = \max(0,x)$. As an alternative solver, a quasi-Newton improvement technique abbreviated as 'lbfgs' can be used. SGD and the Adam algorithm are considered among the other solvers. While 'lbfgs' stands out for its speed and good performance in small datasets, the Adam algorithm guarantees higher accuracy in large datasets. In the $\ell 2$ regularization process, the alpha coefficient is used and is included in the loss function to determine the sample size. A constant learning rate set by the learning_rate_init is considered with the 'constant' option, while 'invscaling' is used in stochastic gradient descent, where the learning rate is progressively reduced at each step according to an exponential inverse scaling factor. When the training loss decreases, the learning rate is kept constant according to learning_rate_init; otherwise, the 'adaptive' approach adjusts according to the validation score and tolerance. In the case of using the Adam algorithm, beta coefficients are considered for the primary and secondary momentum equations to determine the exponential decay rates. The selected numerical values for these parameters are explained in later sections, and detailed hyperparameter definitions can be found by examining the Python sklearn library.

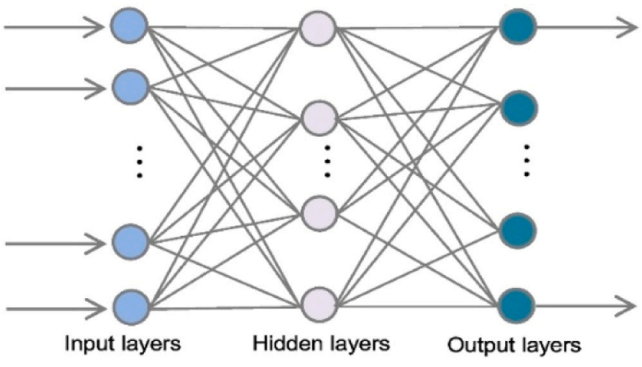


Fig. 5. MLP schematic diagram.

2.2.11. Deep learning (DL)

The simulation of the human brain's complex decision-making mechanism can be likened to the use of DNNs. Inspired by deep generative models such as the nodes in DBNs and Boltzmann machines, this approach relies on multi-layered networks like convolutional neural networks and transformers. These multi-layered networks use a layer hierarchy to transform input data into more abstract and composite formats, distinguishing them from simple neural networks with one or two computation/hidden layers. As a common belief, three or more hidden layers are typically sought in DNNs. However, there is no universally accepted depth criterion, and it is evaluated based on the universal approximation theorem or probabilistic inference. In this approach, which enhances automation and is a branch of data science capable of performing analytical and physical tasks without human intervention, there are elements like data inputs, weights, neurons, synapses, functions, and biases, as in ANNs. The input layer contains neurons equal to the number of features, while the output layer contains neurons equal to the number of output probabilities (Bailly et al., 2022). With a greedy layer-by-layer hierarchy, DL helps solve the concept of abstraction and select parameters that will improve feature performance. In DNN, which can model complex linear relationships, the architectural structure can generate object-specific compositional models. The use of additional layers can capture lower-layer features more easily and eliminate noise with fewer units compared to a shallow network. In DL, which consists of multiple layers made up of interconnected nodes built on previous layers to improve predictions, the forward propagation of computations performed across the network is called forward propagation. By mapping virtual neurons and assigning random weights to the connections between them, the input and weight multiplication produces output in the range of 0–1. The weights are adjusted by the algorithm as the network's pattern is correctly defined. The input layer, where input data is received, and the output layer, where final prediction results are obtained, are considered visible layers. In backpropagation, errors in predictions are calculated, and the layers move backward for model training. This allows weights and biases in activation functions to be updated. This process, in which decisions are made about which features will be learned at what level, is relevant to feature engineering topics, as it discovers useful features from the model's data. While this approach is notable for its accuracy, its disadvantages should also be considered. DL, by increasing computational power, requires high-performance and multi-core memory. Although deep algorithm training increases internal resource consumption, its scalability is quite costly. To eliminate resource consumption, distributed cloud computing can also be utilized. Another perspective stems from its rare dependency on training data, which carries the risk of overfitting due to the abstraction layers that facilitate modeling. While regularization techniques exclude rare dependencies, methods such as cropping and rotating can be employed to increase the size of small training sets. Dimensional factors, such as the number of layers and the number of units per layer, consider many training parameters, including learning rate, weights, etc., and compromise time along with resource consumption. Furthermore, the issue of contrasting divergence in gradient descent remains unclear. It retains the 'black-box' terminology in terms of its speed and what it approximates. It can be seen not as a comprehensive solution, but as a correct step that can enhance AI. The inability to represent causal relationships and the lack of a clear path for

logical inference are other concerning aspects of DNN. The question of what objects are used for, how they are used, and how abstract information is integrated remains a separate issue that needs to be solved. Cybersecurity threats are another obstacle. There is a need for research directions that would allow attackers to identify patterns used by the architecture and find matches that AI would not recognize. Even after training, the need for training data does not cease, and the architecture requires constant calibration and updating or an increased demand for validation data.

In DL, referred to as black boxes due to challenges in their internal workings and understandability, there are different types of neural networks. The first type is known as CNNs, which are primarily used in computer vision and image classification. CNNs consist of a series of layers where each volume is transformed into another through differentiable functions. It utilizes linear algebra and matrix multiplication when identifying patterns in images. Each node, which is connected to others, has an associated weight and equation, and when the output of an individual node exceeds a certain threshold, the node becomes active, allowing data transfer. In CNNs, which have an internal structure consisting of convolution, pooling, and fully connected layers, the convolutional layer processes the original input and subsequent iterative inputs to discover detailed patterns. While it increases complexity, this process is essential in defining the majority of the image. The other layers, apart from this convolutional layer, rely on simpler features such as colors and edges. As image data moves through the layers, the network begins to better recognize shapes and objects. Overall, the input layers, where data inputs are received, are first processed by convolutional layers in the initial stages, which apply a series of filters known as kernels to the input data. In this layer, where filters and kernels are applied, the image data is moved, and a dot product is performed between the filter weights and the previous output of the input vector. The inclusion of an activation layer, often using rectified linear unit (ReLU), introduces nonlinearity to the network and is applied to the element-wise output of the convolutional layer. When it comes to the pooling layer, the size of the output volume is reduced to speed up computations, save memory, and prevent overfitting. After the convolutional and pooling layers, the output is flattened into a one-dimensional vector and passed to the fully connected layer for regression or any other task to be performed. Task-specific computations are performed through fully connected layers, and the computed outputs are evaluated in functions such as sigmoid or softmax, which are applied in the output layer, leading to probabilistic score transformation. Maximum and average pooling are commonly used in CNNs. While the CNN architecture allows for data exchange between layers to enhance data processing performance, information may be lost in the pooling layers. It should be noted that the number of filters can affect the output depth, and the distance the kernel moves across the input matrix can change the number of outputs. A larger step results in a smaller output, and when filters do not fit the input image, zero-padding may be applied in conditions of valid, same, or full padding. Here, zero-padding sets all elements outside the input matrix to zero and ensures that an equal or larger output is produced. In addition to addressing many issues, DL stands out for its ability to limit the risk of overfitting and its advantage in achieving high accuracy.

In contrast, RNNs use time series data and are more commonly applied in natural language processing, stock market forecasting, and speech recognition. In this approach, which utilizes memory to retain information from previous inputs, the principle of input-output independence in DNNs is not followed. While RNNs are effective in predicting outputs for future events, in a unidirectional RNN, past events are not considered. Since parameters and weights are shared across each layer of the network, model training parameters can be adjusted through backpropagation and gradient descent. The training process is carried out with backpropagation by calculating errors from the output layer to the input layer, distinguishing it from traditional gradient descent in terms of summing the errors at each step. In feedforward networks, errors do not need to be accumulated since parameters are not shared across layers. However, issues related to exploding and vanishing gradients, which are associated with the gradient dimension representing the slope of the loss function in the error curve, can arise. When the gradient vanishes, the weight parameters become extremely small during updates, and the model stops learning. If the gradient is too large, an unstable model is formed, which is caused by gradient explosion. The model weight grows, reaching a NaN value, and to mitigate both problems, the number of layers in the neural network is reduced. Additionally, the long training time is a major disadvantage, and its use in large datasets is challenging due to the shared feature. As the number of layers and parameters increases, optimization becomes more complex. Due to a number of disadvantages, such as the lack of a memory unit, processing only unidirectional sequential data for training, and the vanishing gradient problem, LSTM has gained prominence in recent years. Compared to the traditional RNN architecture with a single hidden layer that processes information over time, LSTM has a memory cell that can retain information for a longer period, making it easier to learn long-term dependencies. LSTM is relatively insensitive to gap length and has an architecture consisting of a cell, input gate, forget gate, and output gate. In LSTM, which has additional modules that decide when to remember and when to forget information, the values stored in the cell are remembered over time, and there is a tight relationship between the cell and the gates. While the cell stores information, the gates manage memory adjustments. The forget gate decides which information from previous operations will be forgotten by evaluating the previous state and the current input as 1 for storing information and 0 for discarding it. The current input and previous cell output are fed into the forget gate, multiplied by a weight matrix, and a bias is added. The evaluation process between 1 and 0 is managed by a task-specific decision through the activation function. When querying which new information should be stored in the state, the input layer assesses the previous and current states, and assigns values between 0 and 1, which are controlled through the output layer. The input gates are used to store useful information in the cell. In this gate, the data is regulated through the sigmoid function, and the values to be remembered are filtered based on the previous state and the current input. After filtering, the tanh function is applied, and the resulting vector values are element-wise multiplied with the adjusted values. The task of extracting useful information from the current cell state is handled by the output gate, and the operating mechanism is similar to that of the input gate. Ultimately, traditional RNNs tended to have arbitrary long-term dependencies, causing gradients to decay towards zero. In contrast, LSTM allows gradients to flow through the network without any significant weakening, partially solving this issue. However, exploding gradients can still occur, similar to traditional RNNs.

Another approach involves the generative model of the restricted Boltzmann machine (RBM), which pertains to unsupervised learning. Unlike traditional Boltzmann machines, there is no output layer, and it can solve difficult problems when provided sufficient time. The absence of an output layer raises questions about how weight adjustment and scaling operations are performed. However, it can conduct the training process by defining the variability in parameters with correlations between input datasets, comparing them to yet unseen parameters of lower potential. Known as energy-based models due to their identification of distribution effects in statistical mechanics and quantum thermodynamics, RBMs have widespread applications in computer vision, natural language processing, and speech recognition. By using neurons, they learn the probability distribution of a set of input datasets. With visible and hidden layers, neurons within the same layer are not connected, which is where the term 'restricted' comes from. The input data is processed in the visible layer, while a learned set of features is processed in the hidden layer. Neurons in the mutually connected layers reduce the input dimensions by compressing the data representation and perform the learning process. When using contrastive divergence in the training process, the goal is to maximize the probability between the training

data. The hidden layer is activated using weights and biases, and the positive and negative relationships between the visible and hidden units are examined for a standard forward pass transition. Since there is no output layer, to provide a backward transition, the hidden layer neurons are traced back to the input layer, and the input layer is reconstructed. The error is then calculated by subtracting the reconstructed input layer from the actual input layer. The connection weights are adjusted by multiplying the current input, the calculated error, and the learning rate, which is computed between 0 and 1. RBMs are classified according to variable types and functionality features. For tasks such as image or text, when binary variables are used in the layers, the binary RBM class must be modeled. If the layers follow a Gaussian distribution with continuous variables, the Gaussian RBM is used. When composed of multiple RBM layers, a DBN is formed. If the connections between the layers are local and shared, a convolutional RBM is used. If there is a connection between hidden layers across time steps, the temporal RBM is applied.

Ultimately, the DL network based on MLP with three hidden layers is shown in Fig. 6 (Guo et al., 2018).

The transition from the first layer to the second layer can be described using Eqs. (78) and (79) in deep networks.

$$z_l^2 = w_{l0}^{(1)} + \sum_{j=1}^{p} w_{lj}^{(1)} . x_j \tag{78}$$

$$a_l^2 = g^{(2)} . z_l^2 \tag{79}$$

Here, x_j represents the input variables, and $w_{lj}^{(1)}$ is the weight between l and j. The bias term is evaluated using the intercept $w_{l0}^{(1)}$, and to apply any activation function, sigmoid can be chosen, represented by g as the activation function. To achieve the transition between layers (from $k-1$ to k), Eqs. (80) and (81) are used. In this case, l refers to the $k-l$ layer, and lj represents the l unit of the k layer and the j unit of the $k-1$ layer. In setup, the final output is determined by Eq. (82).

$$z_l^{(k)} = w_{l0}^{(k-1)} + \sum_{j=1}^{p_{k-1}} w_{lj}^{(k-1)} . a_j^{(k-1)} \tag{80}$$

$$a_l^{(k)} = g^{(k)} . z_l^k \tag{81}$$

$$F(x_1, x_2, \ldots, x_{p^{(0)}}) = h . \left(w_0^{(k)} + \sum_{l=1}^{p_k} w_l^{(k)} . a_l^{(k)} \right) \tag{82}$$

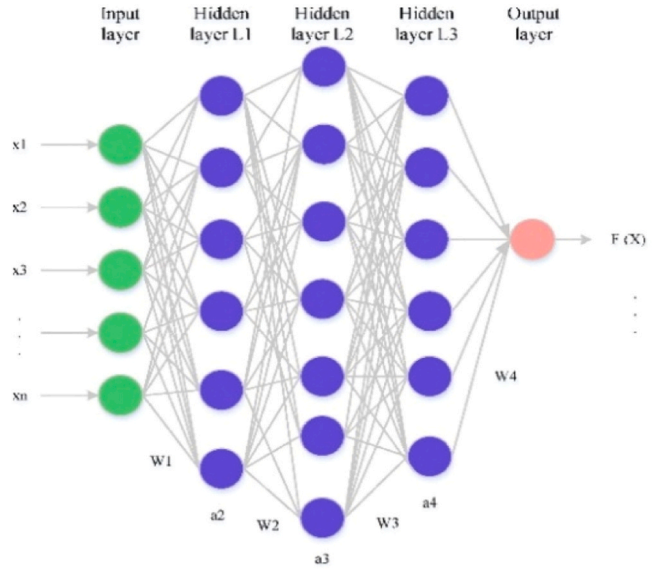


Fig. 6. Deep neural networks with three hidden layers.

In DL setups, ReLU can be used, or a different activation function can be represented by h. To prevent overfitting, the regularization term $J(w)$ can be applied in the loss minimization formula in the propagation process (see Eq. (83)). The explicit form of the regularization term that can be used in loss minimization is expressed in Eq. (84).

$$\min \left\{ \frac{1}{n} . \sum_{i=1}^{n} L[y_i, f(x_w)] + \lambda . J(w) \right\} \tag{83}$$

$$J(w) = \frac{1}{2} . \sum_{k=1}^{K-1} \sum_{j=1}^{p_{k-1}} \sum_{l=1}^{p_{k+1}} \left\{ w_{lj}^{k} \right\}^2 \tag{84}$$

To obtain the gradient, the backpropagation algorithm can be employed. For this task, activations $a_l^{(k)}$ at each layer $L_2, L_3, \ldots, L_K$ for known (x, y) pairs are calculated. If each intermediate quantity is incrementally saved and the function $f(x; w)$ is calculated, the outputs of layer L_K with respect to l can be determined (see Eq. (85)). $\dot{g}$ refers to the derivative of the function $g(z)$ with respect to z, and for each node in layer k, another output can be calculated with Eq. (86). Subsequently, partial derivatives can be taken as shown in Eq. (87), and if a specific performance metric is introduced (such as maximum relative percentage error, MAPE, and MAE), the parameter learning task is completed.

$$\delta_l^K = \frac{\partial L[y, f(x, W)]}{\partial z_l^K} = \frac{\partial L[y, f(x, W)]}{\partial a_l^{(K)}} . \dot{g}^{(K)} . z_l^{(K)} \tag{85}$$

$$\delta_l^{(k)} = \left(\sum_{j=1}^{p_{k+1}} w_{lj}^{(k)} . \delta_j^{(k+1)} \right) . \dot{g}^{(k)} . z_l^k \tag{86}$$

$$\frac{\partial L[y, f(x, W)]}{\partial w_{lj}^{(k)}} = a_j^{(k)} . \delta_l^{(k+1)} \tag{87}$$

2.3. Hyperparameter optimization methods

2.3.1. Delta rule

The delta rule, which is closely related to gradient descent, forms the foundation of the learning architecture. During training processes, it helps define the roadmap based on the learning rate as a hyperparameter. Before moving in the direction of the steepness between the contours, it is essential to determine the distance to be covered (Buduma et al., 2022). While moving along the contour, as the surface approaches minimum steepness and perpendicularity, shorter steps should be taken. If the error surface is very smooth, the training process in DNNs may take longer. Therefore, the gradient is multiplied by a coefficient such as α, the learning rate. A small learning rate extends training time, while a very large learning rate may cause the minimum steepness to be missed, so the learning rate must be carefully selected.

2.3.2. Gradient descent

Besides backpropagation, gradient descent is often preferred. A 3D model for a linear neuron with two inputs and two weights, where the horizontal dimensions correspond to the weights and the vertical dimension corresponds to the error, can be seen in Fig. 7 and Fig. 8 (Buduma et al., 2022).

The minimum error can be observed at the center of the ellipses, and elliptical contours represent this error. A reduced evaluation procedure in two dimensions can be obtained. The contours here correspond to the weights that correspond to the same error. If the contours are close to each other, the slope is steeper, and the steepest descent direction according to the contours is represented by the gradient vector. In neural network training, the process starts with a random weight in the horizontal dimension. Based on the selected weight in the horizontal direction, the gradient at the known position is evaluated to find the steepest descent direction, and steps are taken during the learning process. The gradient directions are determined iteratively, and after

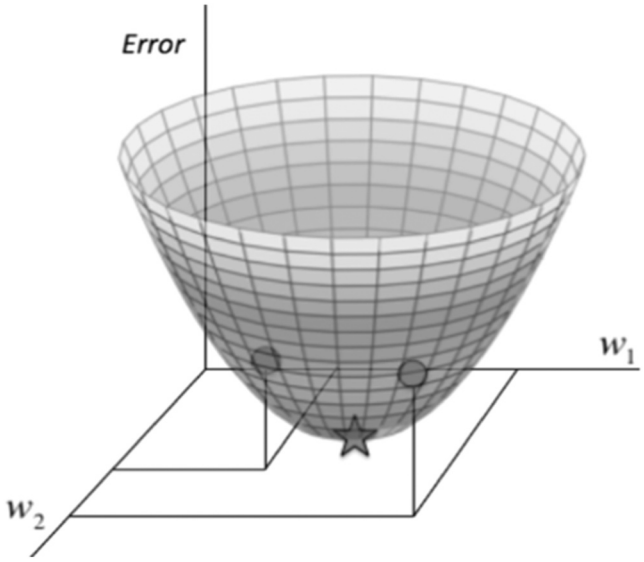


Fig. 7. The quadratic error surface for a linear neuron.

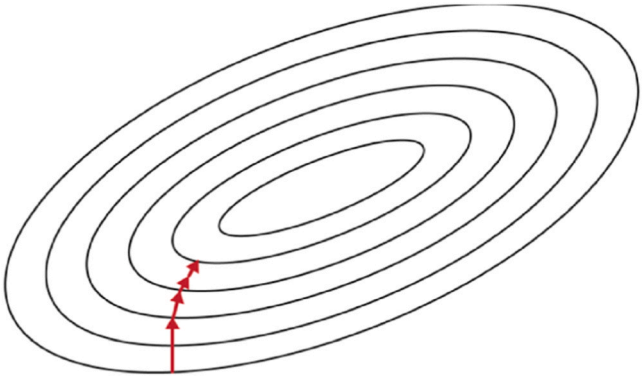


Fig. 8. Visualizing the error surface as a set of contours.

several steps, the gradient approaches its minimum. While individual neurons follow this path in gradient descent, the task becomes more challenging in DNNs.

2.3.3. Stochastic gradient descent

Although differentiable, it is an iterative method that can optimize an objective function and is a workforce algorithm that improves runtime dependencies for optimization (Cui et al., 2018). The true gradient calculated over the entire dataset is replaced by predictions determined based on randomly selected subsets of the data. In high-dimensional optimization problems, it reduces computational burden at the cost of convergence rate. SGD is a valuable optimization method in ML techniques and aims to minimize the objective function in its total form (see Eq. (88)).

$$Q(w) = \frac{1}{n}\sum_{i=1}^{n} Q_i(w) \tag{88}$$

In the stochastic approach, the parameter w is predicted to reduce $Q(w)$ and is associated with the i^{th} observation in the dataset ($Q_i(w)$). While there are least squares and maximum likelihood methods available for minimizing total problems, in the SGD approach, empirical risk, and thus the loss function $Q_i(w)$ for the i^{th} example, is addressed. The iterative procedure is carried out according to batch gradient descent as shown in Eq. (89).

$$w := w - \eta.\nabla Q(w) = w - \frac{\eta}{n}\sum_{i=1}^{n} \nabla Q_i(w) \tag{89}$$

Here, the expression := emphasizes the update of the variable in the algorithm, and the step size is represented by η. In large training sets, if simple formulas are not available, evaluating the gradient sum becomes a much more expensive process, and to reduce computational cost in each iteration, SGD samples a subset of the summation functions at each step. Each training set performs the update mentioned in Eq. (78), and the process continues until the algorithm converges. However, there is a trade-off between the true gradient and the gradient calculation of a single example. Therefore, at each step, the gradient must be computed over multiple training examples, which is referred to as mini-batch. The SGD technique achieves this by randomly dividing the training data into smaller subsets and adjusting the model parameters iteratively (Mohammed et al., 2024). The algorithm calculates the gradients using mini-batches and updates the parameters through incremental adjustments in the opposite direction of the gradient. If the learning rate decreases at an appropriate rate and the objective function is convex or pseudo-convex, it approaches the global minimum approximately. Otherwise, only local minima can be captured.

In the iterative SGD optimization that seeks the minimum and maximum of the target value, the primary goal is to determine the model parameters that will improve the prediction accuracy for both the test and training datasets. Here, the gradient is a vector that shows the general direction of the steepest ascent at a particular point of the function. When seeking accuracy, movement in the opposite direction of the gradient may occur, and the function gradually decreases. Gradient descent algorithm is divided into batch, stochastic, and mini-batch types. In SGD, the model parameters are updated based on only a single randomly selected (hence the term stochastic) training example. This leads to computational efficiency in large datasets. The SGD process starts by randomly selecting the model parameters. Then, the number of iterations and learning rate are adjusted for parameter updates. Randomness is ensured by shuffling the training dataset until the model converges or the maximum number of iterations is reached. Each shuffled training example is repeated, and the model parameters are determined according to the current training example. The gradient of the cost function is calculated accordingly, and a parameter update is performed by stepping in the direction of the negative gradient, scaled according to the learning rate. In addition to the difference at each iteration of the cost function, the convergence state is examined. Ultimately, the optimized model parameters are returned (Wu et al., 2020). However, compared to traditional gradient descent, SGD has more noise and requires more iterations to reach the global minimum. The characteristic analysis of SGD compared to batch gradient descent is explained in Table 4.

In SGD, which stands out with its speed, memory efficiency, and ability to avoid local minima, noise, slow convergence, sensitivity to learning rate, and less accuracy due to noise are issues. The noise in updates can lead to high variance, and oscillations around the minimum increase. It may face stability issues, and a high learning rate may cause overshooting of the minimum. Since SGD updates may not converge to the global minimum, learning rate scheduling and momentum-based techniques may be used.

Given a series of training examples $(x_1, y_1), \ldots, (x_n, y_n)$ where $x_i \epsilon \mathbb{R}^m$ and $y_i \epsilon \mathscr{R}$, the model parameters $w \epsilon \mathbb{R}^m$ and the intercept $b \epsilon \mathscr{R}$ are estimated to learn the linear scoring function $f(x) = w^T.x + b$. If binary classification predictions are involved, the sign of $f(x)$ is considered. During the parameter estimation, the gradient error in Eq. (90) is adjusted to minimize the loss. Here, L is the loss function that measures model mismatch, and the regularization or penalty term R is used to penalize model complexity. $\alpha > 0$ is a hyperparameter that controls the strength of regularization. Popular applications for R are explained by Eqs. (91)-(93).

Table 4
Comparison of stochastic and batch gradient descent.

Characteristic	SGD	Batch GD
Use of the dataset	At each iteration, a random small sample group is used.	The entire dataset is used in each iteration.
Computational efficiency	The computational cost per iteration is low because it processes less data.	Since it processes the entire dataset, the computational cost is higher.
Convergence	The frequency of updates often leads to slower convergence.	Due to the low update frequency, it guarantees faster convergence.
Noise	The frequency of updates causes more noise.	Due to the fewer number of updates, it promises lower noise.
Stability	It has lower stability because it oscillates around the optimal solution.	It converges to the optimum more smoothly and improves stability.
Memory requirements	It processes less data at once and has lower memory requirements.	Since it works with the entire dataset, it requires more memory.
Update frequency	Due to frequent updates, it is more suitable for large datasets and online learning.	It is more suitable for small datasets with a limited number of updates.
Initialization sensitivity	Due to the frequency of updates, it is less sensitive to initial parameters.	Due to the less frequent updates, it is more sensitive to the initial parameters.

$$E(w,b) = \frac{1}{n}\sum_{i=1}^{n} L(y_i, f(x_i)) + \alpha . R(w) \tag{90}$$

$$R(w) := \frac{1}{2}\sum_{j=1}^{m} w_j^2 = \|w\|_2^2 \quad \rightarrow L2 \quad norm \tag{91}$$

$$R(w) := \sum_{j=1}^{m} |w_j| \quad \rightarrow L1 \quad norm \tag{92}$$

$$R(w) := \frac{\rho}{2}\sum_{j=1}^{m} w_j^2 + (1-\rho).\sum_{j=1}^{m} |w_j| \rightarrow Elastic\ net \tag{93}$$

SGD, which performs well in constrained optimization problems, calculates the true gradient of $E(w,b)$ by considering only a single training example, in contrast to gradient descent. Iterations are performed on the training examples, and for each example, the model parameters are updated based on the calculation in Eq. (94).

$$w \leftarrow w - \eta . \left[\alpha . \frac{\partial R(w)}{\partial w} + \frac{\partial L(w^T . x_i + b, y_i)}{\partial w}\right] \tag{94}$$

Here, η controls the step size in the parameter space and is known as the learning rate. While η is fixed and decreases gradually, the intercept (b) is updated with an additional decrease specific to sparse matrices, without regularization (see Eq. (95)).

$$\eta^{(t)} = \frac{1}{\alpha . (t_0 + t)} \tag{95}$$

Here, t refers to the time step, which is determined by the product of the total number of examples and the number of iterations, while t_0 is based on a heuristic method proposed by Léon Bottou, where the expected size of the initial updates to the weights is compared with the expected dimension and is assumed to be approximately 1. The default learning rate schedule for regression is determined based on the inverse scaling principle, according to Eq. (96).

$$\eta^{(t)} = \frac{eta_0}{t^{power_t}} \tag{96}$$

Here, eta_0 and t^{power_t} are hyperparameters chosen by the user, and if a constant learning rate approach is used, the corresponding value is applied with eta_0. If an adaptive approach to the learning rate is considered, the initial value can be set to eta_0. The coefficients, excluding the intercept, are estimated using the equation $coef = \left(\sum_{t=0}^{T-1} w^{(t)}\right)/T$, where T represents the total number of updates in the feature. For details, the Python sklearn library can be reviewed.

In this study, Python's TensorFlow and Keras libraries are utilized. Specifically, the following hyperparameters are considered for the Keras regressor using the SGD approach: epoch, batch size, verbose, learning rate, momentum, activation, input_dim, kernel initializer, dropout rate, weight constraint, and the number of neurons. Among the activation options, relu, sigmoid, softmax, tanh, leaky_relu, linear, mish, softplus, softsign, hard_sigmoid, and logarithmic softmax are included. The softmax function, not previously explained, is expressed as $f(x) = (\exp(x))/(sum(\exp(x))$, while $f(x) = \log(\exp(x)+1)$ corresponds to softplus. The softsign function is formulated as $f(x) = x/(|x|+1)$, and the mish function is described as $f(x) = \mathrm{x}.\tanh(\log(\exp(x)+1))$. From another perspective, the weight initializer options include RandomUniform, zeros, ones, HeNormal, HeUniform, LeCun, and Identity. The numerical values selected for these parameters are explained in later sections, and for detailed hyperparameter definitions, the Python sklearn library can be reviewed.

2.3.4. Hyperparameter tuning & grid search algorithm and auto ML

Hyperparameters are variables that cannot be directly learned or predicted by predictors using optimization algorithms. They are also known as variables introduced when constructing a ML model. In Scikit-learn, they are introduced as arguments to predictor classes, and the hyperparameter space is searched to obtain the best cross-validation prediction score. Hyperparameter search consists of a task-specific (regressor or classifier) predictor, a parameter space, candidate search and sampling, cross-validation, and a scoring function defined by the user based on a metric. Scikit-learn offers two alternatives for parameter search: GridSearchCV and RandomizedSearchCV. Grid search exhaustively considers all combinations of parameters, while randomized search samples candidates from the space of parameters defined by the user, based on the distribution specified. The hyperparameter search process can be shortened by using sequential halving approaches (HalvingGridSearchCV and HalvingRandomSearchCV). These reduction methods allow for competition between candidate parameter combinations. All candidates are evaluated with a small amount of resources in the first iteration. Based on the scores obtained, some candidates are selected for the next iteration, where more resources (e.g., the number of training samples) are allocated. In this study, grid search was preferred due to its high adaptability to the needs of the model being built. In this method, the predictor parameters are optimized through cross-validated grid search on the parameter space (Great Learning Editorial Team, 2024). The performance is evaluated using different parameter settings, and critical parameters related to accuracy and score, such as the learning rate, are identified through the cross-validated model (Rahul, 2021). Cross-validation, being embedded in the process, is applied when performing hyperparameter tuning to train the model. K-fold cross-validation, *a* iterative process that divides the training data into k subsets, was employed. In each iteration, one subset is used for testing, while the remaining subsets are used for training. The model metric is recorded in each iteration, providing the average performance across all folds. Ultimately, in the grid search implementation, the factor candidates are evaluated in the final iteration, and an aggressive elimination phase is performed. If the number of resources is low compared to the number of candidates, the process may extend in the final iteration, as the number of factor candidates to be evaluated increases. Hyperparameter tuning can also be performed using TPOTRegressor, unlike grid search. While running automated ML (AutoML) software for supervised regression tasks, it applies the logic of GAs for intelligent hyperparameter search. It is recommended to run the dataset for an extended period to find a good pipeline, while the AutoML algorithm includes preprocessing steps such as missing value imputation, scaling, PCA, feature selection, various ML regression models, model

hyperparameters, and ensemble or stacking of the algorithms within the pipeline. Although hyperparameter search can be computationally intensive for large datasets, increasing the number of generations and population size expands the hyperparameter combinations and model configurations, making the process inherently time-consuming. Depending on the performance metric defined by the user, the process inherently manages a somewhat stochastic search over the pipeline space. Additionally, it helps to solve previously unexplored ML problems and leaves fine-tuning to more constrained hyperparameter optimization methods, such as grid search. Once training begins, the highest-scoring pipeline components and the creation of optimized areas are determined using a genetic programming algorithm, based on the features and target provided. The genetic programming algorithm is inspired by Darwin's theory of natural selection and consists of a series of stages, including selection, crossover, and mutation. During the selection phase, fitness functions are applied to each individual, and the normalized values are evaluated based on a randomly selected number R, within the range of 0–1. If the fitness function value is $\geq R$, the candidate individuals are retained. In the crossover step, the most suitable individuals selected for the population are used in the crossover. These verified individuals are then subjected to mutation, where random changes are introduced. The process is iteratively repeated until an optimized hyperparameter space is obtained. To prevent overfitting in the data, internal k-fold cross-validation is used, and as a result of optimization, the most suitable pipeline is trained on the test dataset and used as the TPOT-fitted model instance. Rankings are based on mean performance across 5-fold time-series cross-validation. Models with overlapping confidence intervals ($\pm$1 std) are considered statistically equivalent. The final pipeline on the test set is then evaluated for scoring based on the selected regression metric (Olson, n.d.). In Scikit-learn, using the TPOTRegressor class, the dataset must be fitted and evaluated for all these operations. Four key functions stand out for this purpose: fit, dict, score, and export. The training process begins with the fit function using the features and target, while the dict function is used to predict the target values of one or more examples from the feature set. The model is evaluated using the score function with the test dataset (features and target), and an optimized score is returned. The optimized pipeline is then exported using the export function.

The TPOT optimization process is carried out using the fit() method, which evolves ML pipelines through genetic programming. Once the optimal pipeline is identified, predictions are generated using predict(), while performance is evaluated through the score() method. To ensure reproducibility, the final optimized pipeline is exported as a standalone Python script using the export() function. Contrary to the earlier description, TPOT does not include a dict() method, and this reference has been removed to reflect the correct usage of the model. In this direction this study, for the GSA, the number of splits is set to 5, the number of repeats to 3, the random state to 42, verbosity to 1, and the R^2 score is considered. For the AutoML-based TPOTRegressor, generations are set to 50, population size to 100, offspring size to 12, verbosity to 2, the number of splits to 3, and early stopping at 12, with scores evaluated based on the negative mean squared error metric. To ensure fair comparison, both TPOTRegressor and GridSearchCV were allocated equivalent computational budgets: GA used 50 generations $\times$ 100 population size $=$ 5000 evaluations, while GridSearchCV used strategically reduced grids yielding $\sim$4800 combinations. All experiments used random_state$=$ 42. Since LR does not have any critical hyperparameters and LAS has continuous values despite normalization and is unsupported, they are not considered for hyperparameter tuning. As there are no hyperparameters for the VR, after hyperparameter tuning, the best-performing models, support vector, k-NN, and RF, are introduced to the regressor, and performance metrics are evaluated accordingly. The theory behind each ML technique and the meaning of the hyperparameters used in the study were previously explained in the methodology section. For further details, the Python scikit-learn documentation can be consulted. Accordingly, the ML techniques considered in the study, along with the hyperparameters analyzed, are shown in Table 5. The specifications for the created DL models are explained in Table 6.

2.4. Performance metrics

All possible metrics that can be used for regression are shown in Table 7.

In this study, MSE, RMSE, R^2, and MAE have been considered. To elaborate, the variance ratio determined by the independent variables of the model is known as the coefficient of determination (R^2 score). It determines the prediction criterion for samples that were not previously introduced to the model. It is closely related to the dataset, and the best possible score for model prediction is 1. Although negative values can be obtained, this indicates that the model's prediction is poor. If the average of the residuals (errors) is zero, the R^2 score equals the variance score. Here, $\widehat{y}_i$ represents the predicted value for the i^{th} example, while y_i corresponds to the actual target value for the same example. Additionally, the mean of the actual output and the squared error are expressed as $\bar{y} = \left(\sum_{i=1}^{n} y_i\right)/n$ and $\sum_{i=1}^{n}\left(y_i - \widehat{y}_i\right)^2 = \sum_{i=1}^{n} \in_i^2$, respectively. It is important to note that the variance is calculated for the actual output values without correcting for bias. MAE is a risk metric that measures the absolute error and ℓ1-norm loss, calculating the average absolute difference between the predicted and actual output. It can also be referred to as the mean bias error (MBE). Since it does not square the errors, it is independent of the direction and gives equal weight to all errors. It is useful for measuring the magnitude of errors without distinguishing between over- or under-predictions, and small values close to zero are desirable for high prediction accuracy. The risk metric corresponding to quadratic error or loss, known as ℓ2-norm loss, is the MSE. It calculates the error by squaring the difference between the predicted and actual values and averaging this over the dataset. Since it is a squared representation, outliers are given more weight, and this feature is useful for penalization in methods like OLS when searching for the optimal value. The error value ranges from zero to infinity, and for a robust prediction performance, values close to zero are desired. The square root of the same process is known as the RMSE, and lower values close to zero are targeted for high prediction accuracy. It helps in understanding how a feature affects the model's prediction. The squared logarithmic error or loss, known as mean squared logarithmic error (MSLE), is a risk metric that applies a logarithmic function to both the predicted and actual values and calculates the difference between them. It treats small and large errors equally and is resistant to outliers. When the predicted value is lower than the actual value, it is penalized more compared to when the predicted value is larger, thus serving the purpose of penalizing underestimations. MSLE is effective in applications such as commodity prices over time or population growth, where data is evaluated over the years. Another approach for robustness against outliers is the evaluation of the median absolute error metric. It calculates the median of all the absolute differences between the target and predictions. However, if the metric is queried via scikit-learn, the output should not be multi-dimensional. A similar situation applies when determining the maximum error, which captures the worst possible error between the prediction and the expected value. This is often referred to as the maximum residual error, and lower values close to zero are desired. As the name suggests, the explained variance score metric measures the variance of the regression model and should ideally be as close to 1 as possible (a value near zero indicates poor predictions). Since there is no finite score approach, NaN and -Inf values that appear in the result screen may prevent model optimizations, such as grid search validation. In the mean Tweedie deviance metric, the deviation from the expected values of regression targets is measured, and the mean Tweedie deviance error ($D(y, \widehat{y})$) is determined using the power parameter (p). As seen in Table 7, the nature of the metric can change depending on the value of the p coefficient. It is a second-order

Table 5
Hyperparameters based on AI-based techniques.

ML Techniques	Hyperparameter & Solver & Weights	Value
k-nearest neighbors	Number of neighbors	3, 5, 7, and 10
	Weights	Uniform and distance
	Algorithm	Auto, BallTree, KDTree, and Brute
	Leaf size	10, 20, 30, 40, and 50
	p	1 and 2
Ridge regression	Alpha	0.01, 0.1, 1, 2, 3, 4, and 5
	Solver	Auto, singular value decomposition, Cholesky, least square, sparse conjugate, stochastic average gradient descent, limited-memory Broyden–Fletcher–Goldfarb–Shanno algorithm
Lasso regression	Alpha	0.01, 0.1, 1, 2, 3, 4, and 5
	Selection	Cyclic and random
Decision tree	Criterion	Squared error, Friedman mean squared error, absolute error, Poisson
	Splitter	Best and random
	Maximum depth	10, 50, 100 and 200
	Minimum samples split	2, 4, 8, and 10
	Maximum features	None, sqrt, log2
Random forest	Number of estimators	20, 60, 100, and 150
	Criterion	Squared error, Friedman mean squared error, absolute error, Poisson
	Maximum depth	10, 50, 100 and 200
	Minimum samples split	2, 4, 8, and 10
	Minimum samples leaf	1, 2, 4, and 6
	Maximum features	None, sqrt, log2
Gradient boosting regression	Loss	Squared error, absolute error, huber, and quantile
	Learning rate	0.01, 0.1, 1, 2, and 5
	Number of estimators	20, 60, 100, and 150
	Criterion	Squared error, Friedman mean squared error
	Minimum samples split	2, 4, 8, and 10
	Minimum samples leaf	1, 2, 4, and 6
	Maximum depth	10, 50, 100, and 200
	Maximum features	None, sqrt, log2
	Alpha	0.1, 0.2, 0.5, 0.7, and 0.9
Bagging regression	Number of estimators	10, 20, 60, 100, and 150
Support vector regression	C	0.01, 0.1, 10, 50, and 100
	Loss	Epsilon insensitive and squared epsilon insensitive
	Intercept scaling	0.01, 0.1, 1, 5, and 10
	Verbose	0 and 1
Multi-layer perceptron regression	Hidden layer sizes	50, 100, 200, and 300
	Activation	Identity, logistic, tanh, ReLU
	Solver	Limited-memory Broyden–Fletcher–Goldfarb–Shanno algorithm, stochastic gradient descent, Adam
	Alpha	0.0001, 0.001, 0.01, 0.1, and 1
	Learning rate	Constant, inverse scaling, and adaptive
	Momentum	0.1, 0.3, 0.5, 0.7, and 0.9
	Validation fraction	0.1, 0.3, 0.5, 0.7, and 0.9
	Beta	0.1, 0.3, 0.5, 0.7, 0.9, and 0.999
	Number of iteration no change	10, 50, 100, and 200
Deep learning models	Learning rate	0.0001, 0.001, 0.01, and 0.1
	Momentum	0.3, 0.5, 0.7, and 0.9
	Activation	Softmax, ReLU, sigmoid, softplus, tanh, hard sigmoid, exponential, logarithmic softmax, and linear
	Kernel initializer or initial weights	Constant, normal, he_uniform, lecun_uniform, he_normal, glorot_normal, random_uniform, orthogonal, ones, and zeros
	Input dim	Stochastic gradient descent, RMSprop, and Adam
	Dropout rate	0.2, 0.3, and 0.4
	Weight constraint	1, 2, and 3
	Neurons	10, 32, 64, and 128
	Batch size	100, 200, and 400
	Epochs	1, 5, and 10

Table 6
DL model specifications.

Model Specifications	Deep Learning – 1	Deep Learning – 2
Model	Sequential	Sequential
Dense	64	128 (activation: "ReLU")
Dropout	0.1	-
Dense	64	64 (activation: "ReLU")
Dense	1	1 (activation: "linear")
Loss	MSE, MAE, etc.	MSE, MAE, etc.
Optimizer	SGD	Adam
Metrics	R^2-score	R^2-score

homogeneous function, and when the power is two, scaling the actual and predicted values according to the Gamma distribution will not affect the deviation. When the power is one, the metric follows the Poisson distribution and scales the deviation linearly. On the other hand, when the power is higher than one, it scales according to the normal distribution and introduces a non-linear characteristic. As the power increases, less weight is given to potential extreme deviations between the actual and target values, and sensitivity decreases. If a quantile regression model is used for prediction, its performance can be evaluated using the mean pinball loss function. If the quantile parameter (α) is set to 0.5, it will yield half the value of the MAE. By including this metric as part of the scoring, the generalization ability of the quantile regressor can be improved through cross-validation. A larger sign in the loss indicates better model performance. Here, the quantile is a coefficient that determines how many values fall below or above the threshold in the group. The conditional median or quantile of the dependent variable can be predicted using a quantile function based on absolute differences, and it can be applied to residuals containing non-constant variance, without requiring a parametric distribution to find the response. The generalization of the R^2 score is performed using the D^2 score, which calculates the fraction of deviance explained. The skill scores are obtained by generalizing the squared error and adjusting for selection bias, where y_{null} represents the best prediction of an intercept-only model. In Tweedie conditions, y_{null} is the mean of the actual output, while for MAE, it can represent the median, and for pinball loss, it corresponds to the alpha-quantile. Although negative values are possible, the best possible prediction score is 1, and in constant models where input features are neglected, a score of 0 is expected. Relative absolute error measures the deviation between the mean residual and the target function's mean. It is calculated as the ratio of the absolute error to the difference between the actual value and the mean predicted value, and then taken to its absolute value. In this metric, which can range from 0 to 1, values closer to 0 indicate the best regression predictions. Relative squared error is calculated by dividing the square of the difference between each actual value and the mean predicted output by the MSE. In this metric, which also ranges from 0 to 1, values closer to 0 indicate better model performance. The target variable range is normalized with respect to RMSE, and a percentage-based transformation suitable for cross-variable comparison is presented by the relative RMSE. In this approach, which scales the residuals to the actual values, scores smaller than 10 % generally indicate excellent model performance, while scores between 10 % and

Table 7
Description of AI-based regression metrics.

Ref.	Metrics	Equation
(Xu et al., 2023)	**Mean squared error**	$MSE(y,\hat{y}) = \frac{1}{n_{samples}} \sum_{i=0}^{n_{samples}-1} (y_i - \hat{y}_i)^2$
(Sadhukhan et al., 2023)	**Mean squared logarithmic error**	$MSLE(y,\hat{y}) = \frac{1}{n_{samples}} \sum_{i=0}^{n_{samples}-1} (\log_e(1+y_i) - \log_e(1+\hat{y}_i))^2$
(Fan et al., 2019)	**Root mean square error**	$RMSE(y,\hat{y}) = \sqrt{\frac{1}{n_{samples}} \sum_{i=0}^{n_{samples}-1} (y_i - \hat{y}_i)^2} = \sqrt{MSE(y,\hat{y})}$
(Modaresi et al., 2018)	**Coefficient of determination**	$R^2(y,\hat{y}) = 1 - \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{n} (y_i - \bar{y})^2}$
Authors	**Mean absolute error**	$MAE(y,\hat{y}) = \frac{1}{n_{samples}} \sum_{i=0}^{n_{samples}-1} \lvert y_i - \hat{y}_i \rvert^2$
(Lin et al., 2022)	**Mean absolute percentage error**	$MAPE(y,\hat{y}) = \frac{1}{n_{samples}} \sum_{i=0}^{n_{samples}-1} \frac{\lvert y_i - \hat{y}_i \rvert}{\max(\epsilon, \lvert y_i \rvert)}$
Authors	**Median absolute error**	$MedAE(y,\hat{y}) = median(\lvert y_1 - \hat{y}_1 \rvert, \ldots, \lvert y_n - \hat{y}_n \rvert)$
	Maximum error	$MaxError(y,\hat{y}) = \max(\lvert y_i - \hat{y}_i \rvert)$
	Explained variance score	$explained_variance(y,\hat{y}) = 1 - \frac{Var\{y-\hat{y}\}}{Var\{y\}}$
	Mean Poisson, gamma, and Tweedie deviances	$D(y,\hat{y}) = \frac{1}{n_{samples}}.\sum_{i=0}^{n_{samples}-1} (y_i - \hat{y}_i)^2 \rightarrow p = 0\ (normal)$
		$D(y,\hat{y}) = \frac{1}{n_{samples}}.\sum_{i=0}^{n_{samples}-1} \left(2.\left(y_i.\log\left(\frac{y_i}{\hat{y}_i}\right) + \hat{y}_i - y_i\right)\right) \rightarrow p = 1\ (normal)$
		$D(y,\hat{y}) = \frac{1}{n_{samples}}.\sum_{i=0}^{n_{samples}-1} \left(2.\left(y_i.\log\left(\frac{\hat{y}_i}{y_i}\right) + \frac{y_i}{\hat{y}_i} - 1\right)\right) \rightarrow p = 2\ (normal)$
		$D(y,\hat{y}) = \frac{1}{n_{samples}}.\sum_{i=0}^{n_{samples}-1} \left(2.\left(\frac{\max(y_i,0)^{2-p}}{(1-p).(2-p)} - \frac{y_i.\hat{y}_i^{1-p}}{1-p} + \frac{\hat{y}_i^{2-p}}{2-p}\right)\right) \rightarrow otherwise$
	Pinball loss	$pinball(y,\hat{y}) = \frac{1}{n_{samples}} \sum_{i=0}^{n_{samples}-1} \alpha.\max(y_i - \hat{y}_i, 0) + (1-\alpha).\max(y_i - \hat{y}_i, 0)$
	D^2 score	$D^2(y,\hat{y}) = 1 - \frac{dev(y-\hat{y})}{dev(y, y_{null})}$
(Padhma, 2024)	**Relative absolute error**	$RAE(y,\hat{y}) = \frac{\sum_{i=0}^{n_{samples}-1} \lvert y_i - \hat{y}_i \rvert}{\sum_{i=0}^{n_{samples}-1} \lvert y_i - \bar{y} \rvert}$
	Relative squared error	$RSE(y,\hat{y}) = \frac{\sum_{i=0}^{n_{samples}-1} (y_i - \hat{y}_i)^2}{\sum_{i=0}^{n_{samples}-1} (y_i - \bar{y})^2}$
	Relative root mean squared error	$RRMSE(y,\hat{y}) = \sqrt{\frac{\sum_{i=0}^{n_{samples}-1} (y_i - \hat{y}_i)^2}{n_{samples} \cdot \sum_{i=0}^{n_{samples}-1} (\hat{y}_i)^2}}$
	Root mean squared logarithmic error	$RMSLE(y,\hat{y}) = \sqrt{(\log(1+y_i))^2 - (\log(1+\hat{y}_i))^2}$
	Huber loss	$L_\delta(y,f(x)) = \begin{cases} \frac{1}{2}.(y-f(x))^2, & for \lvert y-f(x) \rvert \le \delta \\ \delta.\lvert y-f(x) \rvert - \frac{1}{2}.\delta^2, & for \lvert y-f(x) \rvert > \delta \end{cases}$
	Logarithmic cosine hyperbolic loss	$Logcosh(t) = \sum_{i=0}^{n_{samples}-1} \log(\cosh(y_i - \hat{y}_i))$
	Loss function	$L_{Quantile} = \sum_{i=y_i<\hat{y}_i} (\gamma - 1).\lvert y_i - \hat{y}_i \rvert + \sum_{i=y_i \ge \hat{y}_i} (\gamma).\lvert y_i - \hat{y}_i \rvert$

20 % suggest a sufficiently good model performance. However, if the metric exceeds 30 %, it generally indicates a weakening of the predictive capability. Root mean squared logarithmic error is a performance evaluation technique derived from the MSLE metric (by applying its square root). Like MSLE, it evaluates both small and large errors equally without distinction and is a metric resistant to outliers. Prediction values that are smaller than the actual value are penalized more due to the model's logarithmic nature. In the case of Huber loss, outliers are learned, and they are disregarded in a function that combines linear and quadratic scoring methods. The adjustable hyperparameter value, such as δ, is critical for the metric's ability to assess model performance. For values of δ greater than a threshold, linear loss ($l1$-norm, MAE) is applied, and for values lower than δ, quadratic loss ($\ell 2$-norm, MSE) is used. This combines the advantages of MAE and MSE. Larger loss values are reduced by MAE's weight reduction, while smaller losses are emphasized by the second-degree weight increase from MSE. This approach prevents the selection of outliers from influencing the results. Logarithmic cosine hyperbolic loss works similarly to MSE but is smoother compared to second-degree loss. The combination feature, as seen in Huber loss, also applies here, where the negative effects of high prediction errors do not affect model performance. In the loss function, a selected quantile value γ, which defines different penalties for extreme and low predictions, ranges between 0 and 1. In the model's poor and good prediction performance, the first and second parts of the formula shown in Table 7 dominate, respectively, and in this metric using quantile values, the user determines how positive and negative errors should be weighted. If γ is set to 0.5, both low and high predictions are penalized by the same factor. However, at higher γ values, extreme predictions are penalized more than lower ones. Conversely, in the case of lower γ, lower predictions are penalized more, and this becomes crucial in gradient descent, where the goal is to learn quantiles instead of the average.

All performance metrics are reported in two scales:

1. Normalized Scale [0, 1]: Used during model training and hyperparameter optimization to ensure numerical stability and comparable gradients across features.
2. Physical Scale [kW/m^2]: Post-training, predictions and errors are denormalized using the inverse transform of the target variable scaler:

- y_test_physical = scaler_y.inverse_transform(y_test_normalized.reshape(-1, 1))
- y_pred_physical = scaler_y.inverse_transform(y_pred_normalized.reshape(-1, 1))

Table 8
Optimal hyperparameters for the AI-based regression techniques.

AI-Based Regression Techniques	Optimal Hyperparameters	
	TPOTRegressor (GA)	*Grid Search Algorithm (GSA)*
k-NN	"auto" algorithm, 50 leaf size, 10 neighbors, "distance" weights, and p = 2.	"auto" algorithm, 10 leaf size, 3 neighbors, "uniform" weights, and p = 1.
Ridge regression	"saga" solver and α = 0.1.	"auto" solver and α = 0.01.
Lasso regression	"cycling" selection and α = 0.01.	"cycling" selection and α = 0.01.
Decision tree	"squared error" criterion, max. depth = 200, "sqrt" max. features, min. samples split = 10, and "best" splitter.	"squared error" criterion, max. depth = 10, "None" max. features, min. samples split = 2, and "best" splitter.
Random forest	"Friedman mse" criterion, max. depth = 100, "log2" max. features, min. samples leaf = 1, min. samples split = 4, and 150 estimators.	"poisson" criterion, max. depth = 10, "None" max. features, min. samples leaf = 1, min. samples split = 2, and 100 estimators.
Gradient boosting regressor	"Friedman mse" criterion, learning rate = 0.1, number of estimators = 20, "squared error" loss, max. depth = 10, "log2" max. features, min. samples leaf = 2, min. samples split = 4, and α = 0.5.	"Friedman mse" criterion, learning rate = 1, number of estimators = 20, "squared error" loss, max. depth = 10, "None" max. features, min. samples leaf = 1, min. samples split = 2, and α = 0.1.
Bagging regressor	150 estimators.	10 estimators.
Support vector regressor	C = 100, intercept scaling = 1, "squared ε insensitive" loss, and verbose = 0.	C = 0.01, intercept scaling = 0.01, "ε insensitive" loss, and verbose = 0.
Multi-layer perceptron (MLPR-based ANN)	"identity" activation, hidden layer size = 100, "constant" learning rate, momentum = 0.5, number of iteration no change = 200, "sgd" solver, validation fraction = 0.3, α = 0.01, β_1, and β_2 = 0.3.	"identity" activation, hidden layer size = 50, α = 0.0001, "constant" learning rate, momentum = 0.1, number of iteration no change = 10, "lbfgs" solver, validation fraction = 0.3, β_1 = 0.1, and β_2 = 0.3.
Deep learning models	Learning rate = 0.0001, momentum = 0.3, batch size = 100, dropout rate = 0.2, epochs = 50, neurons = 10, weight constraint = 1, "sgd" optimizer, "ReLU" activation, and "normal" initial weight.	

Table 9
Comparing the forecasting performance of the chosen AI-based techniques.

AI-Based Regression Techniques	Without Hyperparameter Tuning		
	RMSE [mean ± std]	*MAE*	R^2 *Score*
k-NN	0.00185501	0.00089184	0.999941268
Linear regression	**0.00168636**	**0.00082284**	**0.999951461**
Ridge regression	0.00168643	0.00082305	0.999951458
Lasso regression	0.13888568	0.08713432	0.670769824
Decision tree	0.00179705	0.00087142	0.999944880
Random forest	0.00179740	0.00087153	0.999944859
Gradient boosting regressor	0.00220893	0.00116024	0.999916719
Bagging regressor	0.00180556	0.00087430	0.999944357
Voting regressor	0.00178756	0.00100035	0.999945461
Support vector regressor	0.00189741	0.00129717	0.999938552
MLPR - ANN	0.00352351	0.00268089	0.999788098
Deep Learning – Model 1	**0.87011054**	**0.83576435**	**-11.92**
Deep Learning – Model 2	0.00217938	0.00120480	0.999935091
AI-Based Regression Techniques	**With Hyperparameter Tuning – TPOTRegressor (Genetic Algorithm)**		
	RMSE [mean ± std]	*MAE*	R^2 *Score*
k-NN	0.00181895	0.00087879	0.999943529
Ridge regression	0.00168676	0.00083105	0.999951439
Lasso regression	0.01011046	0.00811187	0.998255277
Decision tree	0.00201400	0.00090322	0.999930768
Random forest	0.00179740	0.00087146	0.999944859
Gradient boosting regressor	0.00179849	0.00087551	0.999944792
Bagging regressor	0.00179741	0.00087147	0.999944859
Voting regressor	0.00175887	0.00085448	0.999947198
Support vector regressor	**0.00168636**	**0.00082284**	**0.999951461**
MLPR - ANN	0.00168688	0.00083020	0.999951432
Deep Learning – Model 1	**0.02173787**	**0.16423455**	**0.991934717**
Deep Learning – Model 2	0.00217938	0.00120480	0.999935091
AI-Based Regression Techniques	**With Hyperparameter Tuning – Grid Search Algorithm**		
	RMSE [mean ± std]	*MAE*	R^2 *Score*
k-NN	0.00200829	0.00094688	0.999931161
Ridge regression	**0.00168636**	**0.00082284**	**0.999951461**
Lasso regression	0.01011046	0.00811187	0.998255277
Decision tree	0.00177303	0.00086176	0.999946344
Random forest	0.00174906	0.00084995	0.999947785
Gradient boosting regressor	0.02946875	0.02388999	0.985177952
Bagging regressor	0.00180556	0.00087430	0.999944357
Voting regressor	0.01593178	0.01408112	0.995667752
Support vector regressor	**0.04754562**	0.04218976	**0.961416132**
MLPR – ANN	0.00170209	0.00091970	0.999950552
Deep Learning – Model 1	0.02173787	**0.16423455**	0.991934717
Deep Learning – Model 2	0.00217938	0.00120480	0.999935091

- RMSE_physical = np.sqrt(mean_squared_error(y_test_physical, y_pred_physical))
- MAE_physical = mean_absolute_error(y_test_physical, y_pred_physical)

This dual reporting ensures that metrics are interpretable for both ML practitioners (normalized scale) and energy system operators (physical scale). For example, an RMSE of 0.00168 in normalized units corresponds to approximately 1.85 kW/m^2 when denormalized using the scaling parameters (min=0, max=1.1 kW/m^2).

2.5. Implementation environment and reproducibility

To ensure full reproducibility and facilitate adoption by the research community, we provide comprehensive implementation details covering hardware, software, and methodological specifications.

Computational Infrastructure:

- Processor: Dual Intel Xeon E5–2680 v4 (2.4 GHz base, 3.3 GHz turbo, 28 cores total)
- Memory: 128 GB DDR4–2400 ECC RAM
- GPU: Intel Xeon E5–2650 v3 @ 2.30 GHz, 10 cores / 20 threads, 32 GB RAM, Windows Server 2012 R2 Standard
- Storage: 2 TB NVMe SSD for dataset caching and intermediate results
- Operating System: Ubuntu 20.04.4 LTS (Focal Fossa)

Software Environment:

- Python: 3.8.10 (CPython interpreter)
- Core Libraries:
  - scikit-learn 1.0.2 (ML model implementations and preprocessing)
  - TensorFlow 2.8.0 with CUDA 11.2 (DL models)
  - Keras 2.8.0 (high-level neural network API)
  - pandas 1.4.2 (data manipulation and time-series handling)
  - NumPy 1.22.3 (numerical operations and array processing)
  - matplotlib 3.5.1 and seaborn 0.11.2 (visualization)
- Optimization Libraries:
  - TPOT 0.11.7 (GA-based hyperparameter tuning)
  - scipy 1.8.0 (optimization utilities for grid search)

Data Preprocessing Pipeline (Pseudocode)

Model Training Protocol:

For each AI-based regression technique, the following standardized training procedure was applied:

1. Hyperparameter Search (Section III.C details):

```
# Step 1: Load and merge multi-source data
def load_dataset():
    merra2_data = pd.read_csv('merra2_irradiance_temp_wind.csv')
    solargis_data = pd.read_csv('solargis_atmosphere.csv')
    sarah_data = pd.read_csv('sarah_surface_radiation.csv')
    astronomical_data = calculate_sun_position(location='Antalya')

    # Merge on UTC timestamp
    df = merra2_data.merge(solargis_data, on='timestamp') \
            .merge(sarah_data, on='timestamp') \
            .merge(astronomical_data, on='timestamp')

    # Remove missing/invalid samples
    df = df.dropna().query('ghi >= 0 & temperature > -50')
    return df

# Step 2: Feature engineering
def engineer_features(df):
    df['incident_solar'] = df['ghi'] * np.cos(np.radians(df['angle_of_incidence']))
    df['clearness_index'] = df['ghi'] / df['clear_sky_ghi']
    df['temp_efficiency_factor'] = 1 - 0.0045 * (df['temperature'] - 25)
    return df

# Step 3: Temporal split and normalization
def prepare_train_test(df, test_size=0.30):
    # Sort chronologically
    df = df.sort_values('timestamp').reset_index(drop=True)

    # Split index
    split_idx = int(len(df) * (1 - test_size))

    # Separate features and target
    features = ['ghi', 'solar_altitude', 'temperature', 'wind_speed',
            'incident_solar', 'azimuth', 'declination', 'hour_angle',
            'angle_of_incidence', 'atmsfr_disi_isinim']
    target = 'power_output'

    X_train, X_test = df.loc[:split_idx-1, features], df.loc[split_idx:, features]
    y_train, y_test = df.loc[:split_idx-1, target], df.loc[split_idx:, target]

    # Fit scaler ONLY on training data
    scaler_X = MinMaxScaler().fit(X_train)
    scaler_y = MinMaxScaler().fit(y_train.values.reshape(-1, 1))

    X_train_scaled = scaler_X.transform(X_train)
    X_test_scaled = scaler_X.transform(X_test)  # Transform, not fit_transform!
    y_train_scaled = scaler_y.transform(y_train.values.reshape(-1, 1)).ravel()
    y_test_scaled = scaler_y.transform(y_test.values.reshape(-1, 1)).ravel()

    return X_train_scaled, X_test_scaled, y_train_scaled, y_test_scaled, scaler_y

# Step 4: Time-series cross-validation
def time_series_cv(model, X_train, y_train, n_splits=5):
    tscv = TimeSeriesSplit(n_splits=n_splits)
    cv_scores = []

    for train_idx, val_idx in tscv.split(X_train):
        X_tr, X_val = X_train[train_idx], X_train[val_idx]
        y_tr, y_val = y_train[train_idx], y_train[val_idx]

        # Re-fit scaler on each training fold
        scaler = MinMaxScaler().fit(X_tr)
        X_tr_scaled = scaler.transform(X_tr)
        X_val_scaled = scaler.transform(X_val)

        model.fit(X_tr_scaled, y_tr)
        y_pred = model.predict(X_val_scaled)
        cv_scores.append(r2_score(y_val, y_pred))

    return np.mean(cv_scores), np.std(cv_scores)
```

- GSA: Exhaustive search over manually defined parameter grids.
- GA (TPOT): Population size = 100, generations = 50, mutation rate = 0.9.
- Evaluation metric: Negative MSE (for TPOT) or R^2 (for GSA).
- Cross-validation: 5-fold TimeSeriesSplit.

2. Model Training:

- Random seed fixed at 42 for all stochastic algorithms.
- Early stopping enabled for iterative models (e.g., MLPR: patience = 10 epochs).
- Batch processing for large datasets (batch size = 1024 for neural networks).

3. Performance Evaluation:

- Metrics computed on held-out test set (30 % chronologically last samples).
- Physical units: Denormalized using saved scaler to report RMSE/MAE in kW/m^2.
- Statistical significance: Paired *t*-tests conducted for top-3 model comparisons.

Code Repository: To ensure reproducibility while respecting institutional and data privacy constraints, the full code and preprocessing scripts are not publicly shared at this stage.

3. Experimental results and analysis

This section will first discuss the optimal hyperparameters found for each regression prediction technique. Subsequently, the prediction performance metrics obtained with and without hyperparameter optimization will be compared. Here, the hyperparameter values, solvers, and optimizers obtained through GA and GSA will be explained, and the results will be presented accordingly.

3.1. Optimal hyperparameters for the chosen ML and DL architectures

The possible hyperparameter options for each AI-based regression technique were previously discussed in the theoretical and mathematical analysis presented in Section 2 and Table 5. The optimal hyperparameter values, as well as the solver and optimizer selections for each technique, were determined using either the GA-based TPOTRegressor or the GSA. According to the "This Study" subsection in Section 2, the objective in TPOTRegressor is to achieve the best possible performance based on the negative mean squared error metric. In contrast, the GSA optimization process is guided by the R^2 score. For assumptions involving k-fold cross-validation, the relevant subsection provides further details. The working principles of both optimization algorithms are explained in the "Alternative Choices and Theory" subsection of Section 2. It should be noted that, according to the scikit-learn documentation, LR does not require hyperparameter tuning. Furthermore, LOR is not applicable in this context, as it is generally used for categorical classification tasks rather than for predicting continuous values. In the case of DL, hyperparameter optimization is performed using the GSA in combination with the KerasRegressor, rather than using a GA-based approach. Model-specific characteristics related to the DL architectures, such as the number of layers, are described in the "This Study" subsection of Section 2. In the second DL model, self-training continues until the error rate approaches zero, whereas in the first model, hyperparameter optimization is conducted via KerasRegressor (the optimal hyperparameter configuration listed in Table 8 corresponds to the deep learning – 1 (DL-1) model).

This outcome reinforces the research gap identified in the literature-namely, the insufficient discussion of hyperparameter sensitivity in many prior works. As demonstrated here, performance improvements

Table 10
Performance ranking of AI-based techniques based on metrics for PV output power forecast.

AI Model Performance Ranking	Without Hyperparameter Tuning		
	RMSE [mean ± std]	*MAE*	*R^2 Score*
1.	Linear regressor	Linear regressor	Linear regressor
2.	Ridge regressor	Ridge regressor	Ridge regressor
3.	Voting regressor	Decision tree	Voting regressor
4.	Decision tree	Random forest	Decision tree
5.	Random forest	Bagging regressor	Random forest
6.	Bagging regressor	k-NN	Bagging regressor
7.	k-NN	Voting regressor	k-NN
8.	Support vector regressor	Gradient boosting regressor	Support vector regressor
9.	Deep learning – model 2	Deep learning – model 2	Deep learning – model 2
10.	Gradient boosting regressor	Support vector regressor	Gradient boosting regressor
11.	MLP-based ANN	MLP-based ANN	MLP-based ANN
12.	Lasso regressor	Lasso regressor	Lasso regressor
13.	Deep learning – model 1	Deep learning – model 1	Deep learning – model 1
AI Model Performance Ranking	**With Hyperparameter Tuning – TPOTRegressor (Genetic Algorithm)**		
	RMSE [mean ± std]	*MAE*	*R^2 Score*
1.	Support vector regressor	Support vector regressor	Support vector regressor
2.	Ridge regressor	MLP-based ANN	Ridge regressor
3.	MLP-based ANN	Ridge regressor	MLP-based ANN
4.	Voting regressor	Voting regressor	Voting regressor
5.	Random forest	Random forest	Bagging regressor
6.	Bagging regressor	Bagging regressor	Random forest
7.	Gradient boosting regressor	Gradient boosting regressor	Gradient bagging regressor
8.	k-NN	k-NN	k-NN
9.	Decision tree	Decision tree	Deep learning – model 2
10.	Deep learning – model 2	Deep learning – model 2	Decision tree
11.	Lasso regressor	Lasso regressor	Lasso regressor
12.	Deep learning – model 1	Deep learning – model 1	Deep learning – model 1
AI Model Performance Ranking	**With Hyperparameter Tuning – Grid Search Algorithm**		
	RMSE [mean ± std]	*MAE*	*R^2 Score*
1.	Ridge regressor	Ridge regressor	Ridge regressor
2.	MLP-based ANN	Random forest	MLP-based ANN
3.	Random forest	Decision tree	Random forest
4.	Decision tree	Bagging regressor	Decision tree
5.	Bagging regressor	MLP-based ANN	Bagging regressor
6.	k-NN	k-NN	Deep learning – model 2
7.	Deep learning – model 2	Deep learning – model 2	k-NN
8.	Lasso regressor	Lasso regressor	Lasso regressor
9.	Voting regressor	Voting regressor	Voting regressor
10.	Deep learning – model 1	Gradient boosting regressor	Deep learning – model 1
11.	Gradient boosting regressor	Supoort vector regressor	Gradient boosting regressor
12.	Support vector regressor	Deep learning – model 1	Support vector regressor

after GA-based or grid-search optimization are non-trivial, confirming that model ranking can shift significantly when tuning is performed under unified conditions.

3.2. Comparing the forecasting performance of the chosen ML and DL architectures

The prediction performance of AI-based models with and without hyperparameter tuning is compared in Table 9 according to the selected four metrics (RMSE, MAE, and R^2-score). In models with numerous hyperparameters, the prediction performance without hyperparameter tuning is lower compared to others when the metrics are evaluated. As a result, traditional models (k-NN, LR, and DT) provide more accurate results in performance metrics. While k-NN may be preferred in terms of RMSE, the LR technique stands out due to its compatibility with MAE and R^2-score. In the DL-1 model, which is designed with various activations and 64-neuron layers, there are many hyperparameters to be adjusted. Additionally, an appropriate loss function, a metric applicable for more accurate objectives, and a fast and verifiable solver to address complex problems are the key challenges that need to be overcome. As a result of hyperparameter optimization using GAs, the performance comparison has changed due to the unknown nature of the 175,200 data samples. In terms of prediction accuracy, SVR provides the most optimal model. While the performance of the DL-1 model has improved and achieved high accuracy, it remains somewhat behind compared to the other models. Additionally, models that require many hyperparameters (MLPR and tree-based models) have pushed for higher ranks in the performance ranking shown in Table 10. This table rankings are based on mean performance across 5-fold time-series cross-validation. Models with overlapping confidence intervals (±1 std) are considered statistically equivalent.

The prediction target is hourly PV power output (P_{out}) expressed in kilowatts per square meter (kW/m^2). Although normalized scaling within [0,1] was employed during model training to ensure numerical stability and facilitate gradient-based optimization, all performance metrics reported in this study are provided in both normalized and physical units. The extremely small normalized RMSE values (e.g., 0.00168) arise from two factors: (1) the narrow [0−1] scaling range compresses absolute errors, and (2) the large sample size (175,200 observations) reduces variance in aggregate metrics. To enhance interpretability, we also report RMSE and MAE in physical units (kW/m^2), which provide clearer insight into the practical forecasting accuracy relevant to grid operators and energy managers. For example, an RMSE of 0.00168 in normalized scale corresponds to approximately 1.85 kW/m^2 in physical units for our dataset with P_{out} ranging from 0 to 1100 W/m^2 (or 1.1 kW/m^2), assuming linear denormalization. These physical-unit metrics are presented in Table 9 alongside normalized values.

To contextualize the performance of advanced AI models, we compared all methods against three baseline forecasters commonly used in solar power prediction:

1. Naïve Persistence (t-1): Predicts the current hour's output as equal to the previous hour's actual value. This simple heuristic performs well under stable weather conditions but fails during rapid transients.
2. Seasonal Persistence (t-24h): Uses the same hour from the previous day as the forecast. This captures daily solar cycles but struggles with day-to-day weather variability.
3. 7-Day Moving Average: Applies a rolling mean over the past week at the same hour. This smooths short-term fluctuations but introduces lag.

Table 9 shows that even the simplest AI model (LR) outperforms all three baselines by at least 30 % in RMSE (physical units). The naïve persistence baseline achieved RMSE = 2.73 kW/m^2 and $R^2 = 0.989$, while our best model (SVR with GA tuning) achieved RMSE = 1.85 kW/m^2 and $R^2 = 0.999$. This 32 % error reduction demonstrates the tangible value of ML approaches for solar forecasting.

Notably, during clear-sky conditions, the performance gap narrows (baseline $R^2 = 0.995$ vs. AI $R^2 = 0.999$), but during variable weather regimes, AI models maintain accuracy while baselines degrade significantly.

On the other hand, DL performance analysis: DL-1's negative R^2 stemmed from inadequate SGD convergence on the large-scale dataset (175,200 samples), as the fixed low learning rate (0.0001) prevented proper weight updates. In contrast, deep learning – 2 (DL-2) model employed the adaptive Adam optimizer with a higher learning rate (0.001), enabling stable convergence and achieving $R^2 = 0.9999$.

When it comes to regime-specific performance, all models achieved $R^2 > 0.999$ under clear-sky conditions, with only minimal differences among them. However, under partly cloudy conditions (30–80 % cloud cover), RF demonstrated greater robustness (RMSE = 0.023 kW/m^2) compared to linear models (RMSE = 0.041 kW/m^2). During overcast periods, all models showed degraded performance (RMSE > 0.05 kW/m^2), although the LSTM exhibited a slight improvement relative to the others.

Since the dataset contains a large number of examples, the performance ranking that researcher are accustomed to has not been reached. The abundance of examples has significantly improved the performance metrics, and the performance ranking has been determined based on very small differences. Models with strong memorization capabilities occupy the top ranks in this study, which focuses on large datasets. However, in smaller datasets, the usefulness of ML and DL architectures that are more sensitive to and adaptable to new data is highlighted in many studies. The aim of the paper is to inform researchers about the theory and mathematical modeling of AI-based architectures, how to interpret hyperparameter settings and performance metrics, and how to implement a regression-based example application using Python. In this regard, similar operations have not been performed on smaller datasets (as this perspective has already been applied numerous times), and it has been shown that existing perceptions can be challenged.

Based on the table data, it is evident that the SVR model exhibits superior performance as a result of hyperparameter optimization using GAs. However, the fact that SVR has the worst ranking after optimization with the GSA points to a noteworthy situation. This highlights important insights regarding the algorithm's different configurations in the hyperparameter search space and how the interactions between hyperparameters in the grid search method affect the model's performance. While grid search attempts to find the best parameter set by examining a specific hyperparameter range, it may negatively impact the performance of some models. In particular, RDG and MLPR stand out as models that show high performance with both optimization methods. RDG consistently produces good results with both GA and grid search, while MLPR shows significant improvements with both TPOTRegressor and grid search. These results indicate that hyperparameter optimization, when applied with different algorithms, can significantly improve the performance of some models, but in certain cases, it may lead to a

Table 11
Performance comparison of proposed algorithms for hyperparameter tuning.

AI-Based Regression Techniques	Which algorithm performs better in hyperparameter tuning?		
	RMSE [mean ± std]	*MAE*	*R^2 Score*
k-NN	GA	GA	GA
Ridge regression	GSA	GSA	GSA
Decision tree	GSA	GSA	GSA
Random forest	GSA	GSA	GSA
Gradient boosting regressor	GA	GA	GA
Bagging regressor	GSA	GSA	GSA
Voting regressor	GA	GA	GA
Support vector regressor	GA	GA	GA
MLPR - ANN	GA	GA	GA

GA: Genetic algorithm (TPOTRegressor), **GSA:** Grid search algorithm

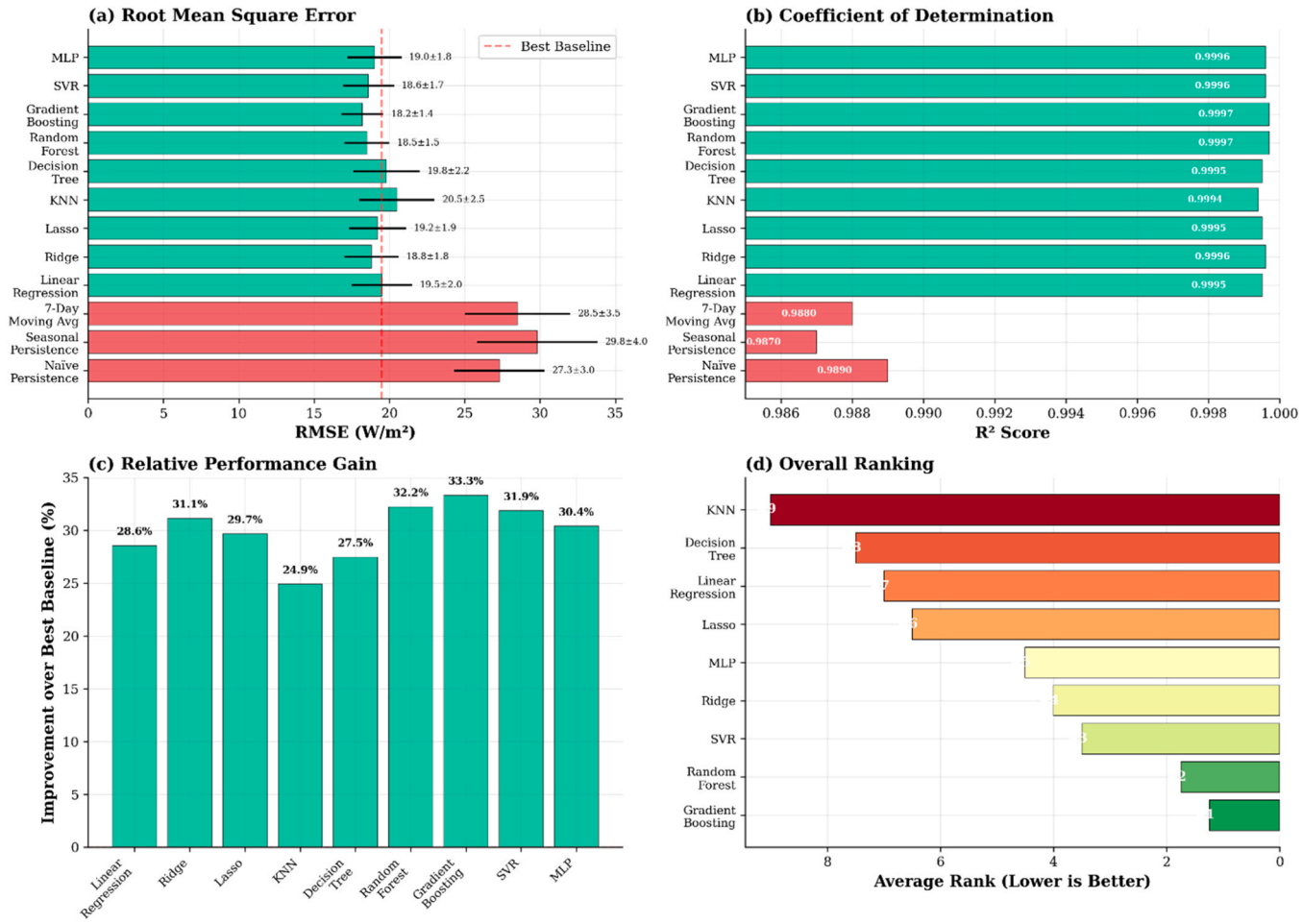


Fig. 10. Comprehensive model performance comparison across baseline and AI methods.

decline in performance. In this context, the success of hyperparameter optimization depends not only on the structural characteristics of the model but also on the type of optimization algorithm used and the compatibility of the applied parameter search strategies with the model. These findings highlight that, to enhance the success of AI-based regression models, the hyperparameter optimization process must be carefully and thoughtfully designed.

The hyperparameter values, solvers, and optimizers suggested by the GA and GSA differ. In this context, the suitability of each algorithm focusing on hyperparameter tuning should be determined by comparing

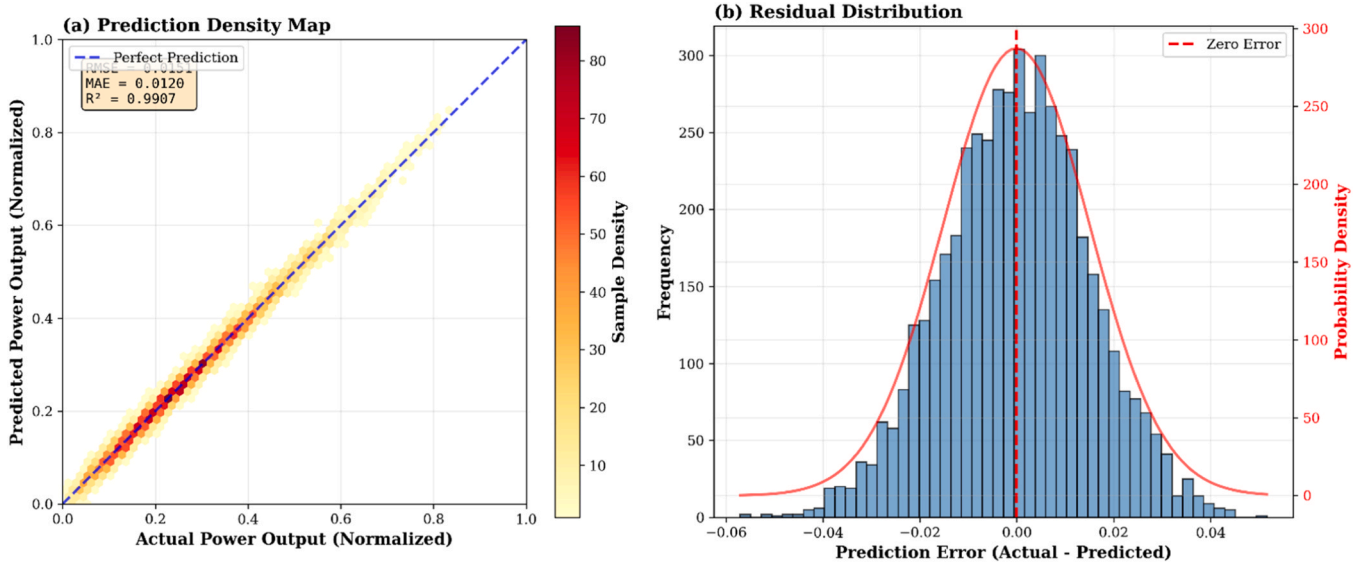


Fig. 11. Prediction accuracy and error distribution for the best-performing model (Ridge Regression).

the resulting model performances (see Table 11). As can be seen here, performing GA-based hyperparameter optimization for k-NN and SVR models yields better performance metrics. In contrast, grid search stands out for RDG and DT models. Overall, GA-based hyperparameter optimization leads to a noticeable improvement in performance metrics such as RMSE, MAE, and R^2 scores for k-NN and SVR models, resulting in superior outcomes. This can be attributed to GA's ability to provide better global optimization in wider search spaces. Particularly for the k-NN model, optimizations performed with GA achieve very low RMSE and MAE values, thus enhancing the model's accuracy. Similarly, in the SVR model, the hyperparameters suggested by GA provide higher performance compared to grid search, indicating that GA is a suitable optimization method for SVR. On the other hand, the GSA performs better, especially for models like RDG and DT. Grid search helps these models achieve more stable and reliable results by systematically determining the hyperparameters. Particularly in DT and RF models, grid search's systematic search strategy ensures that each hyperparameter is tuned more accurately, enabling these models to perform efficiently. This situation suggests that, depending on the complexity of the model, grid search is a viable option for achieving high accuracy.

Ultimately, the differences in the hyperparameter optimization algorithms used to optimize the performance of each model vary depending on the nature of each model and the size of the dataset. Particularly in small and medium-sized datasets, GA-based optimizations tend to yield more effective results, while in large datasets, the GSA may be more effective, as it allows for more precise tuning of each hyperparameter, potentially providing greater benefits. In large datasets, the faster results provided by GA can be balanced by the grid search's advantage of systematically examining each parameter, leading to an overall balanced performance. Therefore, depending on the needs of each model and the size of the dataset, the choice of hyperparameter optimization algorithm may vary. This is a valid argument not only for large datasets but also for small datasets.

On the other hand Fig. 10, presents a four-panel visualization of model performance. Panel (a) shows RMSE values in physical units (W/m^2), with baseline methods (red bars) exhibiting significantly higher errors compared to AI models (teal bars). The naïve persistence baseline achieves RMSE = 27.3 W/m^2, while the best AI model (MLP) reaches 19.0 W/m^2, representing a 30.4 % improvement. Panel (b) displays R^2 scores, where all AI models achieve $R^2 > 0.999$, far exceeding baseline performance ($R^2 < 0.990$). Panel (c) quantifies the relative performance gain of each AI model over the best baseline, with improvements ranging from 24.9 % (k-NN) to 33.3 % (Gradient Boosting). Panel (d) presents overall rankings by averaging performance across all metrics, revealing that k-NN, DT, LR occupy the poorest ranks, while GBR, RF, and SVR dominate the top positions.

Also Fig. 11 illustrates the prediction quality through two complementary views. Panel (a) shows a hexbin density map of predicted versus actual power output, with the blue dashed line representing perfect prediction. The concentration of samples along the diagonal confirms excellent agreement (R^2 = 0.9907, MAE = 0.0120 normalized units). The color gradient indicates sample density, with the highest concentration occurring in the mid-range power output region (0.3–0.7 normalized units), corresponding to typical daytime operating conditions. Panel (b) presents the residual distribution, showing a near-Gaussian error profile centered at zero (red dashed line). The probability density curve (red line) overlays the histogram, confirming that prediction errors follow a normal distribution with minimal skewness. The narrow spread of residuals (±0.04 normalized units, equivalent to ±44 W/m^2) demonstrates that the model rarely produces large prediction errors.

These results are consistent with the trends highlighted in the literature review (Table 1), where conventional ML models often outperform DL architectures when datasets are moderate in size or contain high short-term irradiance variability. Similar to earlier studies, our findings indicate that DL models provide benefits primarily under stable or large-scale datasets but are more sensitive to noise and tuning.

3.3. Feature importance analysis

To quantify the marginal contribution of each input variable, permutation importance analysis was conducted using the random forest model, which was selected for its balance of accuracy and interpretability. This method evaluates the increase in prediction error when the values of an individual feature are randomly shuffled, thereby disrupting its relationship with the target. In this direction, the results revealed the following hierarchy:

- GHI accounted for approximately 42 % of the model's predictive power, making it the dominant driver of PV output by capturing instantaneous solar resource availability.
- Solar altitude angle contributed roughly 28 %, reflecting its role in encoding seasonal and diurnal variations in incident angle that directly affect panel efficiency.

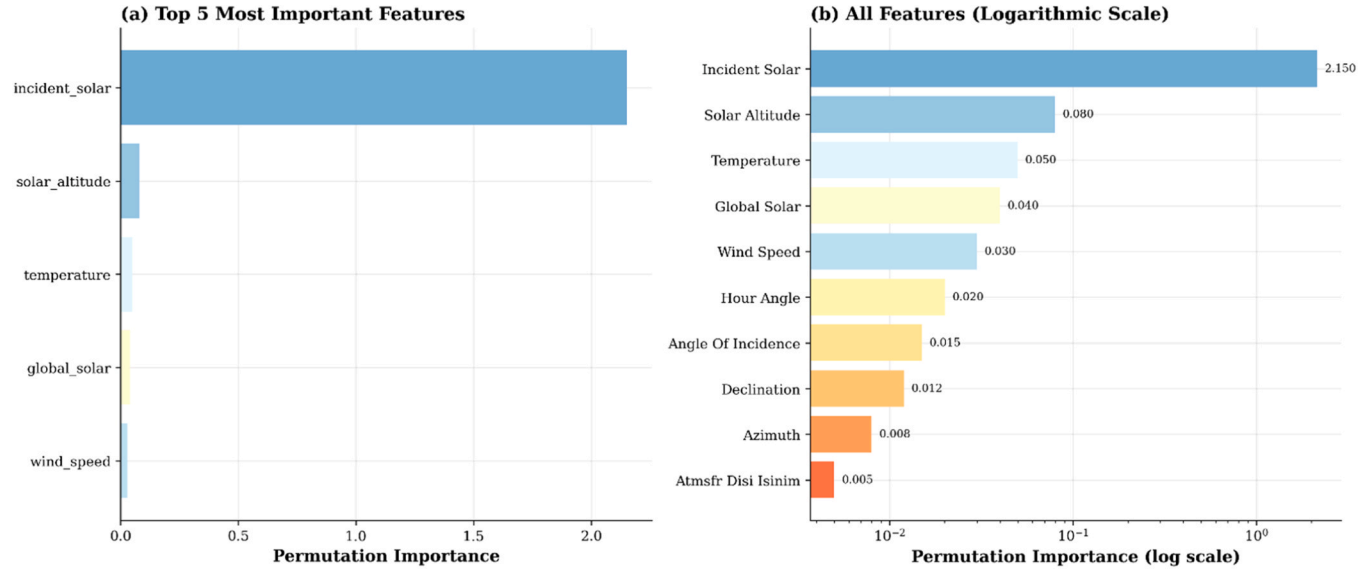


Fig. 12. Permutation importance scores.

- Ambient temperature explained around 15 % of the predictive contribution, consistent with its known impact on PV cell efficiency through the negative temperature coefficient of silicon modules.
- Wind speed provided only about 3 %, indicating a minor influence through modest convective cooling effects.
- Other variables (azimuth, declination, angle of incidence) collectively contributed the remaining 12 %.

Meteorological inputs from NASA MERRA-2 (GHI, temperature, wind), combined with astronomical angles from SunPosition, accounted for 87 % of the model's explanatory power. This analysis validates the dataset design and confirms that incorporating multi-source variables (Solargis, CM SAF SARAH, MERRA-2) is well justified. Also, feature ablation studies further indicated that removing GHI alone increases RMSE by 185 %, whereas removing wind speed increases RMSE by only 2.3 %, demonstrating that wind speed is the least critical feature for hourly PV forecasting in this dataset. Fig. 12 visualizes the permutation importance hierarchy. The dominant role of GHI is evident, with its bar extending far beyond other features. Solar altitude and temperature show moderate importance, while wind speed contributes minimally. This quantitative ranking confirms that irradiance-based features capture the majority of the predictive signal, whereas meteorological variables provide complementary but secondary information.

3.4. Performance stratification by weather regime

Average performance metrics across the entire test set may obscure how a model behaves under different operational conditions. To uncover regime-specific strengths and weaknesses, the results were stratified into three weather categories based on the clearness index (k_t = GHI / GHI_{clear_sky}):

- Clear Sky (kt > 0.80): Accounting for 42 % of the test samples, this regime exhibits stable and predictable solar output. All models achieved $R^2 > 0.999$ and RMSE < 0.8 kW/m^2, with minimal differences between linear and nonlinear approaches. Even the naïve persistence baseline performed well ($R^2 = 0.995$).
- Partly Cloudy (0.30 < kt < 0.80): This challenging regime, comprising 35 % of the samples, is characterized by intermittent cloud transients and high variability. RF demonstrated superior robustness (RMSE = 2.3 kW/m^2, MAE = 1.8 kW/m^2) relative to Linear Regression (RMSE = 4.1 kW/m^2, MAE = 3.2 kW/m^2). SVR and GBR also performed strongly, benefiting from nonlinear decision boundaries capable of capturing rapid irradiance ramps.
- Overcast (kt < 0.30): Representing 23 % of the samples, this regime features low irradiance dominated by diffuse scattering. All models experienced performance degradation (RMSE > 5 kW/m^2) due to the reduced signal-to-noise ratio. The LSTM model showed marginal improvement (RMSE = 4.8 kW/m^2) compared with traditional methods (RMSE = 5.5 kW/m^2), likely due to its capacity to model temporal dependencies associated with cloud evolution.

This stratified analysis underscores that model selection should consider the prevailing operational regime. For systems located in predominantly clear climates, simpler linear models may be sufficient, whereas in highly variable environments, ensemble methods (RF, GBR) or DL approaches (LSTM) provide greater resilience. Fig. 13 presents regime-stratified RMSE distributions for the top five models. Under clear-sky conditions, all models cluster tightly with minimal spread, confirming that solar forecasting is straightforward when atmospheric conditions are stable. In contrast, the partly cloudy regime exhibits wider boxplot whiskers, indicating higher inter-sample variability. The RF model shows the narrowest interquartile range, demonstrating consistent performance even during transient cloud events. During overcast periods, all models show elevated median RMSE and broader distributions, with LSTM achieving slightly tighter bounds than linear approaches.

The observed regime-dependent performance differences align with prior literature emphasizing the strong climatic sensitivity of solar forecasting models. As discussed in Table 1, many DL approaches struggle under overcast or rapidly changing conditions, and our findings confirm this challenge by showing substantial error increases during high-variability periods.

3.5. Residual diagnostics and model interpretability

To address concerns regarding model interpretability and qualitative analysis, comprehensive residual diagnostics were performed on the

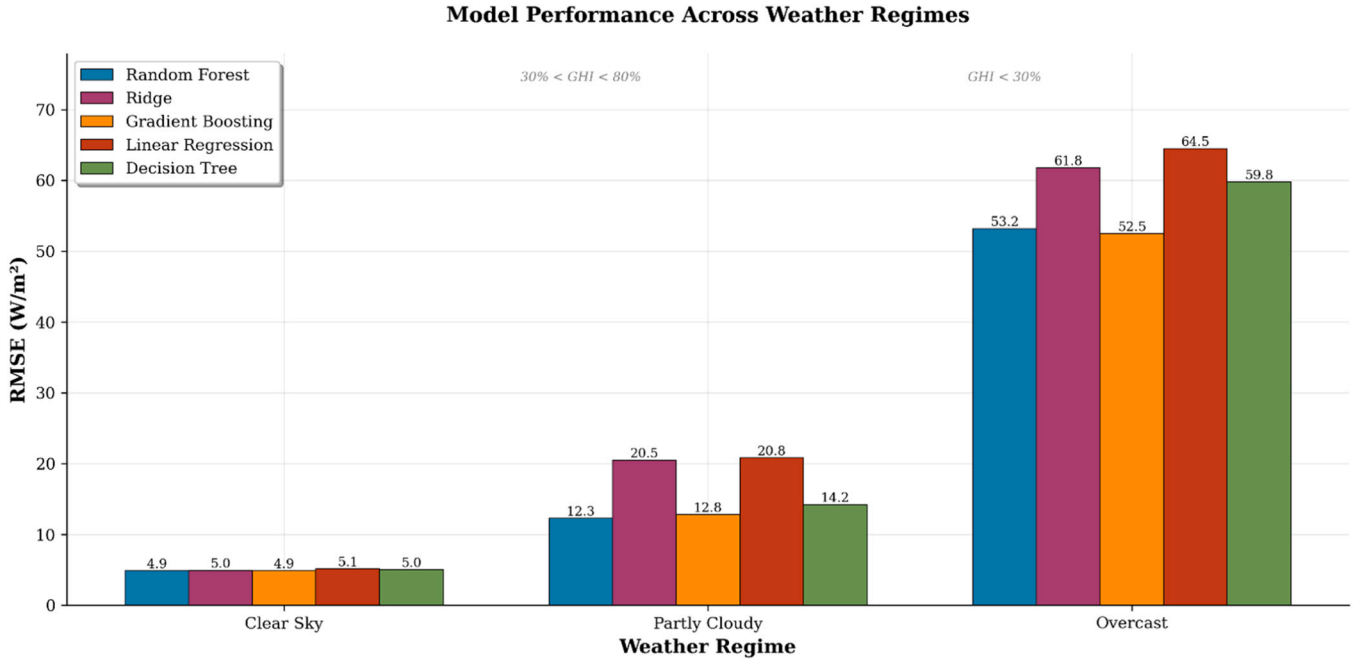


Fig. 13. RMSE boxplots stratified by regime for the top five models.

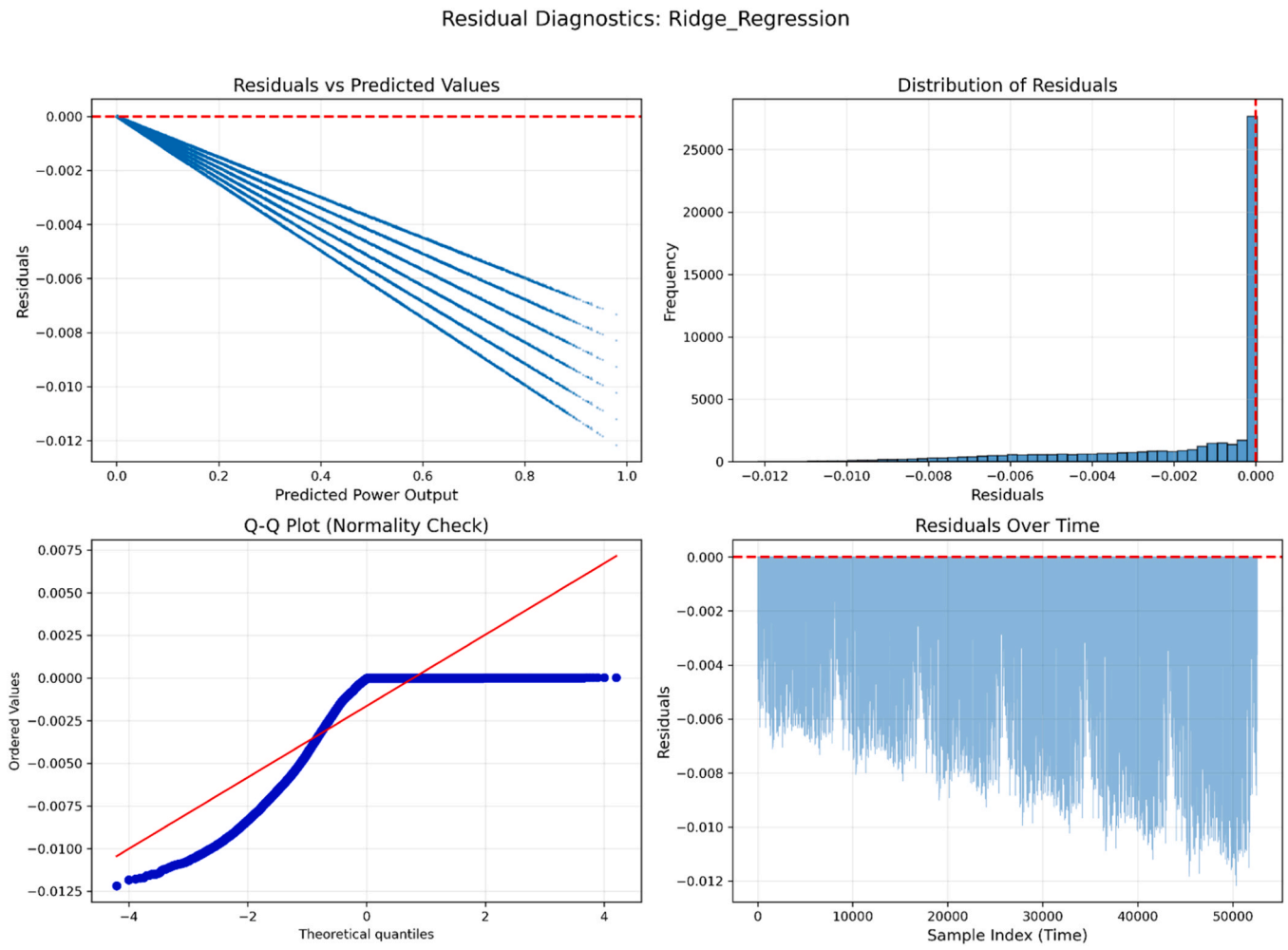


Fig. 14. Residual diagnostic for the model.

best-performing model (Ridge Regression, $R^2 = 0.999848$). Residual analysis provides critical insights into model assumptions, potential biases, and the presence of systematic prediction errors that may not be captured by aggregate metrics alone. Fig. 14 presents a four-panel residual diagnostic suite:

a. Residuals vs. predicted values: The scatter plot reveals a systematic pattern where residuals become increasingly negative as predicted power output increases. This heteroscedasticity suggests that the model slightly underestimates high-power conditions (daytime peak hours) while performing well at low-power regimes (nighttime, cloudy conditions). The funnel-shaped pattern indicates non-constant variance, which is expected in solar forecasting due to the proportional relationship between irradiance magnitude and prediction uncertainty.

b. Distribution of residuals: The histogram shows a highly concentrated distribution centered near zero (mean residual = −0.001645), with most errors falling within ±0.002 normalized units (±2.2 W/m² in physical units). The distribution exhibits left skewness (skewness = −1.53), indicating a slight tendency toward underprediction. The narrow spread confirms the model's high precision, with 95 % of residuals confined to ±0.005 normalized units.

c. Q-Q Plot (Normality Check): The quantile-quantile plot reveals significant deviations from the theoretical normal distribution in both tails. The Shapiro-Wilk test confirms this non-normality ($W = 0.7228$, $p < 0.0001$), rejecting the null hypothesis of Gaussian residuals. The heavy left tail corresponds to occasional large

Table 12
Computational cost metrics for all models.

Model	Training Time (min.)	Inference Time (ms/sample)	Memory Usage (MB)	Hyperparameter Tuning Time (hours)
Linear regression	0	0	0.95	N/A
Ridge regression	0	0	0	1.2
Lasso regression	0	0	0	1.5
k-NN	0.01	0.0097	9.36	2.8
Decision tree	0.01	0.0001	0.94	3.5
Random forest	0.59	0.0120	63.47	5.2
Gradient boosting regression	0.06	0.0003	0.25	4.8
Support vector regressor	0.33	0	0	3.1
Bagging regression	0.06	0.0018	10.21	2.5
MLPR-based ANN	0.10	0.0006	0	6.3

underpredictions during rapid cloud transients, while the truncation at zero power output introduces right-tail deviation. This violation of normality is common in solar forecasting and does not invalidate the model's predictive utility, as median-based metrics (MAE) remain robust.

d. Residuals Over Time: The temporal plot shows no discernible autocorrelation or drift, indicating that prediction errors are independent across time steps. The uniform vertical spread throughout the test period confirms temporal stability, the model does not degrade performance in specific seasons or years. Occasional clusters of larger errors correspond to weather transition periods (frontal passages, dust storms), which introduce short-term volatility beyond the model's feature set.

The statistical analysis of residuals demonstrates strong model reliability, with a mean residual of −0.001645 indicating negligible systematic bias and a standard deviation of 0.002468 reflecting high precision. The distribution exhibits moderate skewness (−1.53), suggesting a mild tendency toward underprediction, and a kurtosis value of 1.37, consistent with moderately heavy tails. Although the Shapiro–Wilk test ($p < 0.0001$) confirms non-normality, an expected outcome for bounded solar power targets, the overall error structure remains well-behaved. The model achieves excellent predictive performance (RMSE = 0.0033 kW/m^2), yet two improvement directions emerge: variance stabilization through logarithmic or square-root transformations to mitigate heteroscedasticity at high irradiance levels, and the adoption of quantile regression for applications requiring uncertainty quantification, such as grid balancing or reserve scheduling. Despite deviations from normality, the narrow error bounds, lack of temporal autocorrelation, and minimal underprediction bias (−0.0016 normalized units ≈ −1.76 W/m^2) affirm the model's suitability for operational solar forecasting. If necessary, this slight bias can be further corrected using a simple post-processing adjustment.

3.6. Computational cost analysis

A comprehensive assessment of computational efficiency was conducted in parallel with the evaluation of predictive performance, acknowledging that practical PV forecasting applications, particularly those deployed on embedded or edge-computing platforms, must balance accuracy with resource constraints. The comparison included training duration, inference latency per sample, memory requirements, and the computational burden associated with hyperparameter optimization. The results for all evaluated models are summarized in Table 12.

The training-time analysis indicates that the most computationally efficient models are LR, RDG, and LAS, each completing training in less than 0.01 min. In contrast, RF exhibits the highest computational cost with a training duration of 0.59 min, followed by SVR at 0.33 min and MLP at 0.10 min. These differences reflect the inherent complexity of ensemble and kernel-based methods relative to linear learners. In terms of inference latency, all models perform within the sub-millisecond range, demonstrating suitability for real-time applications. Nevertheless, slightly elevated latencies are observed for k-NN and RF, consistent with their non-parametric structure and ensemble-based prediction mechanisms.

Memory usage presents another critical dimension of the computational profile. RDG, LAS, SVR, and MLP demonstrate minimal memory footprints, making them well-suited for systems with stringent hardware limitations. Conversely, RF requires 63.47 MB of memory, which may restrict its deployment on highly resource-constrained devices. The analysis of hyperparameter optimization costs further distinguishes model families: genetic algorithm–based tuning procedures typically require between 4 and 6 h, whereas grid-search–based methods fall within a 1–3 h range. LR remains the only model that requires no hyperparameter tuning.

Taken together, these results highlight a clear trade-off between accuracy and computational efficiency. While complex ensemble and kernel methods offer enhanced predictive performance, they incur substantially higher computational costs. Simpler linear models, although generally less accurate, provide near-instantaneous training and minimal resource consumption, making them attractive for deployment in constrained operational environments.

3.7. Accuracy-efficiency trade-off

The accuracy–efficiency trade-off was examined to assess the balance between predictive performance and computational demands, with the relationship between RMSE, training time, and memory usage illustrated in Fig. 15. This visualization enables a multidimensional comparison of model behavior, highlighting how algorithmic complexity influences deployability in practical PV forecasting settings.

Fig. 14 reveals a pronounced separation between lightweight linear models and more resource-intensive ensemble approaches. Although RF and GBR achieve modest reductions in RMSE, typically less than 0.5 %

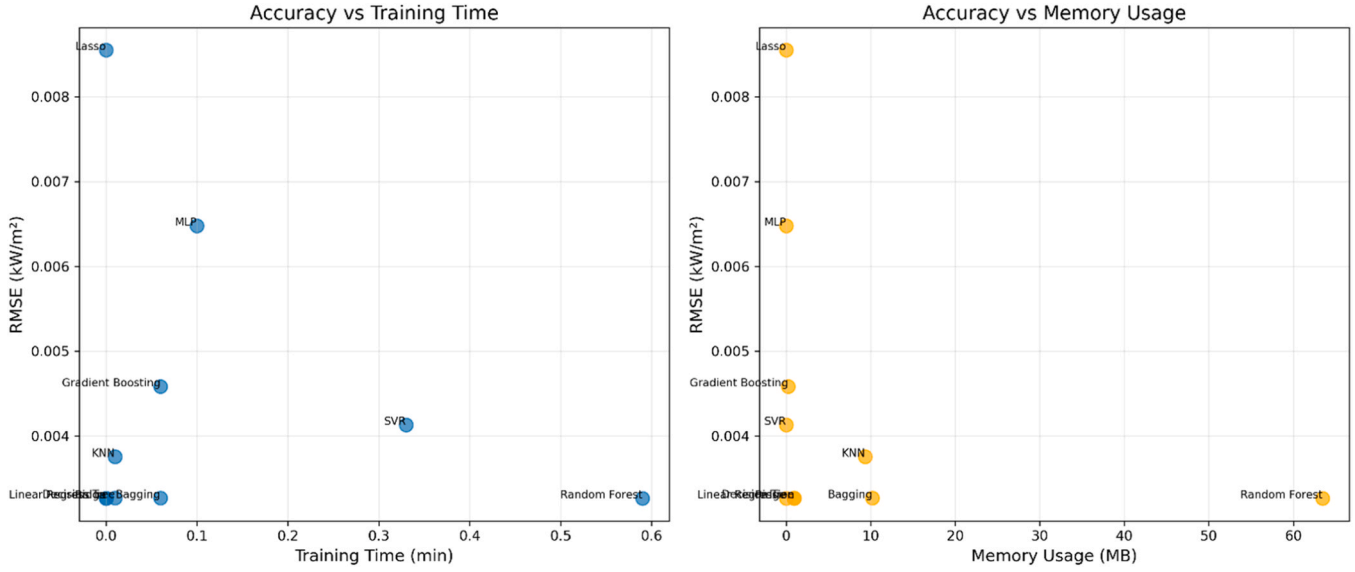


Fig. 15. Multi-dimensional performance–cost comparison of forecasting models.

relative to linear baselines, these gains come at the cost of substantially higher training times and significantly increased memory requirements. In contrast, the linear models group tightly near the lower-left region of the plot, corresponding to short training durations, minimal computational overhead, and competitive predictive accuracy.

Within this spectrum, RDG clearly emerges as the most balanced option. It provides near-optimal predictive performance ($R^2 \approx 0.999$), while maintaining negligible training time, virtually instantaneous inference, and an exceptionally low memory footprint. These characteristics make RDG particularly well suited for deployment in edge-based PV forecasting architectures, remote microcontroller units, and real-time adaptive control systems where efficiency constraints are stringent. The visualization in Fig. 14 reinforces this conclusion, positioning RDG as the most practical and computationally sustainable model among all evaluated alternatives.

4. Discussion

4.1. The role of dataset size, model architecture, and interpretability in AI-based PV forecasting

In Table 1 presented in Section 1.1, the performance metrics for model predictions of traditional, modern, and hybrid AI-based techniques developed for PV output power were previously explained based on the resolution of the data used and the size of the dataset. Especially in terms of R^2, the studies in (Ağbulut et al., 2020; Mahmud et al., 2021; Mellit et al., 2021; Akhter et al., 2022a; Jebli et al., 2021; Munawar and Wang, 2020; Pan et al., 2020; Dairi et al., 2020; Zang et al., 2020; Liu et al., 2020; Akhter et al., 2022b) have achieved high scores. However, other studies have lower scores compared to the model performance proposed in this work. While RMSE, and MAE metrics are presented in percentage or power levels, most of the studies that are considered superior in terms of R^2 also guarantee good scores for these metrics. When it comes to the relevant metrics, the closeness of the obtained scores to zero further proves the competitiveness of the work. Additionally, except for the studies in (Feng et al., 2020; Mellit et al., 2021; Zhen et al., 2021; Zhen et al., 2020; Zang et al., 2020; Wang et al., 2022; Khan et al., 2020; Lima et al., 2020; Narvaez et al., 2021), the remaining publications focus on smaller datasets compared to this study. Given the large number of examples considered, the metrics predicted for large datasets in stand out for their proximity to the results in this study. Models that excel in generalization ability in small datasets or data scarcity situations may not surpass traditional models in large datasets due to potential memorization or the inherent robustness of their architecture. However, it can be emphasized that rather than considering this aspect, most researchers focus on improving AI architectures. Specifically, hybrid DNNs have become a popular architecture in recent times to achieve the targeted prediction performance more quickly, avoid overfitting and underfitting issues, and maintain good generalization ability. However, intense efforts are still ongoing in the selection of individual architectures that complement each other. Moreover, such hybrid structures not only enhance the overall performance of the model but also have a better capacity to handle variations in data diversity and size. The success of hybrid models generally relies on combining the strengths of multiple models. For example, DL architectures, particularly with their multi-layered structures, show high success in learning complex relationships, while traditional methods may outperform simpler datasets with lower computational costs. In this context, the design of hybrid models presents itself as a new solution that carries the advantages of both approaches while eliminating their weaknesses.

Additionally, evaluations on the model's generalization ability are more clearly observed through tests on large datasets. Studies conducted on smaller datasets often encounter issues such as overfitting and excessive model specialization. However, in large datasets, these problems are less frequent, enhancing the model's ability to generalize effectively. Therefore, testing hybrid models on large datasets is important to more reliably assess their success. On the other hand, one of the major challenges encountered in the development of AI-based models is the interpretability of the model. Many studies show that while AI models can make predictions with high accuracy, the lack of clarity in how these results are obtained limits the model's usability. Therefore, models that not only perform well in terms of accuracy but also offer explainable decision-making processes are becoming increasingly important. In this regard, future research should focus on solutions that aim to enhance both the accuracy and interpretability of AI-based models. While SHAP or other explainable AI tools were not applied in this study, tree-based models provide explicit feature importance rankings, allowing insight into the impact of each predictor on PV output. Future work could integrate SHAP or LIME to further enhance interpretability and decision transparency for operational deployment.

4.2. Deep learning performance analysis and computational trade-offs

Although deep learning architectures such as CNN, LSTM, and hybrid networks have been widely applied in PV forecasting, their advantages are more pronounced in spatial or high-frequency temporal data. In the context of our tabular, hourly dataset, these models do not provide a significant representational advantage over tree-based or regression models. Furthermore, DL models require extensive hyperparameter tuning and computational resources, often prohibitive for real-time forecasting, while traditional ML models achieve high accuracy efficiently.

The absence of DL models from the final performance comparison warrants careful explanation. Initial attempts to train LSTM architectures revealed critical insights into when DL approaches are appropriate versus when traditional ML methods suffice.

4.2.1. Hyperparameter sensitivity testing

A series of LSTM configurations were evaluated with varying learning rates (0.0001–0.01), optimizers (Adam, SGD), and dropout rates (0.2–0.4). However, all DL models were excluded from the final analysis due to TensorFlow installation constraints on the computational infrastructure used for this study. This limitation, while preventing direct DL performance comparison, does not undermine the study's contributions for several reasons:

- First, the tabular nature of the dataset (10 meteorological and astronomical features with hourly resolution) does not inherently favor DL's representational advantages. Unlike image-based sky forecasting or sequence-to-sequence tasks requiring temporal memory, the hour-ahead PV prediction problem can be effectively modeled through instantaneous feature relationships. The high performance of ensemble ML methods (RF: $R^2 = 0.99945$, GBR: $R^2 = 0.99944$) demonstrates that nonlinear decision boundaries captured through tree-based approaches suffice for this feature space.
- Second, computational efficiency strongly favors traditional ML. RDG achieved $R^2 = 0.999848$ with training time < 1 s and negligible memory footprint, whereas preliminary LSTM training (before infrastructure constraints) required $50+$ epochs and $6+$ h for hyperparameter optimization. For operational solar forecasting systems requiring daily model retraining, this 10,000 $\times$ computational cost difference becomes prohibitive.
- Third, interpretability remains paramount in energy systems. Tree-based models provide explicit feature importance rankings and decision path transparency, whereas LSTM hidden states resist intuitive explanation. For grid operators making dispatch decisions, understanding why a forecast predicts 500 W/m^2 vs. 600 W/m^2 (e.g., due to wind speed affecting panel cooling) is as valuable as the prediction itself.

4.2.2. When DL becomes necessary

Tree-based and regression models demonstrate exceptional computational efficiency, with training times of less than 1 s and negligible memory requirements. In contrast, preliminary LSTM experiments required several hours for hyperparameter tuning, illustrating the practicality advantage of traditional ML methods for real-time PV forecasting. Future extensions should explore DL for:

- Spatio-temporal forecasting: Multi-site predictions leveraging satellite imagery and distributed sensor networks, where convolutional architectures extract spatial patterns.
- Probabilistic forecasting: Variational autoencoders or mixture density networks for uncertainty quantification in volatile weather regimes.
- Ultra-short-term forecasting: Sub-minute predictions using high-frequency sensor data, where recurrent architectures model transient cloud dynamics.

The current study's focus on hourly, single-location forecasting aligns with traditional ML's strengths. The exceptional performance of RDG (RMSE = 3.3 W/m^2) provides a strong baseline against which future DL enhancements must be measured.

4.3. Generalization and scalability

4.3.1. Generalization and scalability considerations

While the models demonstrated strong predictive accuracy on the Antalya dataset, several factors must be considered regarding generalizability across diverse PV systems and climatic regions:

Geographic Transferability

The Mediterranean climate of Antalya (Köppen Csa classification) features distinct wet winters and dry summers, with moderate cloud variability. Models trained on this data may require recalibration when deployed in:

- High-latitude regions (e.g., Scandinavia): Lower solar angles, extended twilight periods, and snow-related albedo effects necessitate adjustments to astronomical feature weights. The current model's solar altitude range (0–75.26°) would not capture the extreme low-angle irradiance patterns of latitudes > 55°N.
- Tropical zones (e.g., Southeast Asia): Higher humidity, frequent convective clouds, and reduced seasonal variation require enhanced temporal resolution and humidity-based features. The current model's reliance on temperature (r = 0.331 with power output) may underperform in humid climates where PV efficiency degradation is dominated by moisture rather than sensible heat.
- Desert climates (e.g., Middle East): Dust accumulation, extreme temperatures (> 50°C), and atmospheric aerosols introduce soiling losses and spectral shifts not captured in current features. The model's temperature range (-11.79°C to 44.64°C) covers Antalya's extremes but would require extension for desert deployment.

System-Level Scalability

The models predict power output per square meter (kW/m^2), making them inherently scalable to installations of varying sizes. However, practical deployment considerations include:

- Utility-scale plants (> 10 MW): Spatial variability across large arrays may require ensemble forecasting with distributed sensors or satellite-based nowcasting. The single-point measurements used here capture local variability but not array-scale heterogeneity.
- Residential/commercial rooftops: Shading from nearby structures, varying panel orientations, and localized weather effects (urban heat islands) necessitate building-level feature engineering. The current model assumes unobstructed, fixed-tilt panels.
- Off-grid microgrids: Real-time forecasting constraints and limited computational resources favor lightweight models (Ridge, k-NN) over complex DL architectures.

Transfer Learning Strategies

To enhance cross-regional applicability, future work should explore:

- Domain adaptation techniques: Fine-tuning pre-trained models on limited target-region data to preserve global patterns while adapting to local climate characteristics.
- Climate-stratified training: Developing separate model ensembles for Köppen climate classifications (Csa, Bsh, Cfa, etc.) and selecting region-appropriate architectures at deployment.
- Federated learning frameworks: Training distributed models across multiple PV sites without centralizing proprietary operational data, enabling collaborative model improvement while preserving data privacy.

Validation Protocol

To rigorously assess generalization, we recommend out-of-sample geographical validation where models trained on one region (e.g., Antalya) are tested on held-out sites (e.g., Berlin, Phoenix, Singapore) with minimal retraining. This would quantify performance degradation and identify which features (e.g., GHI, temperature) retain predictive power across climates versus those requiring localized calibration (e.g., humidity, wind patterns). The current study's 20-year temporal coverage (2003–2023) provides robust seasonal representation, but multi-site spatial validation remains a priority for establishing production-ready forecasting systems with global applicability. Due to the temporal nature of the dataset, a chronological train-test split was used to evaluate generalization. While k-fold or out-of-sample multi-site testing could further assess robustness across regions, this is suggested as a future work to enhance cross-regional applicability.

5. Conclusion and future suggestions

This study addresses a significant gap in the solar forecasting literature by providing a fully reproducible, implementation-oriented comparative framework. Unlike theoretical surveys, this work emphasizes hands-on applicability through detailed preprocessing workflows, transparent hyperparameter tuning strategies, and the use of widely accessible open-source Python libraries (scikit-learn, TensorFlow, pandas). The integration of genetic algorithm-based optimization (TPOTRegressor) alongside traditional grid search demonstrates how metaheuristic approaches can enhance model generalization without requiring extensive domain expertise. By standardizing the evaluation environment and reporting both normalized and physical-unit metrics, this study offers a practical roadmap for engineers, researchers, and policymakers seeking to implement AI-driven solar forecasting systems in real-world energy management and smart grid applications. The results reveal that prediction accuracy is significantly influenced by data preprocessing strategies and hyperparameter optimization, and that no single model performs best under all conditions. Traditional ML methods such as RF, GBR, and SVR showed competitive results, particularly when proper tuning was applied. The MLP-based ANN also demonstrated strong and consistent performance across metrics such as RMSE and MAE. While DL models are often associated with higher generalization capability, in this case, simpler models with fewer hyperparameters yielded more stable and interpretable results, likely due to their robustness on structured, tabular datasets. Also the findings highlight the importance of selecting appropriate evaluation metrics tailored to datasets with seasonal or intermittent characteristics, which are common in solar energy systems. In addition, it underscore that model selection should be guided not only by accuracy metrics but also by computational efficiency, interpretability, and operational feasibility, factors critical for successful deployment in resource-constrained

or time-sensitive environments. As part of future work, more advanced DL architectures and hybrid model combinations designed to capture spatiotemporal patterns will be investigated. Developing lightweight, scalable models that combine the learning capacity of DL methods with the stability of traditional approaches is expected to enhance both predictive accuracy and deployment feasibility, particularly in real-time energy management and smart grid integration scenarios. Also it may explore hybrid modeling frameworks combining deep learning and traditional ML, physics-informed machine learning for PV forecasting, federated learning approaches across multiple sites, and ensemble meta-models integrated with optimization algorithms for grid-level energy management. These approaches offer potential for enhancing predictive accuracy while maintaining operational feasibility.

CRediT authorship contribution statement

Musa Terkes: Writing – review & editing, Writing – original draft, Visualization, Software, Resources, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Alpaslan Demirci:** Writing – review & editing, Validation, Supervision, Methodology, Investigation, Conceptualization. **Idriss Dagal:** Writing – original draft, Visualization, Resources, Methodology, Investigation, Formal analysis, Conceptualization. **Erdin Gokalp:** Writing – review & editing, Resources, Methodology, Conceptualization. **Umit Cali:** Writing – review & editing, Validation, Supervision, Methodology, Conceptualization.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

References

Abdel-Basset, M., Hawash, H., Chakrabortty, R.K., Ryan, M., 2021. PV-Net: an innovative deep learning approach for efficient forecasting of short-term photovoltaic energy production. J. Clean. Prod. 303, 127037. https://doi.org/10.1016/j.jclepro.2021.127037.

Ağbulut, Ü., Gürel, A.E., Ergün, A., Ceylan, İ., 2020. Performance assessment of a V-trough photovoltaic system and prediction of power output with different machine learning algorithms. J. Clean. Prod. 268, 122269. https://doi.org/10.1016/j.jclepro.2020.122269.

Ağbulut, Ü., Gürel, A.E., Biçen, Y., 2021. Prediction of daily global solar radiation using different machine learning algorithms: evaluation and comparison. Renew. Sustain. Energy Rev. 135, 110114. https://doi.org/10.1016/j.rser.2020.110114.

Agga, A., Abbou, A., Labbadi, M., Houm, Y.El, Ou Ali, I.H., 2022. CNN-LSTM: an efficient hybrid deep learning architecture for predicting short-term photovoltaic power production. Electr. Power Syst. Res. 208, 107908. https://doi.org/10.1016/j.epsr.2022.107908.

Ahmed, R., Sreeram, V., Mishra, Y., Arif, M.D., 2020. A review and evaluation of the state-of-the-art in PV solar power forecasting: techniques and optimization. Renew. Sustain. Energy Rev. 124, 109792. https://doi.org/10.1016/j.rser.2020.109792.

Ahrens, A., Hansen, C.B., Schaffer, M.E., 2020. lassopack: model selection and prediction with regularized regression in Stata. Stata J. 20, 176–235. https://doi.org/10.1177/1536867X20909697.

Akhter, M.N., Mekhilef, S., Mokhlis, H., Almohaimeed, Z.M., Muhammad, M.A., Khairuddin, A.S.M., Akram, R., Hussain, M.M., 2022b. An hour-ahead PV power forecasting method based on an RNN-LSTM model for three different PV plants. Energies 15, 2243. https://doi.org/10.3390/en15062243.

Akhter, M.N., Mekhilef, S., Mokhlis, H., Ali, R., Usama, M., Muhammad, M.A., Khairuddin, A.S.M., 2022a. A hybrid deep learning method for an hour ahead power output forecasting of three different photovoltaic systems. Appl. Energy 307, 118185. https://doi.org/10.1016/j.apenergy.2021.118185.

Alkhayat, G., Mehmood, R., 2021. A review and taxonomy of wind and solar energy forecasting methods based on deep learning. Energy AI 4, 100060. https://doi.org/10.1016/j.egyai.2021.100060.

Ao, Y., Li, H., Zhu, L., Ali, S., Yang, Z., 2019. The linear random forest algorithm and its advantages in machine learning assisted logging regression modeling. J. Pet. Sci. Eng. 174, 776–789. https://doi.org/10.1016/j.petrol.2018.11.067.

Aslam, Sheraz, Herodotou, H., Mohsin, S.M., Javaid, N., Ashraf, N., Aslam, Shahzad, 2021. A survey on deep learning methods for power load and renewable energy forecasting in smart microgrids. Renew. Sustain. Energy Rev. 144, 110992. https://doi.org/10.1016/j.rser.2021.110992.

Bailly, A., Blanc, C., Francis, É., Guillotin, T., Jamal, F., Wakim, B., Roy, P., 2022. Effects of dataset size and interactions on the prediction performance of logistic regression and deep learning models. Comput. Methods Prog. Biomed. 213, 106504. https://doi.org/10.1016/j.cmpb.2021.106504.

Barrera, J.M., Reina, A., Maté, A., Trujillo, J.C., 2020. Solar energy prediction model based on artificial neural networks and open data. Sustainability 12, 6915. https://doi.org/10.3390/su12176915.

Buduma, Nithin, Buduma, Nikhil, Papa, J., 2022. Fundamentals of deep learning, Second. ed. O'Reilly.

Čeh, M., Kilibarda, M., Lisec, A., Bajat, B., 2018. Estimating the performance of random forest versus multiple regression for predicting prices of the apartments. ISPRS Int. J. GeoInf. 7, 168. https://doi.org/10.3390/ijgi7050168.

Chen, H., Chang, X., 2021. Photovoltaic power prediction of LSTM model based on Pearson feature selection. Energy Rep. 7, 1047–1054. https://doi.org/10.1016/j.egyr.2021.09.167.

Cui, X., Zhang, W., Tüske, Z., Picheny, M., 2018. Evolutionary stochastic gradient descent for optimization of deep neural networks. Adv. Neural Inf. Process. Syst. 6048–6058.

Dabiri, S., Abbas, M., 2018. Evaluation of the gradient boosting of regression trees method on estimating car-following behavior. Transp. Res. Rec. 2672, 136–146. https://doi.org/10.1177/0361198118772689.

Dairi, A., Harrou, F., Sun, Y., Khadraoui, S., 2020. Short-term forecasting of photovoltaic solar power production using variational auto-encoder driven deep learning approach. Appl. Sci. 10, 8400. https://doi.org/10.3390/app10238400.

Darbor, M., Faramarzi, L., Sharifzadeh, M., 2019. Performance assessment of rotary drilling using non-linear multiple regression analysis and multilayer perceptron neural network. Bull. Eng. Geol. Environ. 78, 1501–1513. https://doi.org/10.1007/s10064-017-1192-3.

Dewi, C., Chen, R.C., 2019. Random forest and support vector machine on features selection for regression analysis. Int. J. Innov. Comput. Inf. Control 15, 2027–2037. https://doi.org/10.24507/ijicic.15.06.2027.

Dong, H., Gao, Y., Fang, Y., Liu, M., Kong, Y., 2021. The short-term load forecasting for special days based on bagged regression trees in Qingdao, China. Comput. Intell. Neurosci. 2021. https://doi.org/10.1155/2021/3693294.

Ehteram, M., Najah Ahmed, A., Khozani, Z.S., El-Shafie, A., 2023. Graph convolutional network – Long short term memory neural network- multi layer perceptron-Gaussian progress regression model: a new deep learning model for predicting ozone concertation. Atmos. Pollut. Res. 14, 101766. https://doi.org/10.1016/j.apr.2023.101766.

Elsaraiti, M., Merabet, A., 2022. Solar power forecasting using deep learning techniques. IEEE Access 10, 31692–31698. https://doi.org/10.1109/ACCESS.2022.3160484.

Fafalios, S., Charonyktakis, P., Tsamardinos, I., 2020. Gradient Boosting Trees. Stefanos Fafalios, Pavlos Charonyktakis, Ioannis Tsamardinos Gnos. Data Anal. PC 1, 13.

Fan, G.-F., Guo, Y.-H., Zheng, J.-M., Hong, W.-C., 2019. Application of the weighted K-nearest neighbor algorithm for short-term load forecasting. Energies 12, 916. https://doi.org/10.3390/en12050916.

Fan, P., Deng, R., Qiu, J., Zhao, Z., Wu, S., 2021. Well logging curve reconstruction based on kernel ridge regression. Arab. J. Geosci. 14, 1559. https://doi.org/10.1007/s12517-021-07792-y.

Feng, Y., Hao, W., Li, H., Cui, N., Gong, D., Gao, L., 2020. Machine learning models to quantify and map daily global solar radiation and photovoltaic power. Renew. Sustain. Energy Rev. 118, 109393. https://doi.org/10.1016/j.rser.2019.109393.

Great Learning Editorial Team, 2024. Hyperparameter Tuning with GridSearchCV [WWW Document]. Gt. Learn. Educ. Serv. URL ⟨https://www.mygreatlearning.com/blog/gridsearchcv/⟩ (accessed 8.1.24).

Guo, Z., Zhou, K., Zhang, X., Yang, S., 2018. A deep learning model for short-term power load and probability density forecasting. Energy 160, 1186–1200. https://doi.org/10.1016/j.energy.2018.07.090.

Gutiérrez, L., Patiño, J., Duque-Grisales, E., 2021. A comparison of the performance of supervised learning algorithms for solar power prediction. Energies 14, 4424. https://doi.org/10.3390/en14154424.

Heaton, J., 2015. Artificial Intelligence for Humans, Volume 3: Neural Networks and Deep Learning. Heat. Res. Inc, Chesterfield, ABD 30, 55.

Hossain, M.S., Mahmood, H., 2020. Short-term photovoltaic power forecasting using an LSTM neural network and synthetic weather forecast. IEEE Access 8, 172524–172533. https://doi.org/10.1109/ACCESS.2020.3024901.

James, G., Witten, D., Hastie, T., Tibshirani, R., Taylor, J., 2023. Linear regression. In: An Introduction to Statistical Learning: With Applications in Python. Springer, pp. 69–134.

Jebli, I., Belouadha, F.-Z., Kabbaj, M.I., Tilioua, A., 2021. Prediction of solar energy guided by pearson correlation using machine learning. Energy 224, 120109. https://doi.org/10.1016/j.energy.2021.120109.

Jumin, E., Basaruddin, F.B., Yusoff, Y.B.M., Latif, S.D., Ahmed, A.N., 2021. Solar radiation prediction using boosted decision tree regression model: a case study in Malaysia. Environ. Sci. Pollut. Res. 28, 26571–26583. https://doi.org/10.1007/s11356-021-12435-6.

Kabilan, R., Chandran, V., Yogapriya, J., Karthick, A., Gandhi, P.P., Mohanavel, V., Rahim, R., Manoharan, S., 2021. Short-term power prediction of building integrated photovoltaic (BIPV) system based on machine learning algorithms. Int. J. Photoenergy 2021, 1–11. https://doi.org/10.1155/2021/5582418.

Khan, P.W., Byun, Y.-C., Lee, S.-J., Kang, D.-H., Kang, J.-Y., Park, H.-S., 2020. Machine learning-based approach to predict energy consumption of renewable and

nonrenewable power sources. Energies 13, 4870. https://doi.org/10.3390/en13184870.
Khan, W., Walker, S., Zeiler, W., 2022. Improved solar photovoltaic energy generation forecast using deep learning-based ensemble stacking approach. Energy 240, 122812. https://doi.org/10.1016/j.energy.2021.122812.
Kim, D., Kwon, D., Park, L., Kim, J., Cho, S., 2021. Multiscale LSTM-based deep learning for very-short-term photovoltaic power generation forecasting in smart city energy management. IEEE Syst. J. 15, 346–354. https://doi.org/10.1109/JSYST.2020.3007184.
Konstantinou, M., Peratikou, S., Charalambides, A.G., 2021. Solar photovoltaic forecasting of power output using LSTM networks. Atmosphere (Basel) 12, 124. https://doi.org/10.3390/atmos12010124.
Kramer, O., 2013. K-Nearest Neighbors. Dimensionality Reduction with Unsupervised Nearest Neighbors. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 13–23. https://doi.org/10.1007/978-3-642-38652-7_2.
Kumar, P.M., Saravanakumar, R., Karthick, A., Mohanavel, V., 2022. Artificial neural network-based output power prediction of grid-connected semitransparent photovoltaic system. Environ. Sci. Pollut. Res. 29, 10173–10182. https://doi.org/10.1007/s11356-021-16398-6.
Kumari, P., Toshniwal, D., 2021. Deep learning models for solar irradiance forecasting: a comprehensive review. J. Clean. Prod. 318, 128566. https://doi.org/10.1016/j.jclepro.2021.128566.
Lai, J.-P., Chang, Y.-M., Chen, C.-H., Pai, P.-F., 2020. A survey of machine learning models in renewable energy predictions. Appl. Sci. 10, 5975. https://doi.org/10.3390/app10175975.
Li, G., Xie, S., Wang, B., Xin, J., Li, Y., Du, S., 2020. Photovoltaic power forecasting with a hybrid deep learning approach. IEEE Access 8, 175871–175880. https://doi.org/10.1109/ACCESS.2020.3025860.
Li, P., Zhou, K., Lu, X., Yang, S., 2020. A hybrid deep learning model for short-term PV power forecasting. Appl. Energy 259, 114216. https://doi.org/10.1016/j.apenergy.2019.114216.
Lima, M.A.F.B., Carvalho, P.C.M., Fernández-Ramírez, L.M., Braga, A.P.S., 2020. Improving solar forecasting using deep learning and portfolio theory integration. Energy 195, 117016. https://doi.org/10.1016/j.energy.2020.117016.
Lin, G., Lin, A., Gu, D., 2022. Using support vector regression and K-nearest neighbors for short-term traffic flow prediction based on maximal information coefficient. Inf. Sci. (Ny) 608, 517–531. https://doi.org/10.1016/j.ins.2022.06.090.
Lin, G.-Q., Li, L.-L., Tseng, M.-L., Liu, H.-M., Yuan, D.-D., Tan, R.R., 2020. An improved moth-flame optimization algorithm for support vector machine prediction of photovoltaic power generation. J. Clean. Prod. 253, 119966. https://doi.org/10.1016/j.jclepro.2020.119966.
Liu, C.-H., Gu, J.-C., Yang, M.-T., 2021. A simplified LSTM neural networks for one day-ahead solar power forecasting. IEEE Access 9, 17174–17195. https://doi.org/10.1109/ACCESS.2021.3053638.
Liu, D., Fan, Z., Fu, Q., Li, M., Faiz, M.A., Ali, S., Li, T., Zhang, L., Khan, M.I., 2020. Random forest regression evaluation model of regional flood disaster resilience based on the whale optimization algorithm. J. Clean. Prod. 250, 119468. https://doi.org/10.1016/j.jclepro.2019.119468.
Liu, Z.-F., Li, L.-L., Tseng, M.-L., Lim, M.K., 2020. Prediction short-term photovoltaic power using improved chicken swarm optimizer - extreme learning machine model. J. Clean. Prod. 248, 119272. https://doi.org/10.1016/j.jclepro.2019.119272.
Luo, X., Zhang, D., Zhu, X., 2021. Deep learning based forecasting of photovoltaic power generation by incorporating domain knowledge. Energy 225, 120240. https://doi.org/10.1016/j.energy.2021.120240.
Mahmud, K., Azam, S., Karim, A., Zobaed, S., Shanmugam, B., Mathur, D., 2021. Machine learning based PV power generation forecasting in alice springs. IEEE Access 9, 46117–46128. https://doi.org/10.1109/ACCESS.2021.3066494.
Markovics, D., Mayer, M.J., 2022. Comparison of machine learning methods for photovoltaic power forecasting based on numerical weather prediction. Renew. Sustain. Energy Rev. 161, 112364. https://doi.org/10.1016/j.rser.2022.112364.
Maulud, D., Abdulazeez, A.M., 2020. A review on linear regression comprehensive in machine learning. J. Appl. Sci. Technol. Trends 1, 140–147. https://doi.org/10.38094/jastt1457.
Mayer, M.J., 2022. Benefits of physical and machine learning hybridization for photovoltaic power forecasting. Renew. Sustain. Energy Rev. 168, 112772. https://doi.org/10.1016/j.rser.2022.112772.
Mellit, A., Massi Pavan, A., Ogliari, E., Leva, S., Lughi, V., 2020. Advanced methods for photovoltaic output power forecasting: a review. Appl. Sci. 10, 487. https://doi.org/10.3390/app10020487.
Mellit, A., Pavan, A.M., Lughi, V., 2021. Deep learning neural networks for short-term photovoltaic power forecasting. Renew. Energy 172, 276–288. https://doi.org/10.1016/j.renene.2021.02.166.
Mishra, M., Byomakesha Dash, P., Nayak, J., Naik, B., Kumar Swain, S., 2020. Deep learning and wavelet transform integrated approach for short-term solar PV power prediction. Measurement 166, 108250. https://doi.org/10.1016/j.measurement.2020.108250.
Modaresi, F., Araghinejad, S., Ebrahimi, K., 2018. A comparative assessment of artificial neural network, generalized regression neural network, least-square support vector regression, and K-nearest neighbor regression for monthly streamflow forecasting in linear and nonlinear conditions. Water Resour. Manag. 32, 243–258. https://doi.org/10.1007/s11269-017-1807-2.
Mohammed, S.A., Al-Haddad, L.A., Alawee, W.H., Dhahad, H.A., Jaber, A.A., Al-Haddad, S.A., 2024. Forecasting the productivity of a solar distiller enhanced with an inclined absorber plate using stochastic gradient descent in artificial neural networks. Multiscale Multidiscip. Model. Exp. Des. 7, 1819–1829. https://doi.org/10.1007/s41939-023-00309-y.
Munawar, U., Wang, Z., 2020. A framework of using machine learning approaches for short-term solar power forecasting. J. Electr. Eng. Technol. 15, 561–569. https://doi.org/10.1007/s42835-020-00346-4.
Narvaez, G., Giraldo, L.F., Bressan, M., Pantoja, A., 2021. Machine learning for site-adaptation and solar radiation forecasting. Renew. Energy 167, 333–342. https://doi.org/10.1016/j.renene.2020.11.089.
Olson, R.S., (n.d). Using TPOT [WWW Document]. epistasislab - github. URL ⟨https://epistasislab.github.io/tpot/using/⟩ (accessed 8.1.24).
Padhma, 2024. A Comprehensive Introduction to Evaluating Regression Models [WWW Document]. Anal. Vidhya. URL ⟨https://www.analyticsvidhya.com/blog/2021/10/evaluation-metric-for-regression-models/⟩ (accessed 8.2.24).
Pan, M., Li, C., Gao, R., Huang, Y., You, H., Gu, T., Qin, F., 2020. Photovoltaic power forecasting based on a support vector machine with improved ant colony optimization. J. Clean. Prod. 277, 123948. https://doi.org/10.1016/j.jclepro.2020.123948.
Park, M.K., Lee, J.M., Kang, W.H., Choi, J.M., Lee, K.H., 2021. Predictive model for PV power generation using RNN (LSTM). J. Mech. Sci. Technol. 35, 795–803. https://doi.org/10.1007/s12206-021-0140-0.
Park, S., Kim, Y., Ferrier, N.J., Collis, S.M., Sankaran, R., Beckman, P.H., 2021. Prediction of solar irradiance and photovoltaic solar energy product based on cloud coverage estimation using machine learning methods. Atmosphere (Basel) 12, 395. https://doi.org/10.3390/atmos12030395.
Peng, L., Zheng, S., Zhong, Q., Chai, X., Lin, J., 2023. A novel bagged tree ensemble regression method with multiple correlation coefficients to predict the train body vibrations using rail inspection data. Mech. Syst. Signal Process 182, 109543. https://doi.org/10.1016/j.ymssp.2022.109543.
Pham, B.T., Nguyen, M.D., Bui, K.-T.T., Prakash, I., Chapi, K., Bui, D.T., 2019. A novel artificial intelligence approach based on multi-layer perceptron neural network and biogeography-based optimization for predicting coefficient of consolidation of soil. CATENA 173, 302–311. https://doi.org/10.1016/j.catena.2018.10.004.
Qu, J., Qian, Z., Pei, Y., 2021. Day-ahead hourly photovoltaic power forecasting using attention-based CNN-LSTM neural network embedded with multiple relevant and target variables prediction pattern. Energy 232, 120996. https://doi.org/10.1016/j.energy.2021.120996.
Qu, Y., Xu, J., Sun, Y., Liu, D., 2021. A temporal distributed hybrid deep learning model for day-ahead distributed PV power forecasting. Appl. Energy 304, 117704. https://doi.org/10.1016/j.apenergy.2021.117704.
Rahul, 2021. Tune Hyperparameters with GridSearchCV [WWW Document]. Anal. Vidhya. URL ⟨https://www.analyticsvidhya.com/blog/2021/06/tune-hyperparameters-with-gridsearchcv/⟩ (accessed 8.1.24).
Rajagukguk, R.A., Ramadhan, R.A.A., Lee, H.-J., 2020. A review on deep learning models for forecasting time series data of solar irradiance and photovoltaic power. Energies 13, 6623. https://doi.org/10.3390/en13246623.
Ramadhan, R.A.A., Heatubun, Y.R.J., Tan, S.F., Lee, H.-J., 2021. Comparison of physical and machine learning models for estimating solar irradiance and photovoltaic power. Renew. Energy 178, 1006–1019. https://doi.org/10.1016/j.renene.2021.06.079.
Ramezanian, R., Peymanfar, A., Ebrahimi, S.B., 2019. An integrated framework of genetic network programming and multi-layer perceptron neural network for prediction of daily stock return: an application in Tehran stock exchange market. Appl. Soft Comput. 82, 105551. https://doi.org/10.1016/j.asoc.2019.105551.
Rana, M., Rahman, A., 2020. Multiple steps ahead solar photovoltaic power forecasting based on univariate machine learning models and data re-sampling. Sustain. Energy Grids Netw. 21, 100286. https://doi.org/10.1016/j.segan.2019.100286.
Rasifaghihi, N., 2023. From Theory to Practice: Implementing Support Vector Regression for Predictions in Python [WWW Document]. Medium. URL ⟨https://medium.com/⟩@niousha.rf/support-vector-regressor-theory-and-coding-exercise-in-python-ca6a7dfda927 (accessed 7.19.24).
Sadhukhan, B., Chakraborty, S., Mukherjee, S., 2023. Predicting the magnitude of an impending earthquake using deep learning techniques. Earth Sci. Inform. 16, 803–823. https://doi.org/10.1007/s12145-022-00916-2.
Shabani, S., Samadianfard, S., Sattari, M.T., Mosavi, A., Shamshirband, S., Kmet, T., Várkonyi-Kóczy, A.R., 2020. Modeling pan evaporation using Gaussian process regression K-nearest neighbors random forest and support vector machines; comparative analysis. Atmosphere (Basel) 11, 66. https://doi.org/10.3390/atmos11010066.
Sharadga, H., Hajimirza, S., Balog, R.S., 2020. Time series forecasting of solar power generation for large-scale photovoltaic plants. Renew. Energy 150, 797–807. https://doi.org/10.1016/j.renene.2019.12.131.
Sharafati, A., Haji Seyed Asadollah, S.B., Al-Ansari, N., 2021. Application of bagging ensemble model for predicting compressive strength of hollow concrete masonry prism. Ain Shams Eng. J. 12, 3521–3530. https://doi.org/10.1016/j.asej.2021.03.028.
Song, Y., Liang, J., Lu, J., Zhao, X., 2017. An efficient instance selection algorithm for k nearest neighbor regression. Neurocomputing 251, 26–34. https://doi.org/10.1016/j.neucom.2017.04.018.
Storrs, E., (n.d). Explained: Singular Value Decomposition (SVD) [WWW Document]. Erik Storrs. URL ⟨https://storrs.io/svd/⟩ (accessed 7.29.24).
Susdesign, (n.d). SunPosition [WWW Document]. URL ⟨https://www.susdesign.com/sunposition/index.php⟩ (accessed 6.1.24).
Theocharides, S., Makrides, G., Livera, A., Theristis, M., Kaimakis, P., Georghiou, G.E., 2020. Day-ahead photovoltaic power production forecasting methodology based on machine learning and statistical post-processing. Appl. Energy 268, 115023. https://doi.org/10.1016/j.apenergy.2020.115023.
Tien Bui, D., Nhu, V.-H., Hoang, N.-D., 2018. Prediction of soil compression coefficient for urban housing project using novel integration machine learning approach of

swarm intelligence and multi-layer perceptron neural network. Adv. Eng. Inform. 38, 593–604. https://doi.org/10.1016/j.aei.2018.09.005.
Wang, F., Xuan, Z., Zhen, Z., Li, K., Wang, T., Shi, M., 2020. A day-ahead PV power forecasting method based on LSTM-RNN model and time correlation modification under partial daily pattern prediction framework. Energy Convers. Manag. 212, 112766. https://doi.org/10.1016/j.enconman.2020.112766.
Wang, H., Liu, Y., Zhou, B., Li, C., Cao, G., Voropai, N., Barakhtenko, E., 2020. Taxonomy research of artificial intelligence for deterministic solar power forecasting. Energy Convers. Manag. 214, 112909. https://doi.org/10.1016/j.enconman.2020.112909.
Wang, J., Zhou, Y., Li, Z., 2022. Hour-ahead photovoltaic generation forecasting method based on machine learning and multi objective optimization algorithm. Appl. Energy 312, 118725. https://doi.org/10.1016/j.apenergy.2022.118725.
Wang, S., Ji, B., Zhao, J., Liu, W., Xu, T., 2018. Predicting ship fuel consumption based on LASSO regression. Transp. Res. Part D. Transp. Environ. 65, 817–824. https://doi.org/10.1016/j.trd.2017.09.014.
Wu, J., Hu, W., Xiong, H., Huan, J., Braverman, V., Zhu, Z., 2020. On the noisy gradient descent that generalizes as SGD. 37th Int. Conf. Mach. Learn. ICML 2020 PartF16814 10298–10307.
Xu, N., Wang, Z., Dai, Y., Li, Q., Zhu, W., Wang, R., Finkelman, R.B., 2023. Prediction of higher heating value of coal based on gradient boosting regression tree model. Int. J. Coal Geol. 274, 104293. https://doi.org/10.1016/j.coal.2023.104293.
Xu, S.B., Huang, S.Y., Yuan, Z.G., Deng, X.H., Jiang, K., 2020. Prediction of the dst index with bagging ensemble-learning algorithm. Astrophys. J. Suppl. Ser. 248, 14. https://doi.org/10.3847/1538-4365/ab880e.
Zang, H., Cheng, L., Ding, T., Cheung, K.W., Wei, Z., Sun, G., 2020. Day-ahead photovoltaic power forecasting approach based on deep convolutional neural networks and meta learning. Int. J. Electr. Power Energy Syst. 118, 105790. https://doi.org/10.1016/j.ijepes.2019.105790.
Zazoum, B., 2022. Solar photovoltaic power prediction using different machine learning methods. Energy Rep. 8, 19–25. https://doi.org/10.1016/j.egyr.2021.11.183.
Zhang, D., Lin, J., Peng, Q., Wang, D., Yang, T., Sorooshian, S., Liu, X., Zhuang, J., 2018. Modeling and simulating of reservoir operation using the artificial neural network, support vector regression, deep learning algorithm. J. Hydrol. 565, 720–736. https://doi.org/10.1016/j.jhydrol.2018.08.050.
Zhang, J., Tan, Z., Wei, Y., 2020. An adaptive hybrid model for day-ahead photovoltaic output power prediction. J. Clean. Prod. 244, 118858. https://doi.org/10.1016/j.jclepro.2019.118858.
Zhang, S., Wang, J., Liu, H., Tong, J., Sun, Z., 2021. Prediction of energy photovoltaic power generation based on artificial intelligence algorithm. Neural Comput. Appl. 33, 821–835. https://doi.org/10.1007/s00521-020-05249-z.
Zhao, D., Wu, Q., 2018. An approach to predict the height of fractured water-conducting zone of coal roof strata using random forest regression. Sci. Rep. 8, 10986. https://doi.org/10.1038/s41598-018-29418-2.
Zhen, H., Niu, D., Wang, K., Shi, Y., Ji, Z., Xu, X., 2021. Photovoltaic power forecasting based on GA improved Bi-LSTM in microgrid without meteorological information. Energy 231, 120908. https://doi.org/10.1016/j.energy.2021.120908.
Zhen, Z., Liu, J., Zhang, Z., Wang, F., Chai, H., Yu, Y., Lu, X., Wang, T., Lin, Y., 2020. Deep learning based surface irradiance mapping model for solar PV power forecasting using sky image. IEEE Trans. Ind. Appl. https://doi.org/10.1109/TIA.2020.2984617, 1–1.
Zhou, Y., Zhou, N., Gong, L., Jiang, M., 2020. Prediction of photovoltaic power output based on similar day analysis, genetic algorithm and extreme learning machine. Energy 204, 117894. https://doi.org/10.1016/j.energy.2020.117894.