www.editada.org

International Journal of Combinatorial Optimization Problems and Informatics, 17(5), 2026, 80-105. ISSN: 2007-1558.

https://doi.org/10.61467/2007.1558.2026.v17i5.1441

Comparative Benchmark of Eleven Regression Models for Software Effort Estimation on a COCOMO-Like Dataset

Jaime Aguilar-Ortiz (jao@upp.edu.mx),
Universidad Politécnica de Pachuca,
Víctor M. Zamudio-García (vzamudio@upmh.edu.mx),
Universidad Politécnica Metropolitana de Hidalgo,
Marcos Yamir Gómez-Ramos (myamir@upmh.edu.mx),
Universidad Politécnica Metropolitana de Hidalgo,
Carlos R. Domínguez-Mayorga (carlos.dominguez@upp.edu.mx),
Universidad Politécnica de Pachuca

Abstract. This study presents a controlled comparison of eleven regression methods for software effort estimation on a 62-project COCOMO-like dataset with 20 numeric predictors. A leakage-safe design reserved 49 projects for model development and 13 for final hold-out assessment. The candidate set comprised a three-hidden-layer deep neural network, CatBoost, XGBoost, a stacked ensemble, random forest, support vector regression with a radial-basis function kernel, Gaussian process regression, LightGBM, an optimized M5P model tree, gradient boosting, and AdaBoost. Performance was evaluated through repeated 5×2 cross-validation on the development partition and an independent hold-out test using MAE, RMSE, R^2, MMRE, MdMRE, and PRED(25). By deriving the primary ordering from repeated validation and treating the hold-out split as corroborative evidence, the analysis emphasizes stability as well as point accuracy. Gaussian process regression achieved the most accurate and stable repeated-validation profile (mean rank = 1.0000, RMSE = 0.2152, R^2 = 0.8998, and PRED(25) = 0.9289) and retained the leading position on the hold-out set (MAE = 0.1444, RMSE = 0.1835, R^2 = 0.9084, MMRE = 0.0850, MdMRE = 0.0634, and PRED(25) = 0.9231). The repeated-validation order was Gaussian process regression, optimized M5P, stacked ensemble, gradient boosting, XGBoost, random forest, CatBoost, AdaBoost, support vector regression, deep neural network, and LightGBM. Permutation importance identified KDSI, ACAP, PCAP, RELY, and AAF as the most influential predictors. Within this 62-project sample, kernel and piecewise-linear/tree formulations yielded lower and more stable errors than the deeper neural baseline.

Keywords: software effort estimation, software cost estimation, Gaussian process regression, model trees, stacked regression, benchmarking, MMRE, PRED(25), COCOMO-like dataset, regression ranking.

Article Info
Received April 24, 2026
Accepted Aug 23, 2026

1 Introduction

Software effort estimation remains one of the most delicate decision problems in software engineering because budget negotiation, staffing decisions, delivery commitments, and risk management depend directly on the quality of early forecasts. An underestimate can translate into budget pressure and delayed delivery, whereas an overly cautious figure may discourage viable work or tie up resources unnecessarily. Consequently, software effort studies have drawn on parametric models, expert assessments, analogies, and empirical learning methods to relate project characteristics to observed effort.

Classical COCOMO formulations established a structured vocabulary for software size, product complexity, platform constraints, personnel capability, and schedule pressure (Boehm, 1981; Boehm et al., 2000). Once effort estimation is cast as a nonlinear supervised-learning problem, however, prediction need not depend on a single parametric relationship. Contemporary regressors can represent interactions among cost drivers, capture local nonlinearities, and manage the bias-variance trade-off in ways that may better accommodate heterogeneous project portfolios.

Earlier work on this dataset emphasized a three-hidden-layer neural estimator. The present study reframes the problem as a controlled comparative benchmark: rather than determining whether one deep architecture can produce acceptable estimates, it examines which of eleven regressors offers the most credible balance of predictive accuracy and stability under a common preprocessing, validation, and evaluation protocol.

The study makes four contributions. First, it reconstructs a leakage-controlled benchmark that applies a common partitioning and preprocessing logic to all eleven estimators. Second, it states the mathematical basis of each model and relates that formulation to the evaluation criteria. Third, it interprets the full set of numerical summaries and diagnostic figures rather than relying on a single score. Fourth, it derives the primary ordering from repeated cross-validation and uses the untouched hold-out partition as an independent corroborative check.

2 Experimental procedures

2.1 Dataset characteristics, preprocessing, and validation design

The analysis was conducted on a prepared table containing 62 project records, 20 numeric predictors, and the response Effort. The predictor set comprised KDSI, AAF, RELY, DATA, CPLX, TIME, STOR, VIRT, TURN, ACAP, AEXP, PCAP, VEXP, LEXP, MODP, TOOL, SCED, MODE1, MODE2, and MODE3. Because all fields were complete and no categorical variable remained after preparation, the benchmark operated on a fully numeric COCOMO-based representation.

The analytical file contains 62 records from the classical COCOMO project database collected at TRW and reported by Boehm (1981). Although the historical archive is commonly described as comprising 63 projects, the available spreadsheet contained 62 complete cases, and preprocessing did not remove any additional observation. Because the missing project identifier was not retained, the analysis explicitly treats the file as a 62-record subset rather than as the complete archive.

In the working representation, KDSI is the logarithm of thousands of delivered source instructions and AAF is the adaptation/adjustment factor retained from the source table. RELY, DATA, and CPLX describe required reliability, database size, and product complexity; TIME, STOR, VIRT, and TURN cover execution-time constraint, storage constraint, virtual-machine volatility, and turnaround time; ACAP, AEXP, PCAP, VEXP, and LEXP represent analyst capability, application experience, programmer capability, virtual-machine experience, and language experience; MODP, TOOL, and SCED denote modern programming practices, tool support, and required schedule. MODE1–MODE3 are binary indicators for the organic, semi-detached, and embedded modes. Effort is the logarithmically transformed person-month response in the supplied data.

A single 80/20 partition was formed before model comparison, leaving 49 records for development and 13 for testing. Only the 49 development cases entered the repeated K-fold procedure (five folds, repeated twice), which produced ten validation results per estimator. Rankings were calculated from those repeated-validation summaries; the 13 held-out observations were examined only after the ordering had been fixed.

The 80/20 split was generated with scikit-learn's train_test_split (test_size = 0.20, random_state = 42), yielding 49 development cases and 13 test cases. The fixed seed makes the partition reproducible. More importantly, defining the hold-out set before repeated cross-validation prevents those 13 observations from influencing model selection or hyperparameter comparison.

All transformations were fitted within scikit-learn pipelines so that validation observations did not inform the preprocessing steps. Numeric features were median-imputed and standardized only for scale-dependent estimators: the neural network, SVR, and Gaussian process regression. Tree ensembles and M5P received the unscaled numeric branch, and a variance filter

removed constant columns. Since the prepared file contained no categorical inputs, the one-hot-encoding branch was not invoked.

Each of the eleven configured estimators completed the experiment. TensorFlow was available, although the runtime exposed no GPU, so the neural model and the remaining methods were evaluated on the CPU. Using one computational environment for the full catalog reduces the likelihood that hardware differences, rather than modeling choices, account for the observed performance gaps.

Table 1. Benchmark configuration used in the experiment.

Characteristic	Value
Projects	62
Predictors	20 numeric variables
Target variable	Effort
Missing values	0 across all variables
Hold-out partition	49 training projects / 13 test projects
Internal validation protocol	Repeated K-fold, 5 splits × 2 repeats = 10 validation folds
Evaluation metrics	MAE, RMSE, R^2, MMRE, MdMRE, and PRED(25)
Ranking criterion	Average of metric-specific ranks over the six evaluation measures
Preprocessing	Median imputation, model-dependent scaling, variance-threshold filtering, leakage-safe pipelines
Runtime context	No visible TensorFlow GPU was detected; all models were evaluated on the CPU.
Executed models	11 of 11 configured models were available

2.2 Regression metrics and ranking criterion

Let x_i denote the predictor vector for project i, y_i the observed effort, and $\hat{y}_i$ the corresponding estimate. Equation (1) represents the prediction problem, whereas Eqs. (2)–(8) define the six performance criteria. MAE, RMSE, MMRE, and MdMRE are minimized; R^2 and PRED(25) are maximized. Equation (9) assigns model m an overall score equal to the mean of its metric-specific ranks.

$$\hat{y}_i = f(\mathrm{x}_i; \theta) \tag{1}$$

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{2}$$

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{3}$$

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{4}$$

$$\mathrm{MRE}_i = \frac{|y_i - \hat{y}_i|}{\max(|y_i|, \varepsilon)} \tag{5}$$

$$\mathrm{MMRE} = \frac{1}{n}\sum_{i=1}^{n}\mathrm{MRE}_i \tag{6}$$

$$\mathrm{MdMRE} = \mathrm{median}(\mathrm{MRE}_1, \ldots, \mathrm{MRE}_n) \tag{7}$$

$$\mathrm{PRED}(25) = \frac{1}{n}\sum_{i=1}^{n}\mathbf{1}\,(\mathrm{MRE}_i \le 0.25) \tag{8}$$

$$\mathrm{MR}(m) = \frac{1}{6}\sum_{j=1}^{6}\mathrm{rank}_j\,(m) \tag{9}$$

RMSE warrants particular attention because it is used in both the comparative plots and the permutation-importance analysis of the leading estimator. In Eq. (5), a small ε is added to the denominator so that relative error remains defined when the observed response approaches zero. Equation (9) then combines the six metric-specific positions into a single mean rank, preventing model selection from being determined by one favorable metric or one split.

2.3 Mathematical specification of the evaluated regressors

Table 2 summarizes the estimator families and their principal hyperparameters. The following subsections formalize each prediction mechanism and, where relevant, relate it to the performance criteria in Eqs. (2)–(9).

Table 2. Model families and main hyperparameter settings used by the benchmark suite.

Model	Family	Key settings used in the benchmark
Deep Neural Network (3 hidden layers)	Dense network	256-128-64 hidden units; dropout 0.20/0.20/0.10; L2=1e-4; Adam lr=0.001; batch size=16; max epochs=400; early stopping; target scaling
CatBoost Regressor	Ordered boosting	500 trees; learning rate 0.03; depth 6; L2 leaf regularization 3.0; subsample 0.90
XGBoost Regressor	Regularized tree boosting	500 trees; learning rate 0.03; max depth 4; subsample 0.90; colsample_bytree 0.90; λ=1.0
Stacked Ensemble Regressor	Meta-ensemble	Base learners: RF, XGB, SVR and GBR; meta-learner: RidgeCV with logarithmic alpha grid
Random Forest Regressor	Bagged trees	800 trees; unrestricted depth; bootstrap aggregation; random state 42
Support Vector Regression (RBF)	Kernel method	C=10.0; ε=0.10; RBF kernel; gamma='scale'
Gaussian Process Regression	Bayesian kernel regression	ConstantKernel × RBF + WhiteKernel; alpha=1e-6; normalize_y=True; 2 optimizer restarts
LightGBM Regressor	Leaf-wise gradient boosting	500 trees; learning rate 0.03; num_leaves 31; subsample 0.90; colsample_bytree 0.90
Optimized Model Tree (M5P)	Piecewise linear model tree	Pruning enabled; smoothing enabled; smoothing constant 15
Gradient Boosting Regressor	Stage-wise additive trees	500 estimators; learning rate 0.03; max depth 3; subsample 0.90
AdaBoost Regressor	Adaptive boosting (AdaBoost.R2)	500 estimators; learning rate 0.03; default regression tree weak learner

2.3.1 Deep neural network with three hidden layers

The neural comparator is a fully connected regressor with hidden layers of 256, 128, and 64 units. Starting from input x, Eq. (10) gives the first nonlinear representation; Eqs. (11) and (12) apply batch normalization and dropout; and Eq. (13) maps the final hidden state to a continuous response. Training minimizes the L2-regularized squared-error objective in Eq. (14) with Adam, early stopping, and learning-rate reduction.

The three hidden transformations allow the network to represent interactions that a shallow linear model cannot capture directly. In a 62-project sample, however, that flexibility is accompanied by substantial estimation variance because a large parameter set must be inferred from few observations. The empirical comparison therefore evaluates whether the additional expressive capacity translates into generalization rather than merely closer in-sample fit.

$$\mathrm{h}^{(1)} = \mathrm{ReLU}\left(\mathrm{W}^{(1)}\mathrm{x} + \mathrm{b}^{(1)}\right) \tag{10}$$

$$\tilde{\mathrm{h}}^{(1)} = \mathrm{BN}\left(\mathrm{h}^{(1)}\right) = \gamma \odot \frac{\mathrm{h}^{(1)} - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}} + \beta \tag{11}$$

$$\mathrm{d}^{(1)} = \mathrm{m}^{(1)} \odot \tilde{\mathrm{h}}^{(1)}, \qquad m_j^{(1)} \sim \mathrm{Bernoulli}(1 - p_1) \tag{12}$$

$$\hat{y} = \mathrm{W}^{(o)}\mathrm{h}^{(3)} + b^{(o)} \tag{13}$$

$$J(\Theta) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda \sum_{\ell=1}^{L} \| \mathrm{W}^{(\ell)} \|_F^2 \tag{14}$$

2.3.2 CatBoost regressor

CatBoost belongs to the gradient-boosting family, but it differs from conventional boosting because it uses ordered statistics and ordered boosting to reduce target leakage and prediction shift (Prokhorenkova et al., 2018). The ordered statistic in Eq. (15) expresses the general idea: the transformed value of observation i is computed from observations that appear earlier in a random permutation, not from the full sample. The boosted ensemble itself remains additive, as shown in Eq. (16).

In the present dataset all predictors were numeric, so the categorical-statistics mechanism of Eq. (15) was not activated operationally during fitting. Even so, the CatBoost regressor still trained an ordered boosting ensemble of symmetric trees, and Eq. (16) remains the appropriate mathematical representation of its prediction rule.

$$\mathrm{Ctr}_i(c) = \frac{\sum_{\sigma(j)<\sigma(i)} \mathbf{1}(x_j=c)\, y_j + ap}{\sum_{\sigma(j)<\sigma(i)} \mathbf{1}(x_j=c) + a} \tag{15}$$

$$F_M(\mathrm{x}) = \sum_{m=0}^{M} \eta\ h_m(\mathrm{x}) \tag{16}$$

2.3.3 XGBoost regressor

XGBoost is also an additive tree model, but it optimizes a regularized objective with explicit complexity penalties on each tree (Chen & Guestrin, 2016). Equations (17) and (18) describe, respectively, the stagewise prediction update and the complexity penalty applied to the number of leaves T and leaf coefficients w_j. This regularization is one reason why XGBoost frequently performs well on structured tabular data with moderate sample sizes.

The fitted XGBoost model used histogram-based tree construction, 500 stages, a learning rate of 0.03, and a maximum depth of 4. These choices produce many small updates rather than a few aggressive corrections, thereby placing more emphasis on out-of-sample behavior.

$$\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(\mathbf{x}_i) \tag{17}$$

$$\Omega(f_t) = \gamma T + \frac{\lambda}{2}\sum_{j=1}^{T} w_j^2 \tag{18}$$

2.3.4 Stacked ensemble regressor

Stacking combines heterogeneous learners by feeding their out-of-fold predictions into a second-level meta-model (Wolpert, 1992). The feature vector seen by the meta-learner is given by Eq. (19), and the final combination used in this benchmark was a ridge regressor, represented by Eqs. (20) and (21). In other words, the benchmark did not simply average the base learners; it estimated regularized linear weights from their cross-validated predictions.

The base layer comprised random forest, XGBoost, SVR, and gradient boosting, thereby combining bagging, regularized tree boosting, kernel regression, and stage-wise additive learning. Their out-of-fold predictions were supplied to the ridge meta-learner. The intended gain is error diversification: stacking is useful only when the constituent models make sufficiently different mistakes.

$$z_{im} = f_m^{(-i)}(\mathrm{x}_i) \tag{19}$$

$$\hat{y}_i = \beta_0 + \sum_{m=1}^{M} \beta_m\, z_{im} \tag{20}$$

$$\hat{\beta} = \underset{\beta}{\mathrm{argmin}}\{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda \parallel \beta \parallel_2^2\} \tag{21}$$

2.3.5 Random forest regressor

Random forest is a bagging strategy over randomized regression trees (Breiman, 2001). For each member of the ensemble, a bootstrap sample and a random subset of candidate features determine the fitted tree; Eq. (22) averages their predictions. The method tends to reduce variance relative to a single tree while preserving strong nonlinear flexibility. The reported forest comprised 800 trees.

$$\hat{y}(\mathrm{x}) = \frac{1}{B}\sum_{b=1}^{B} T_b\,(\mathrm{x}) \tag{22}$$

2.3.6 Support vector regression with RBF kernel

Support vector regression fits a regularized function under the ε-insensitive loss, which ignores deviations that fall inside an ε-wide tube (Cortes & Vapnik, 1995; Smola & Schölkopf, 2004). Equation (23) gives the primal trade-off between model smoothness and penalized error, while Eq. (24) expresses the prediction as a weighted sum over support vectors. With the RBF kernel used here, the influence of a training case decreases with its squared Euclidean distance from the query point.

For a small sample, SVR is a useful reference because its regularization parameter and ε-insensitive loss constrain effective complexity. It therefore offers a markedly different comparison with the higher-capacity neural architecture.

$$\min_f \left\{\frac{1}{2} \parallel \mathrm{w} \parallel^2 + C\sum_{i=1}^{n} L_\varepsilon\left(y_i, f(\mathrm{x}_i)\right)\right\} \tag{23}$$

$$\hat{y}(\mathrm{x}) = \sum_{i=1}^{n}(\alpha_i - \alpha_i')\exp(-\gamma \parallel \mathrm{x} - \mathrm{x}_i \parallel^2) + b \tag{24}$$

2.3.7 Gaussian process regression

Gaussian process regression places a probability distribution over possible regression functions rather than choosing a fixed parametric form (Rasmussen & Williams, 2006). The prior is stated in Eq. (25); after conditioning on the training observations, Eqs. (26) and (27) provide the posterior mean and uncertainty at x*. The implementation combined a constant and RBF covariance with a white-noise component, allowing a smooth signal while acknowledging observation-level noise.

Under this model, the posterior mean is assembled from the observed responses, with weights determined by covariance similarity and the estimated noise level. For a small numeric dataset, that mechanism can adapt locally without introducing a large number of freely estimated parameters.

$$f(\mathrm{x}) \sim \mathcal{GP}\big(m(\mathrm{x}), k(\mathrm{x}, \mathrm{x}')\big) \tag{25}$$

$$\mu_{\mathrm{new}} = \mathrm{k}_{\mathrm{new}}^{\mathrm{T}}(\mathrm{K} + \sigma_n^2\mathrm{I})^{-1}\mathrm{y} \tag{26}$$

$$\sigma_{\mathrm{new}}^2 = k(\mathrm{x}_{\mathrm{new}}, \mathrm{x}_{\mathrm{new}}) - \mathrm{k}_{\mathrm{new}}^{\mathrm{T}}(\mathrm{K} + \sigma_n^2\mathrm{I})^{-1}\mathrm{k}_{\mathrm{new}} \tag{27}$$

2.3.8 LightGBM regressor

LightGBM builds gradient-boosted trees from binned feature values and grows the tree by splitting the leaf that offers the largest gain (Ke et al., 2017). Equation (28) summarizes that gain through the first- and second-order gradient totals G and H. Unlike level-wise expansion, the procedure concentrates successive splits on the branch with the greatest estimated reduction in loss.

This leaf-wise strategy is often effective when many observations support the partitioning decisions. With few cases, however, a small change in fold membership can alter which leaf appears most profitable, producing materially different trees. The repeated-validation results make that sensitivity visible in this experiment.

$$\mathrm{Gain} = \frac{1}{2}\left[\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda}\right] - \gamma \tag{28}$$

2.3.9 Optimized model tree (M5P)

M5P recursively divides the predictor space and estimates a local linear equation within each terminal node (Quinlan, 1992; Wang & Witten, 1997). Equation (29) represents the resulting piecewise-linear function. The benchmarked implementation also applies smoothing, represented in Eq. (30), so that the terminal prediction is shrunk toward its parent-node prediction according to the local sample size n and smoothing constant k.

For effort data, this structure has a practical advantage: it permits different project regions to follow different linear relationships while preserving an inspectable equation within each region.

$$\hat{y}(\mathrm{x}) = \sum_{r=1}^{R} 1\,(\mathrm{x} \in \mathcal{R}_r)\big(\beta_{r0} + \sum_{j=1}^{p}\beta_{rj}\,x_j\big) \tag{29}$$

$$\tilde{y}_t = \frac{n_t\hat{y}_t + k\hat{y}_{\mathrm{parent}}}{n_t + k} \tag{30}$$

2.3.10 Gradient boosting regressor

Gradient boosting constructs an additive predictor by fitting each new learner to the loss gradient left by the current ensemble (Friedman, 2001). Equation (31) defines the pseudo-residuals and Eq. (32) gives the stagewise update. With regression trees as base learners, each stage targets the remaining pattern of error rather than refitting the response from the beginning.

The benchmark used 500 estimators, a learning rate of 0.03, depth 3, and a subsampling rate of 0.90. The small step size and shallow trees were selected to moderate variance in the limited sample.

$$r_{im} = -\frac{\partial L(y_i, F(\mathrm{x}_i))}{\partial F(\mathrm{x}_i)}\Big|_{F=F_{m-1}} \tag{31}$$

$$F_m(\mathrm{x}) = F_{m-1}(\mathrm{x}) + \nu\,\gamma_m h_m(\mathrm{x}) \tag{32}$$

2.3.11 AdaBoost regressor

AdaBoost.R2 changes the observation weights after each round according to relative prediction error. Equations (33) and (34) define the normalized loss and weight update, and Eq. (35) combines the fitted learners through a weighted median. Cases that are difficult for one stage consequently receive greater influence in the next.

Although the method is more familiar in classification, its regression form is relevant here because it repeatedly redirects the ensemble toward projects that previous weak learners estimated poorly.

$$L_i^{(m)} = \frac{|y_i - h_m(\mathbf{x}_i)|}{\max_j |y_j - h_m(\mathbf{x}_j)|} \tag{33}$$

$$\beta_m = \frac{\bar{L}_m}{1-\bar{L}_m}, \qquad w_i^{(m+1)} = \frac{w_i^{(m)} \beta_m^{\,1-L_i^{(m)}}}{Z_m} \tag{34}$$

$$\hat{y}(\mathrm{x}) = \text{weighted median}\{h_m(\mathrm{x})\}_{m=1}^{M} \tag{35}$$

3 Results

3.1 Overall ranking from repeated cross-validation

Table 3 presents the ordering obtained from the mean-rank rule in Eq. (9). Gaussian process regression placed first on each of the six measures—MAE, RMSE, R^2, MMRE, MdMRE, and PRED(25)—and consequently received a mean rank of 1.0000. Across the repeated folds, its mean RMSE was 0.2152 and its mean R^2 was 0.8998, the strongest values in the comparison.

M5P occupied second place and maintained a similarly consistent profile across the six criteria. The stacked ensemble and conventional gradient boosting formed the next performance tier, followed by XGBoost and random forest. CatBoost, AdaBoost, and SVR occupied the lower-middle range, while the neural network and LightGBM ranked tenth and eleventh, respectively.

For this sample, greater architectural depth did not translate into stronger generalization.

This ordering is more consistent with a sample-size effect than with a simple hierarchy of model capacity. The smooth probabilistic kernel and the locally linear tree partition described the data more reliably than the larger neural parameterization. The result suggests that, within this 62-project sample, nonlinear estimators with stronger structural regularization better match the available information.

Three features of the experiment help explain the advantage of Gaussian process regression. The standardized numeric drivers define a smooth local geometry that is compatible with the RBF covariance; the white-noise term and Gaussian prior regularize the fitted function; and the 49-case training partition is small enough for covariance-based modeling but too limited for reliable estimation of the neural network's many weights. The posterior mean can therefore borrow strength from similar projects while shrinking predictions in sparsely supported regions, which is consistent with the low fold dispersion observed in Figures 3–5.

Table 3. Model ordering from repeated cross-validation.

Pos.	Model	Mean rank	MAE	RMSE	R^2	MMRE	MdMRE	PRED(25)
1	Gaussian Process Regression	1.0000	0.1743	0.2152	0.8998	0.1026	0.0740	0.9289
2	Optimized Model Tree (M5P)	2.0000	0.2164	0.2586	0.8564	0.1268	0.0921	0.8978
3	Stacked Ensemble Regressor	3.1667	0.2565	0.3084	0.8043	0.1614	0.1110	0.8322
4	Gradient Boosting Regressor	4.3333	0.2821	0.3319	0.7647	0.1676	0.1254	0.8233
5	XGBoost Regressor	5.1667	0.3102	0.3652	0.7288	0.1769	0.1395	0.8456
6	Random Forest Regressor	6.1667	0.3087	0.3669	0.7277	0.1868	0.1395	0.7833
7	CatBoost Regressor	7.1667	0.3282	0.4284	0.6534	0.1888	0.1197	0.7744
8	AdaBoost Regressor	7.5000	0.3222	0.3844	0.6975	0.1986	0.1414	0.7822
9	Support Vector Regression (RBF)	8.5000	0.3437	0.4455	0.5849	0.2050	0.1296	0.7233
10	Deep Neural Network (3 hidden layers)	10.0000	0.5838	0.7310	-0.0446	0.3524	0.2359	0.5067
11	LightGBM Regressor	11.0000	0.6489	0.7746	-0.0995	0.3780	0.3048	0.4578

Figure 1 condenses the mean ranks into a visual ordering, where the leading four methods separate from the remainder and the distance from Gaussian process regression to the final two models is substantial. Figure 2 reaches the same conclusion metric by metric: the Gaussian-process row retains the favorable position throughout, rather than relying on one unusually strong statistic.

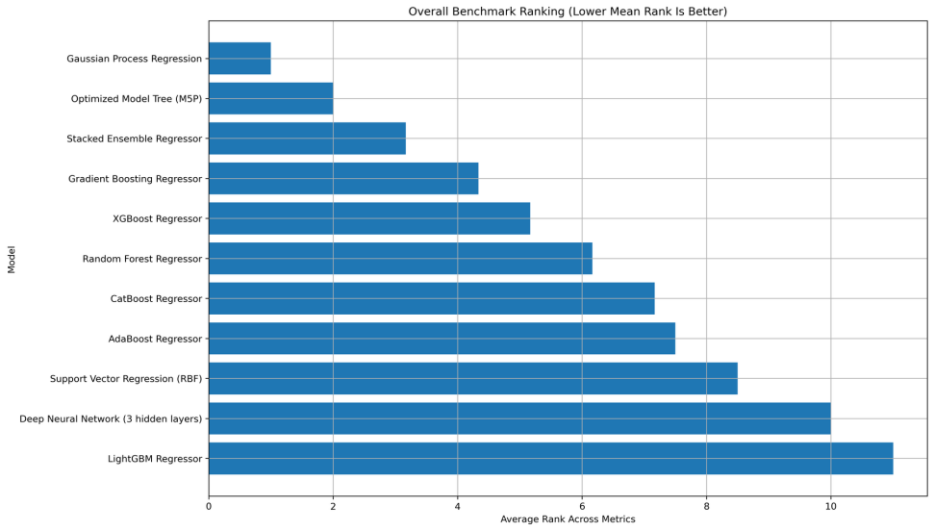

Figure 1. Mean-rank ordering across the six evaluation criteria.

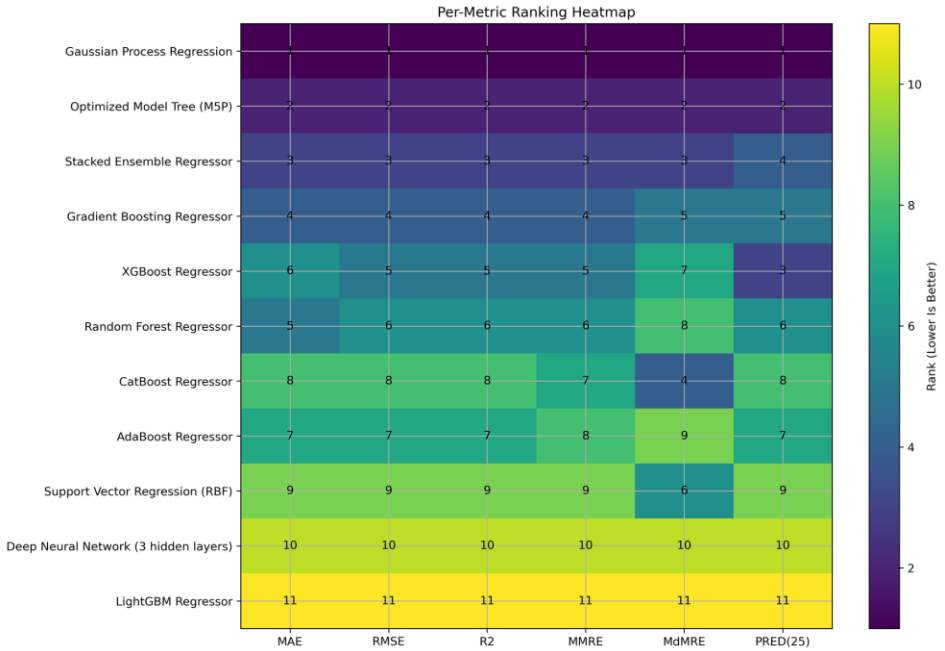

Figure 2. Metric-by-metric ranks obtained from repeated cross-validation.

Figures 3–5 add the fold-to-fold dispersion hidden by Table 3's averages. Gaussian process regression combines the lowest central MAE and RMSE with compact distributions. SVR, the neural network, and LightGBM vary more widely; for the neural model, high RMSE and R^2 values extending below zero indicate inconsistent generalization across folds.

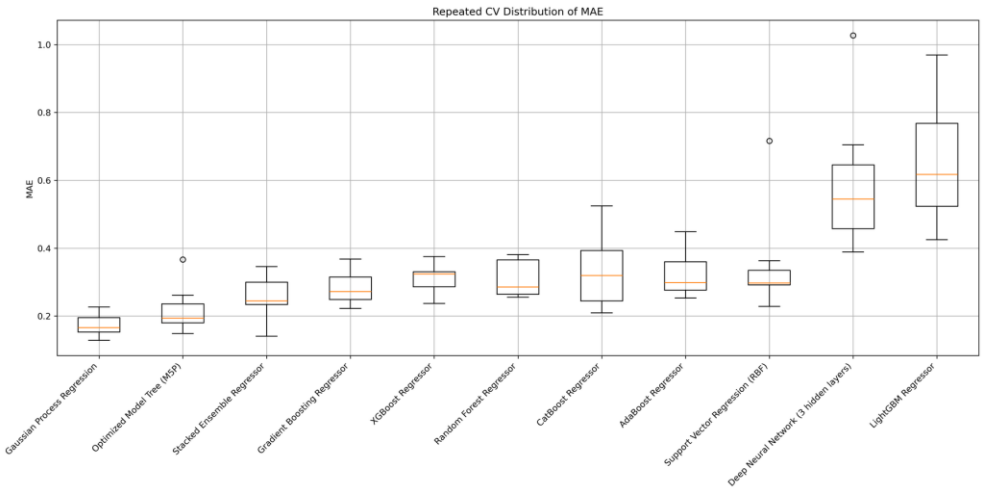

Figure 3. Repeated-cross-validation distribution of MAE for the eleven regressors.

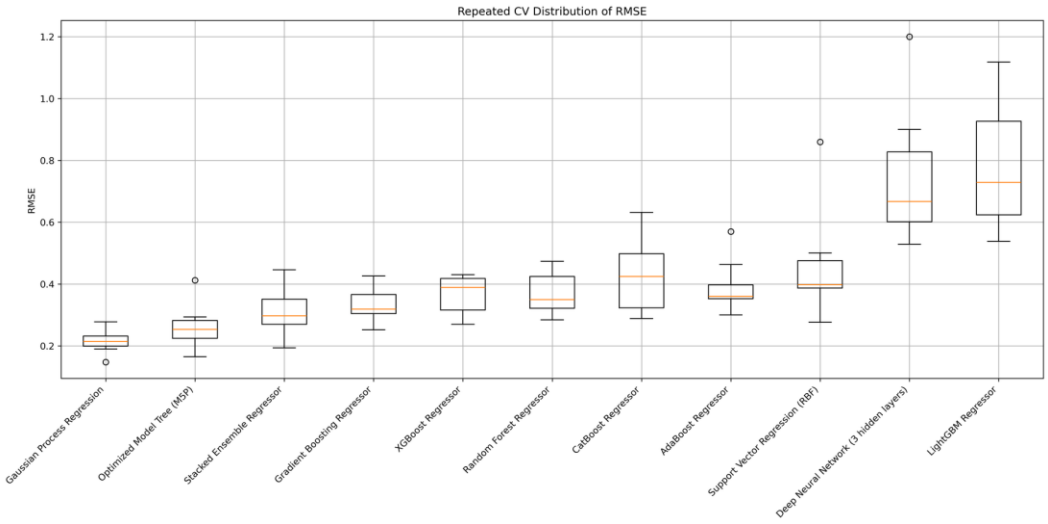

Figure 4. Repeated-cross-validation distribution of RMSE for the eleven regressors.

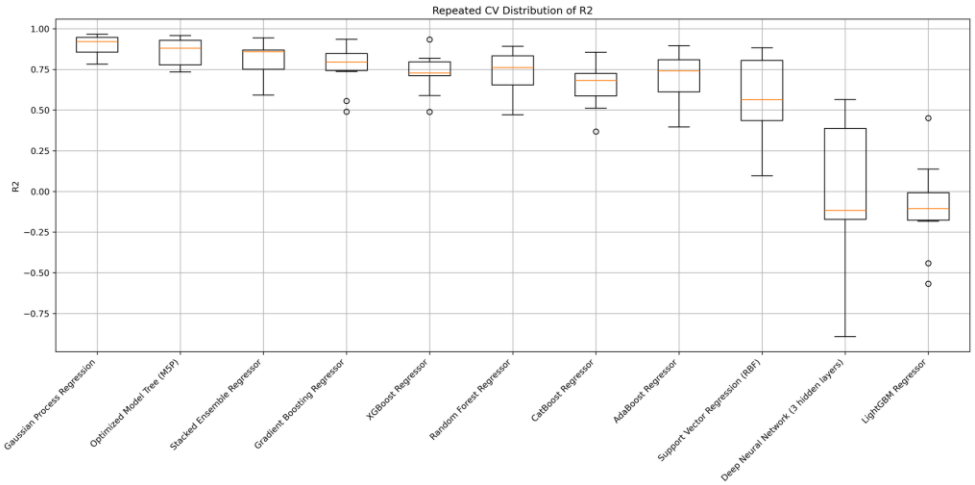

Figure 5. Repeated-cross-validation distribution of R^2 for the eleven regressors.

3.2 Metric-specific performance across evaluation criteria

Figures 6–11 present the best-to-worst repeated-cross-validation ordering for each metric, while Figures 12–17 provide the corresponding hold-out results. Examining the metrics separately reveals whether the aggregate order in Table 3 reflects broad consistency or is driven by an isolated measure.

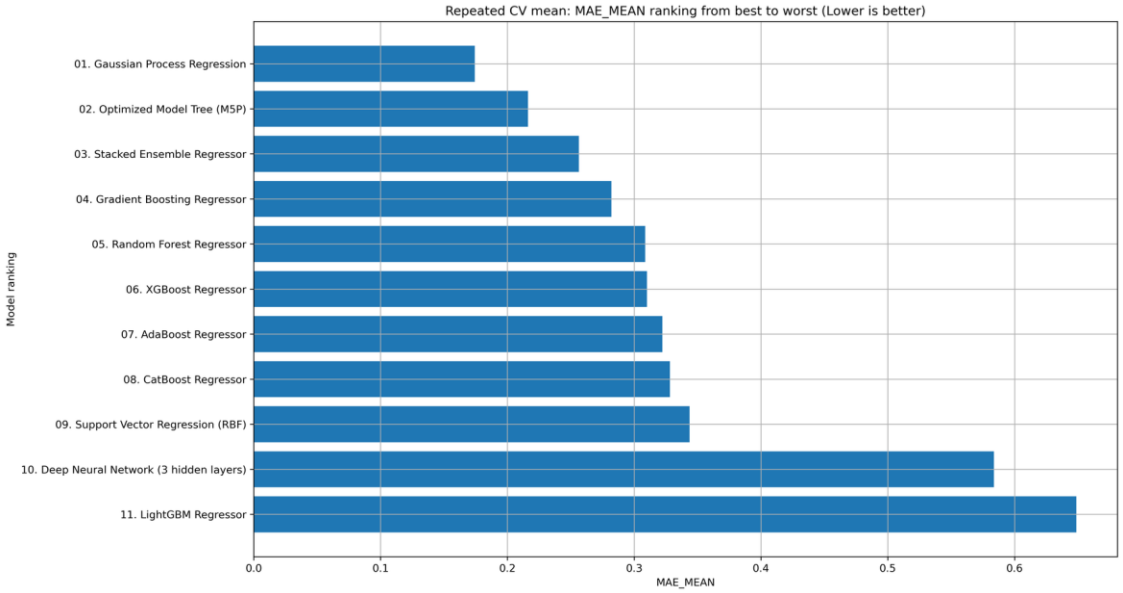

Figure 6. Repeated-cross-validation MAE ranking from best to worst.

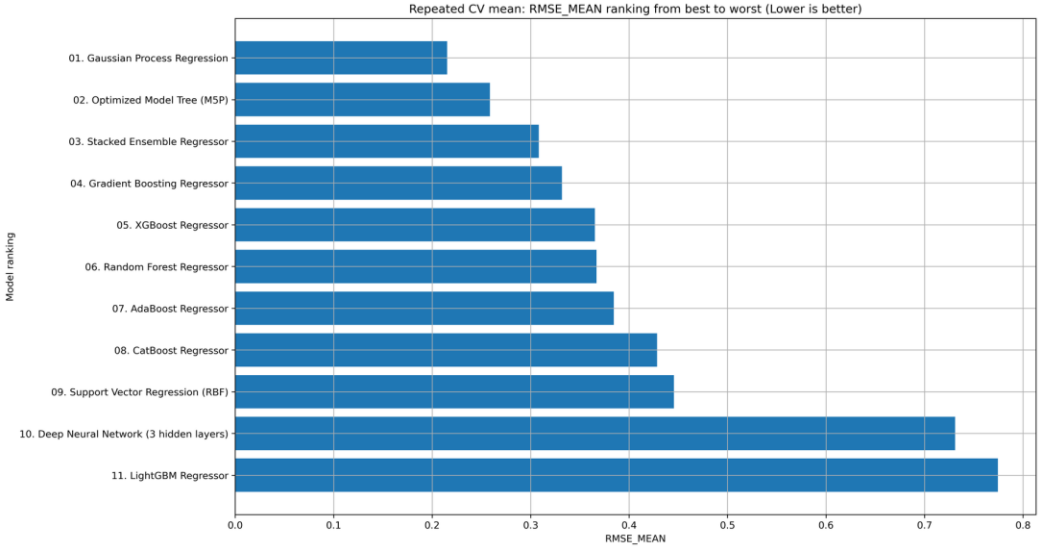

Figure 7. Repeated-cross-validation RMSE ranking from best to worst.

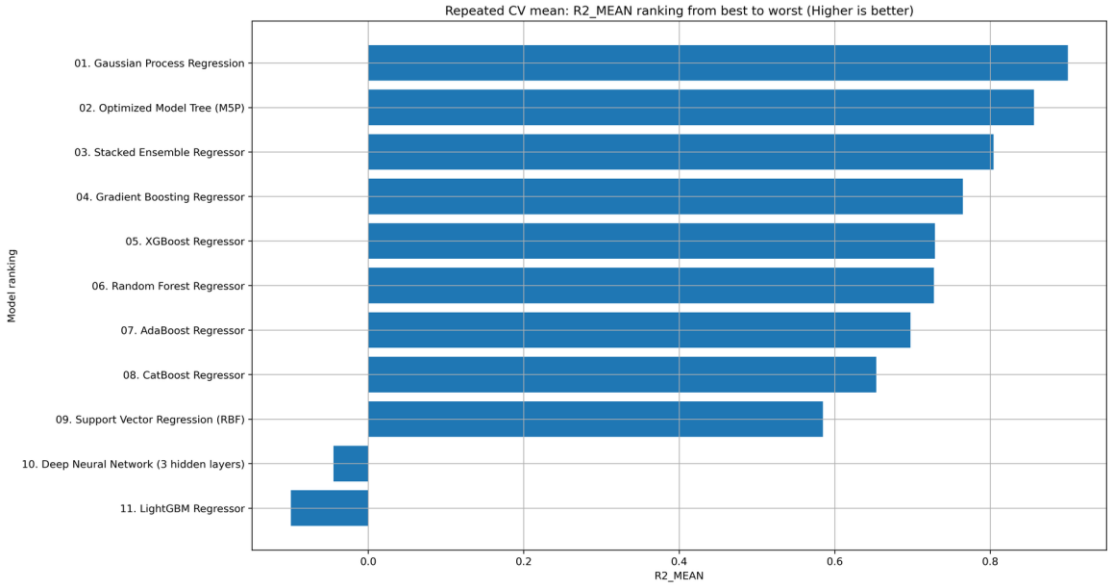

Figure 8. Repeated-cross-validation R² ranking from best to worst.

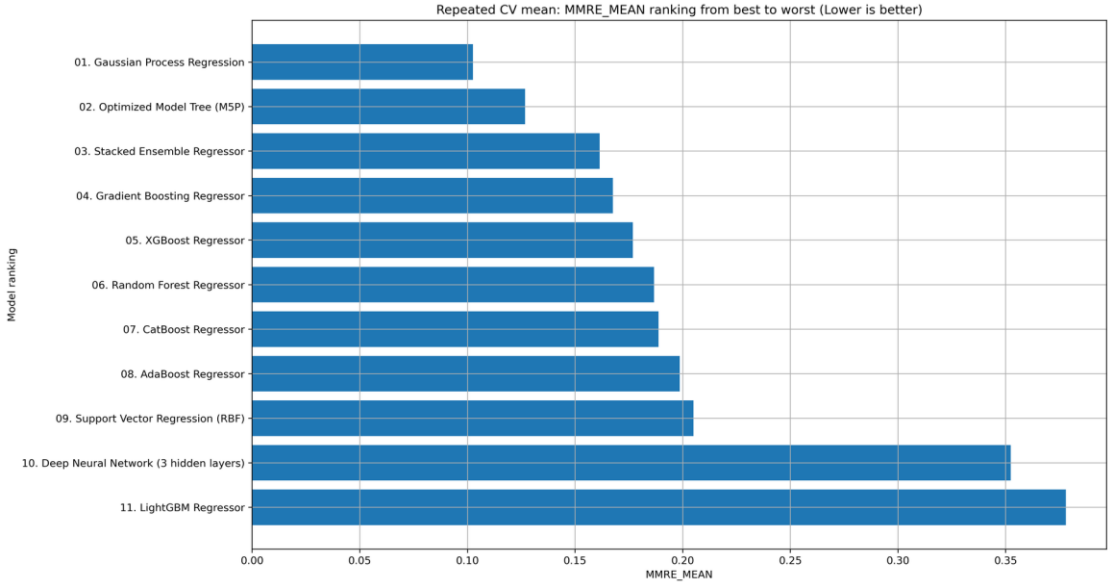

Figure 9. Repeated-cross-validation MMRE ranking from best to worst.

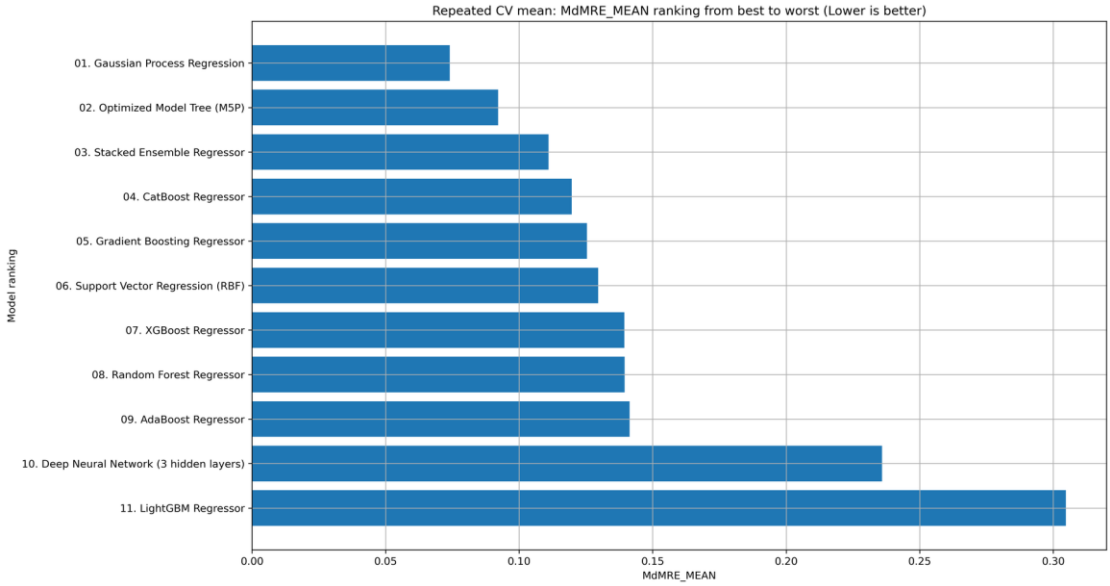

Figure 10. Repeated-cross-validation MdMRE ranking from best to worst.

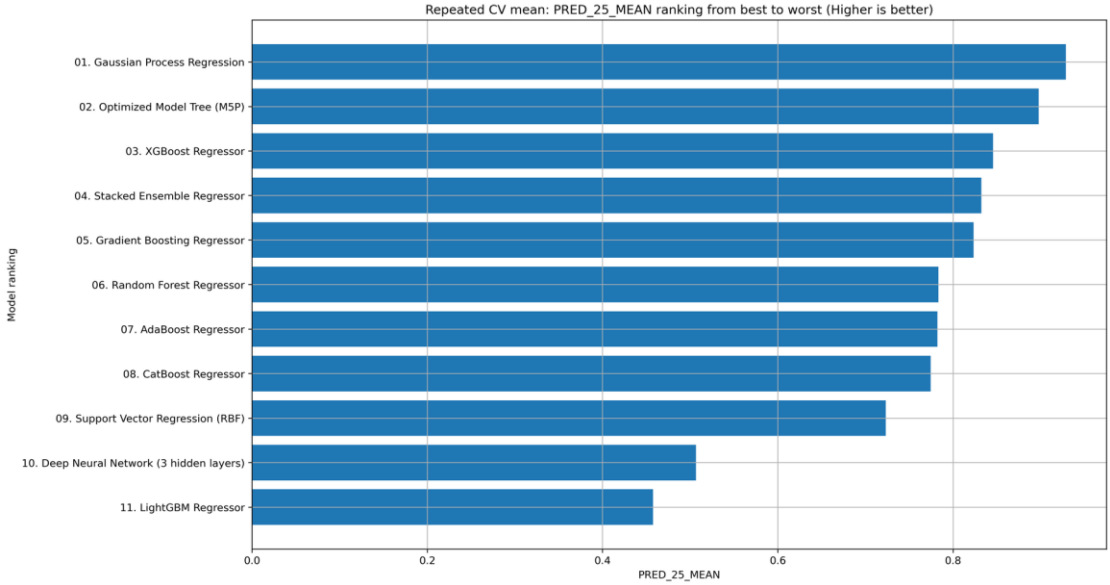

Figure 11. Repeated-cross-validation PRED(25) ranking from best to worst.

Figures 6–11 show that the repeated-cross-validation ordering is not an artifact of one favorable statistic. Across the six metric charts, Gaussian process regression retains first place and M5P remains second. Stacking and gradient boosting follow with only limited changes in position. XGBoost gains ground on PRED(25), but not enough to enter the leading pair. The neural network and LightGBM stay near the bottom for every measure, showing that their weak mean ranks are repeated across the full metric set.

Table 4. Hold-out performance ordered by the aggregate hold-out rank.

Model	MAE	RMSE	R^2	MMRE	MdMRE	PRED(25)
Gaussian Process Regression	0.1444	0.1835	0.9084	0.0850	0.0634	0.9231
LightGBM Regressor	0.2333	0.3359	0.6932	0.1445	0.0695	0.9231
Gradient Boosting Regressor	0.2639	0.3310	0.7021	0.1456	0.0800	0.8462
Stacked Ensemble Regressor	0.2469	0.3341	0.6966	0.1399	0.0835	0.8462
CatBoost Regressor	0.2798	0.3493	0.6683	0.1533	0.1116	0.9231
XGBoost Regressor	0.2857	0.3502	0.6665	0.1589	0.0849	0.8462
Optimized Model Tree (M5P)	0.2734	0.3645	0.6389	0.1629	0.0798	0.8462
Random Forest Regressor	0.2976	0.3506	0.6658	0.1570	0.1052	0.8462
Support Vector Regression (RBF)	0.3279	0.3962	0.5732	0.1870	0.1131	0.6923
AdaBoost Regressor	0.3398	0.4090	0.5451	0.1826	0.1212	0.7692
Deep Neural Network (3 hidden layers)	0.4966	0.6016	0.0158	0.2653	0.1999	0.5385

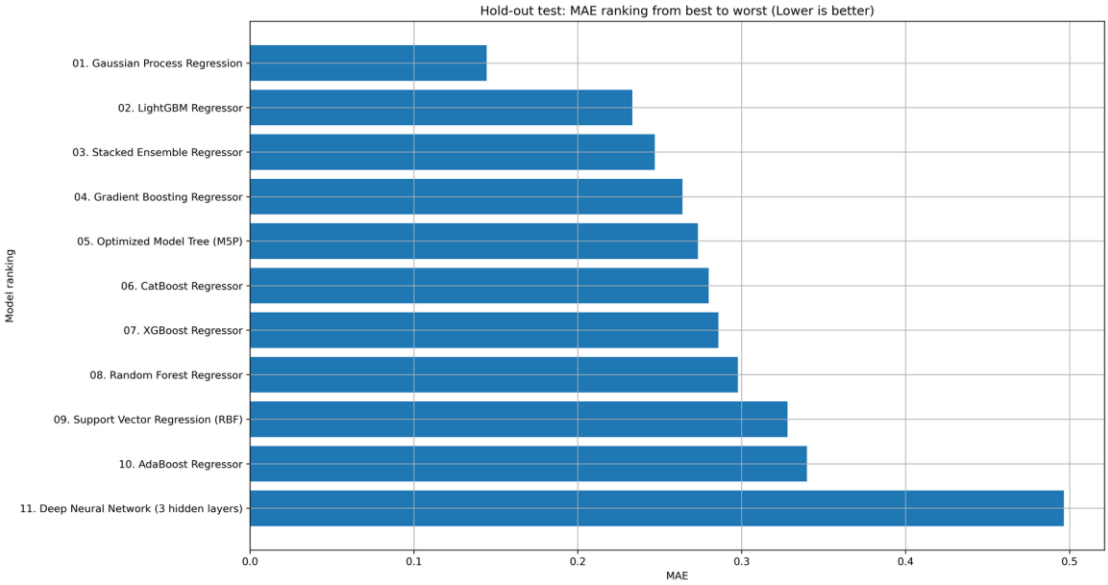

Figure 12. Hold-out test MAE ranking from best to worst.

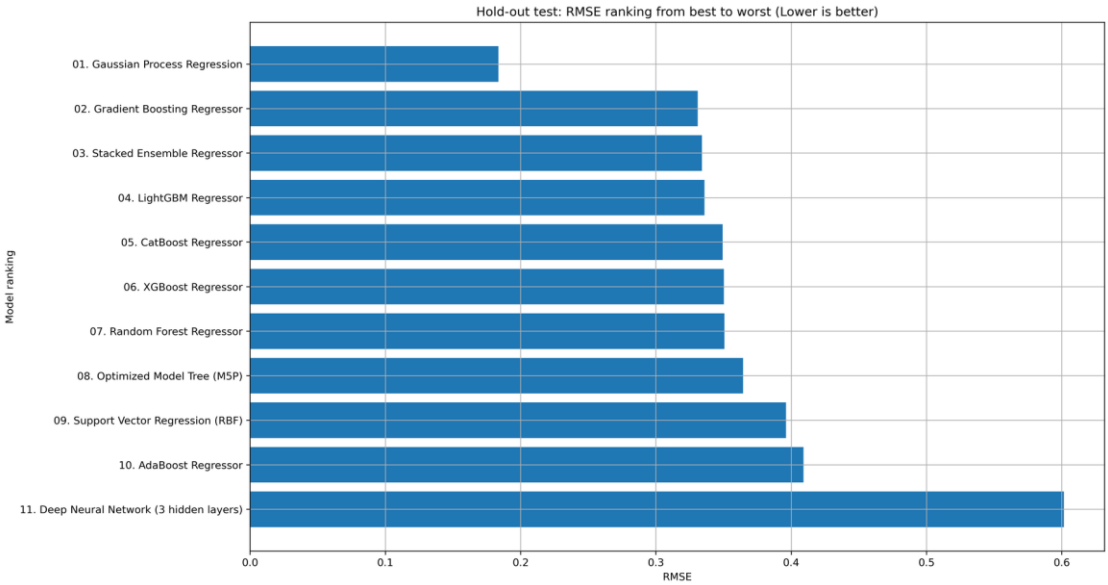

Figure 13. Hold-out test RMSE ranking from best to worst.

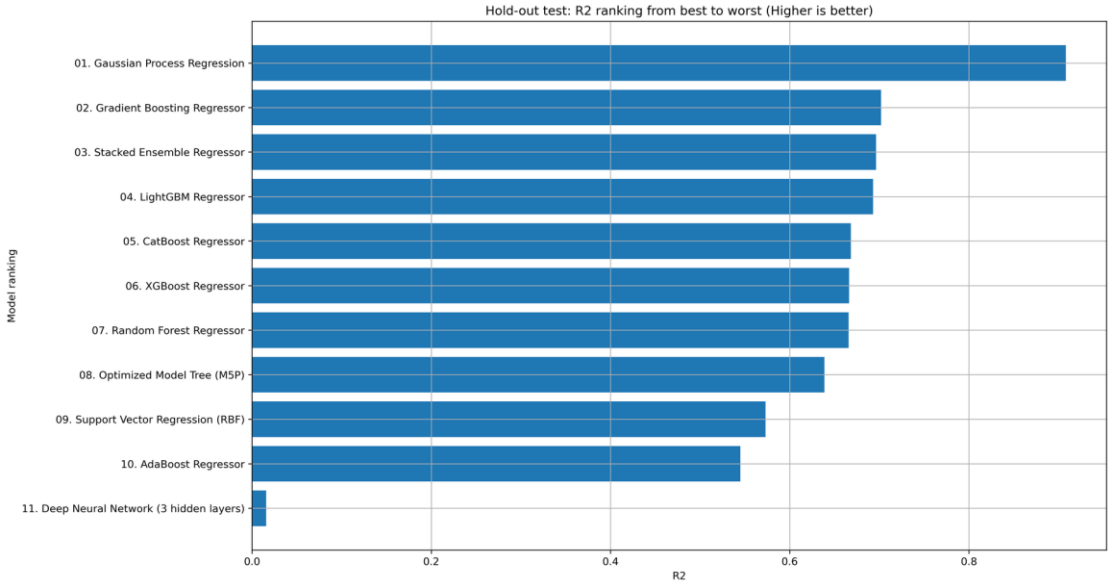

Figure 14. Hold-out test R^2 ranking from best to worst.

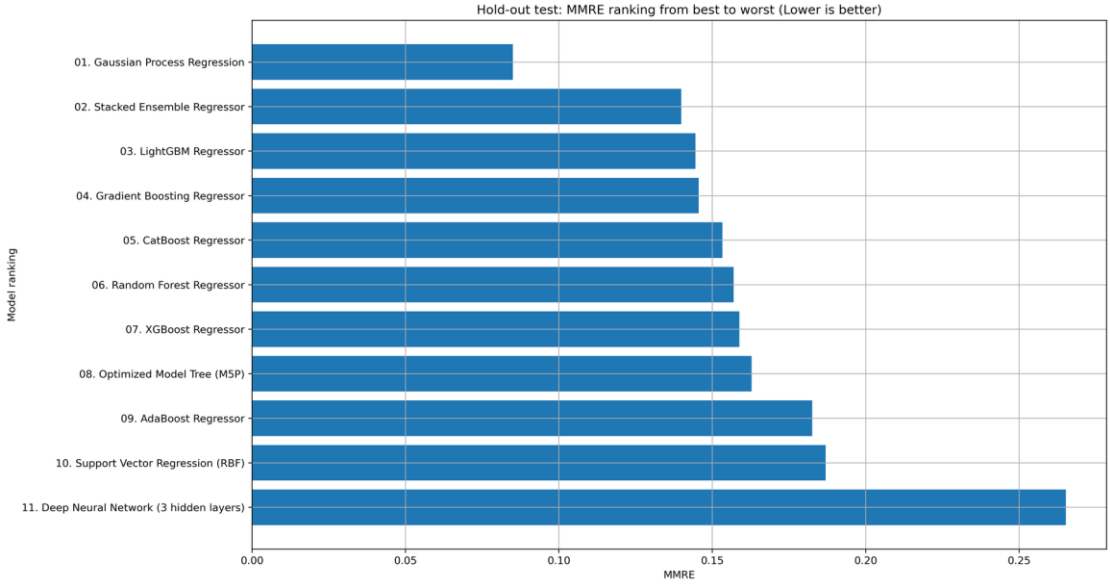

Figure 15. Hold-out test MMRE ranking from best to worst.

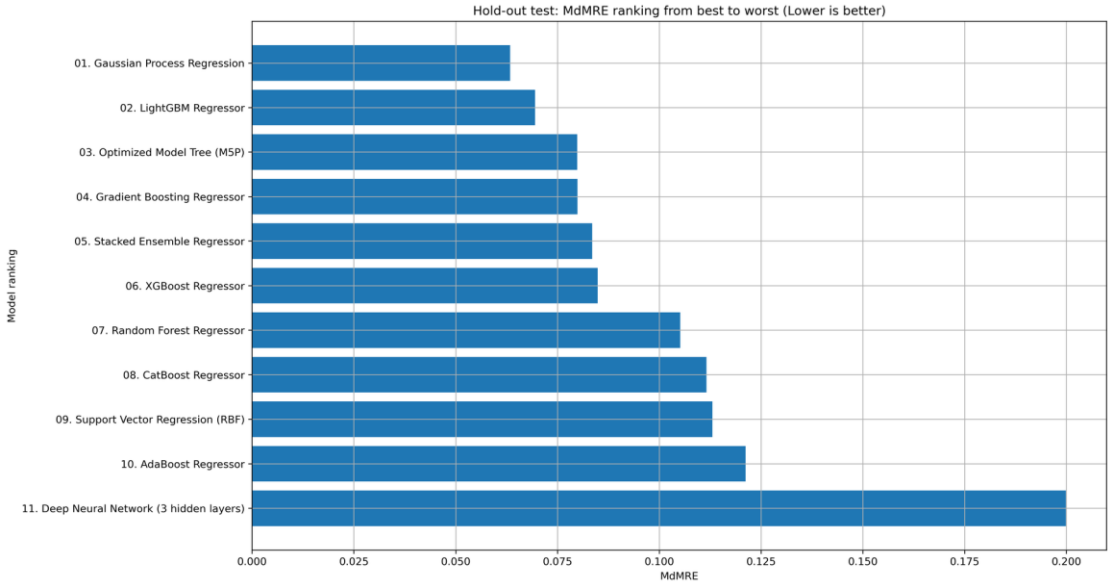

Figure 16. Hold-out test MdMRE ranking from best to worst.

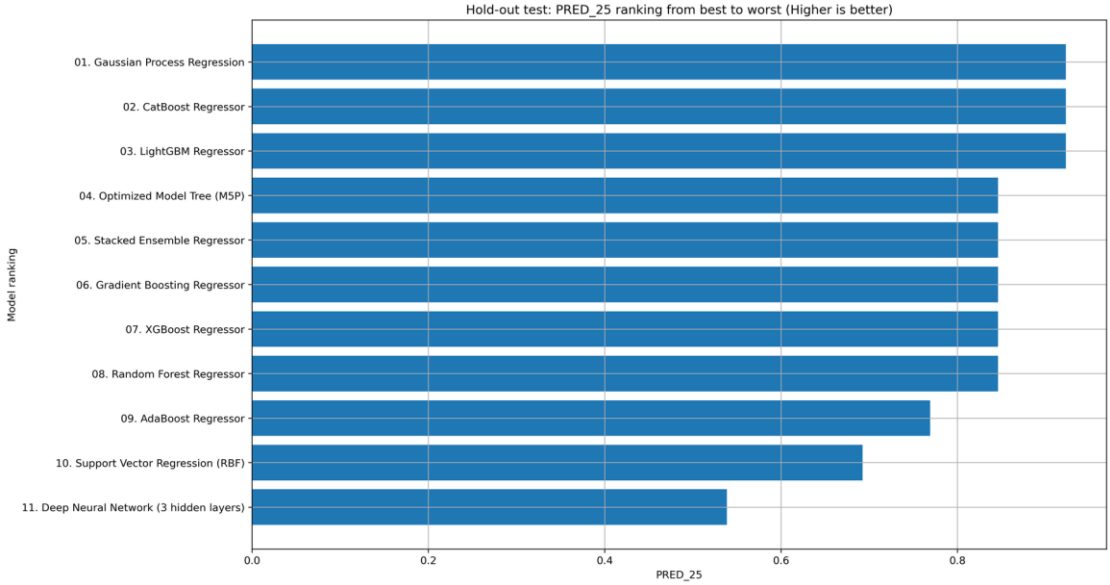

Figure 17. Hold-out test PRED(25) ranking from best to worst.

Table 4 again places Gaussian process regression first on the hold-out set, with MAE = 0.1444, RMSE = 0.1835, R^2 = 0.9084, MMRE = 0.0850, MdMRE = 0.0634, and PRED(25) = 0.9231. Because these 13 observations did not participate in model selection, the result independently corroborates the repeated-cross-validation winner. Given the small size of the test set, however, the hold-out evidence is confirmatory rather than sufficient on its own.

Figures 12–17 reveal, however, that the hold-out ordering is not identical to the repeated-cross-validation ordering. LightGBM rises dramatically on the hold-out split and becomes one of the strongest single-split performers, while M5P drops from second place in repeated cross-validation to the lower part of the upper tier on the hold-out ranking. The disagreement is informative rather than inconsistent, because a test set of 13 projects is highly sensitive to which observations happen to be included. For that reason, the article treats repeated cross-validation as the primary source of model ordering and the hold-out set as corroborative evidence, not as the sole judge of global superiority.

The LightGBM reversal is consistent with the variance of leaf-wise growth under very small samples. With 31 candidate leaves and a minimum of 20 observations per child, minor changes in fold composition can lead to different split sequences; the particular 13-project hold-out set evidently favored one such partition. Its nine-rank improvement is therefore interpreted as split sensitivity, not as a contradiction of the repeated-validation result.

3.3 Agreement and divergence between cross-validation and hold-out results

Table 5 and Figures 18–23 place each repeated-cross-validation result beside its hold-out counterpart. The comparison distinguishes models that are merely accurate on one partition from those that also behave consistently. Large differences between the two evaluations are examined as evidence of sampling variance, an idiosyncratic split, or possible overfitting.

Table 5. Change in model position from repeated cross-validation to the hold-out ranking.

Model	CV rank	Hold-out rank	Δ rank (CV - hold-out)
Gaussian Process Regression	1	1	0
Optimized Model Tree (M5P)	2	7	-5
Stacked Ensemble Regressor	3	4	-1
Gradient Boosting Regressor	4	3	1
XGBoost Regressor	5	6	-1
Random Forest Regressor	6	8	-2
CatBoost Regressor	7	5	2
AdaBoost Regressor	8	10	-2
Support Vector Regression (RBF)	9	9	0
Deep Neural Network (3 hidden layers)	10	11	-1
LightGBM Regressor	11	2	9

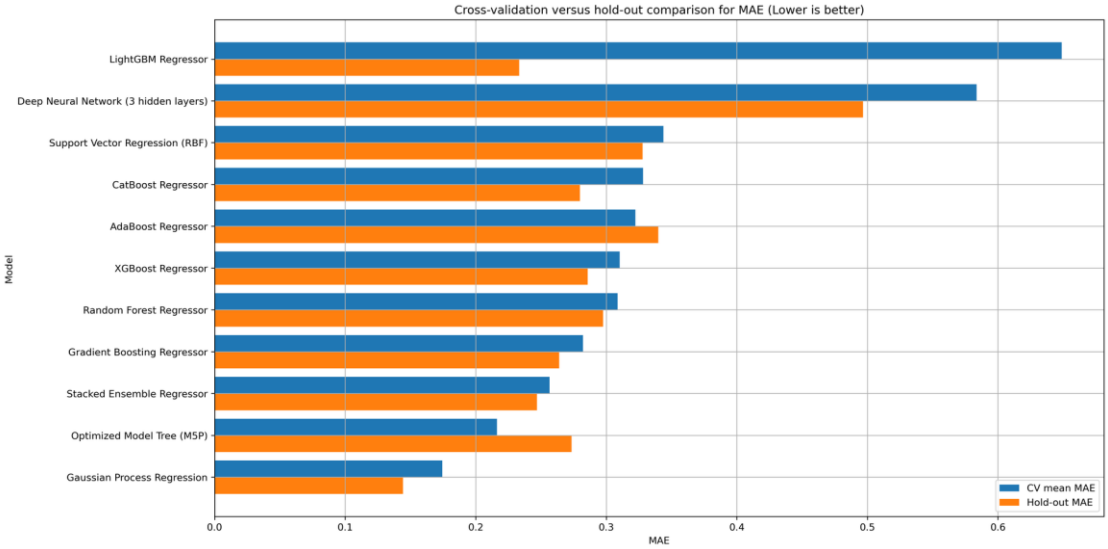

Figure 18. Cross-validation versus hold-out comparison for MAE.

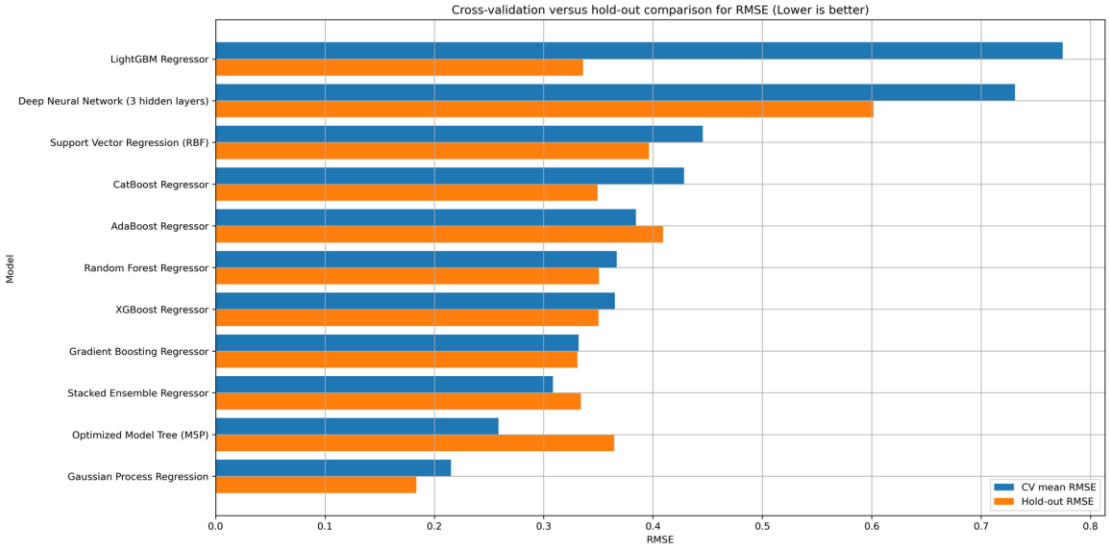

Figure 19. Cross-validation versus hold-out comparison for RMSE.

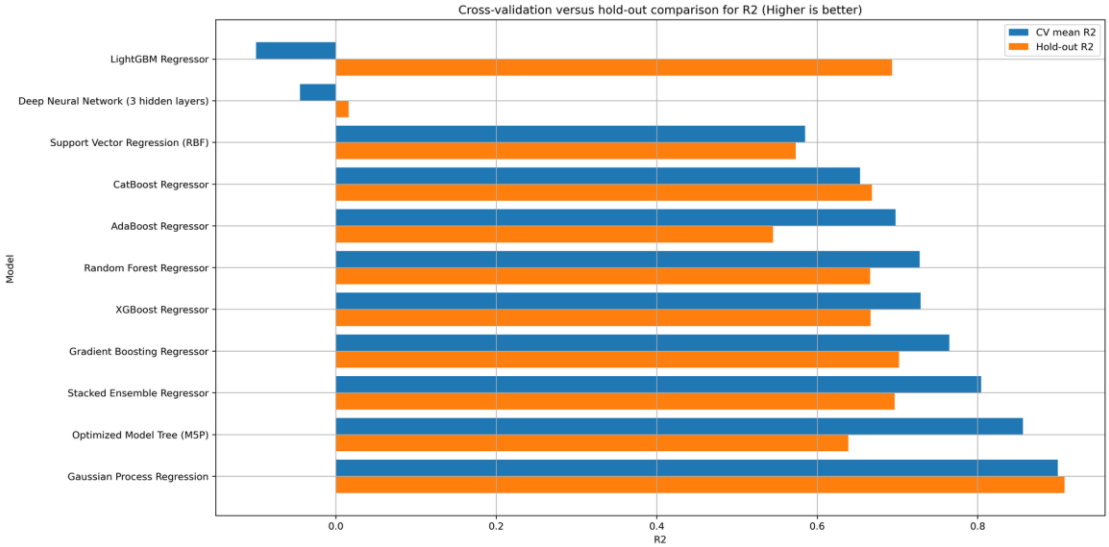

Figure 20. Cross-validation versus hold-out comparison for R^2.

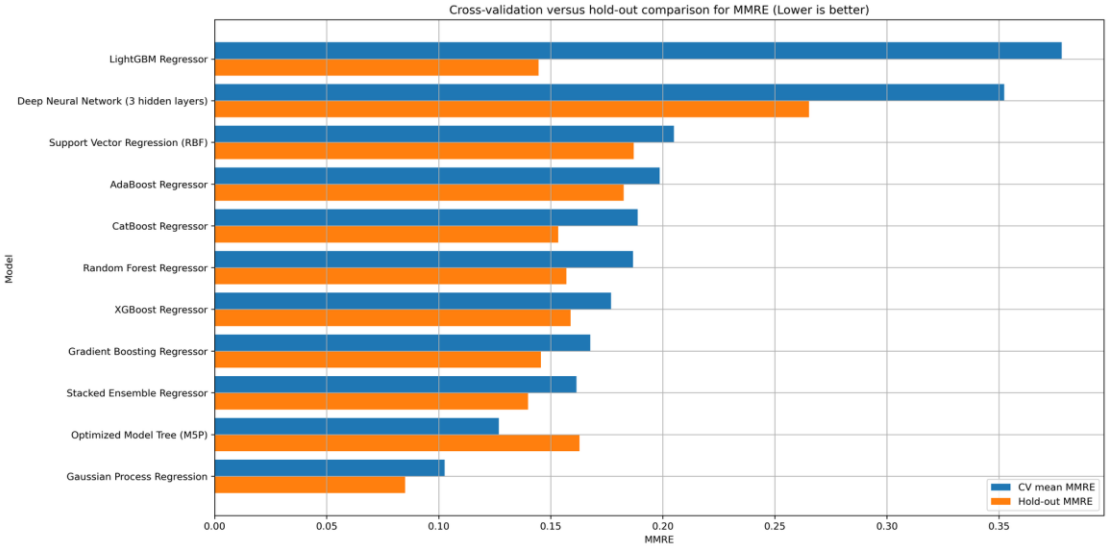

Figure 21. Cross-validation versus hold-out comparison for MMRE.

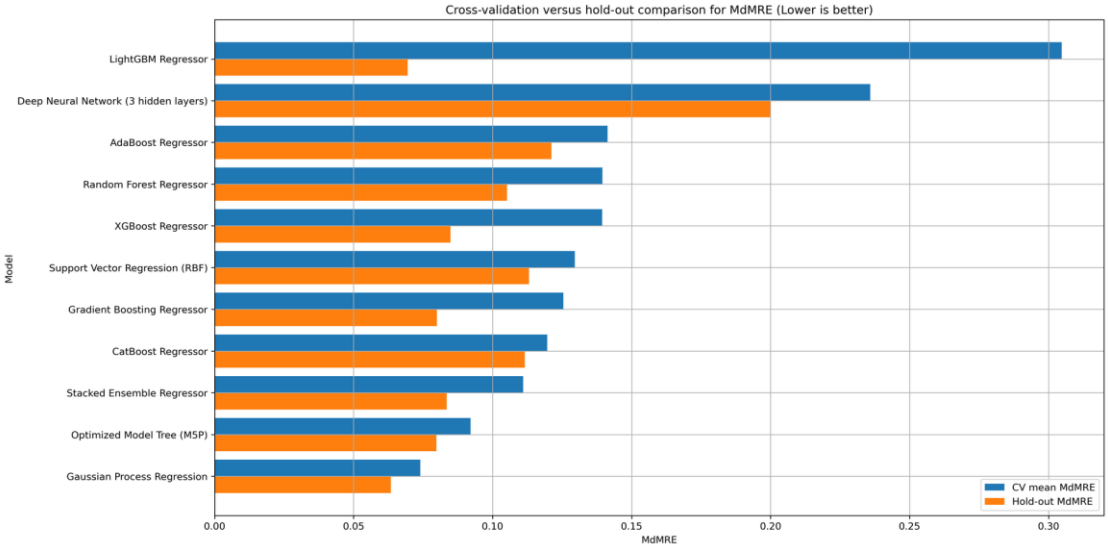

Figure 22. Cross-validation versus hold-out comparison for MdMRE.

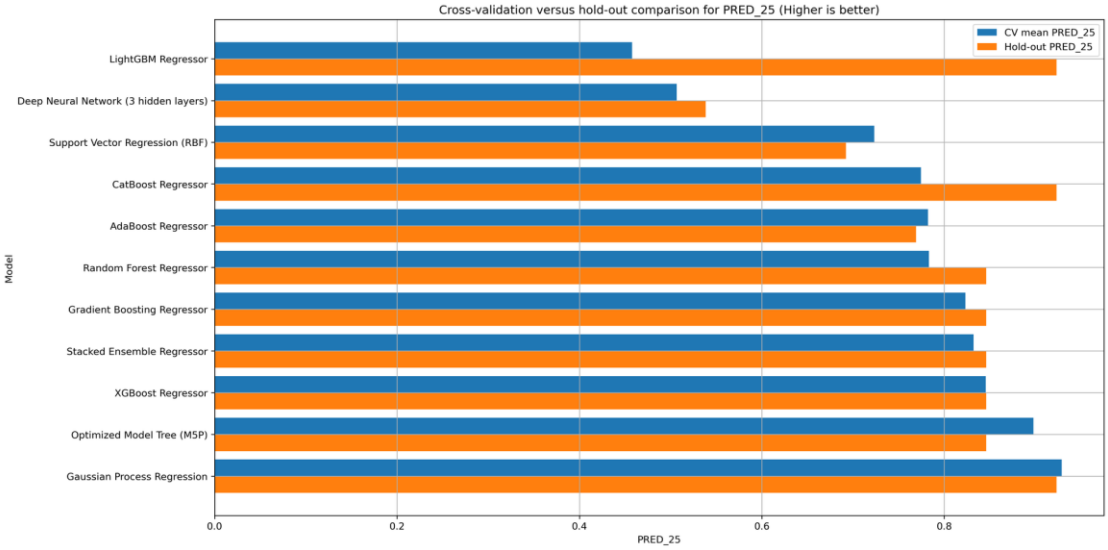

Figure 23. Cross-validation versus hold-out comparison for PRED(25).

The most stable model is again Gaussian process regression, which remains first in both repeated cross-validation and the hold-out test. The stacked ensemble and gradient boosting regressor also exhibit relatively coherent behavior, staying in the upper tier under both evaluation regimes. In contrast, LightGBM presents the most dramatic discrepancy, moving from the last position in repeated cross-validation to the second position on the hold-out split. Such a movement indicates high sensitivity to sample partition and warns against selecting LightGBM solely from one test split.

Figures 18–23 make the same point numerically. The bars for Gaussian process regression remain favorable in all pairwise comparisons between repeated cross-validation and hold-out evaluation. The bars for the deep neural network remain poor in every metric, which indicates that its underperformance is not accidental. M5P, XGBoost, CatBoost, and random forest remain competitive, although their positions change more than that of the Gaussian process. Taken together, the results favor the estimator that couples strong average accuracy with limited movement between evaluation regimes.

3.4 Diagnostic assessment and interpretation of the leading model

The remaining figures examine prediction geometry, residual behavior, variable relevance, and the contrast between the leading estimator and the neural baseline. This diagnostic layer is important because a benchmark should not end with a rank ordering alone; it should also explain why the winning model is preferable from a regression-analysis standpoint.

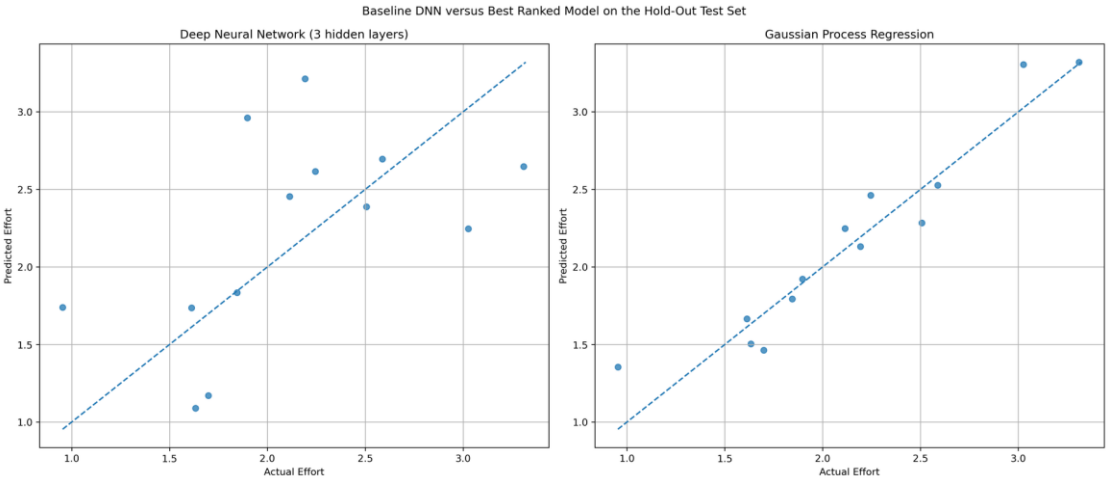

Figure 24. Hold-out prediction comparison between the deep neural network and Gaussian process regression.

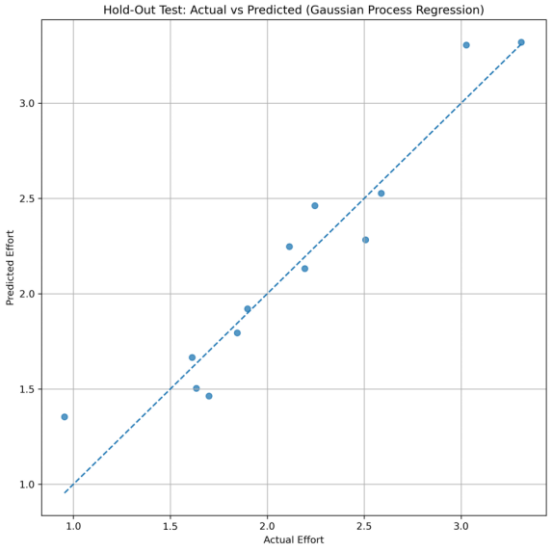

Figure 25. Actual versus predicted effort for Gaussian process regression on the hold-out test set.

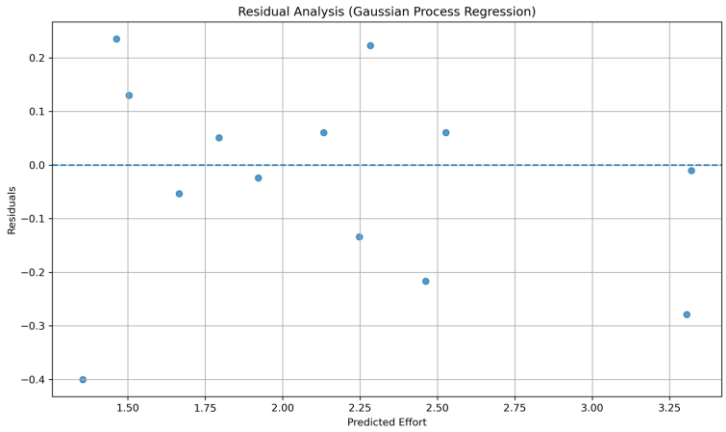

Figure 26. Residual analysis for Gaussian process regression on the hold-out test set.

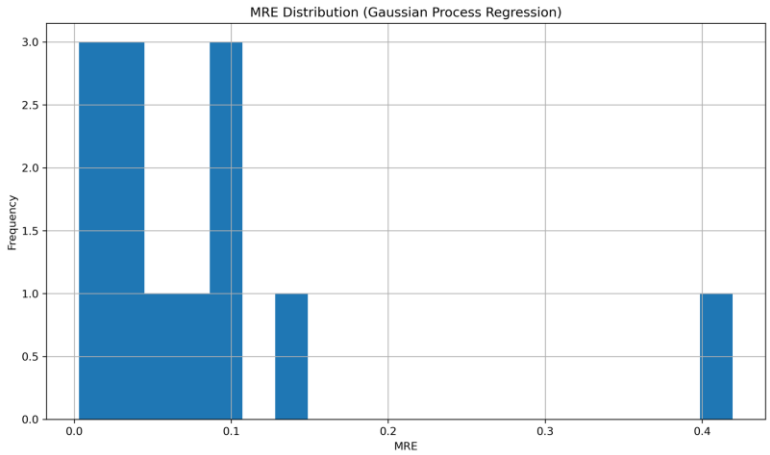

Figure 27. Distribution of the magnitude of relative error for Gaussian process regression.

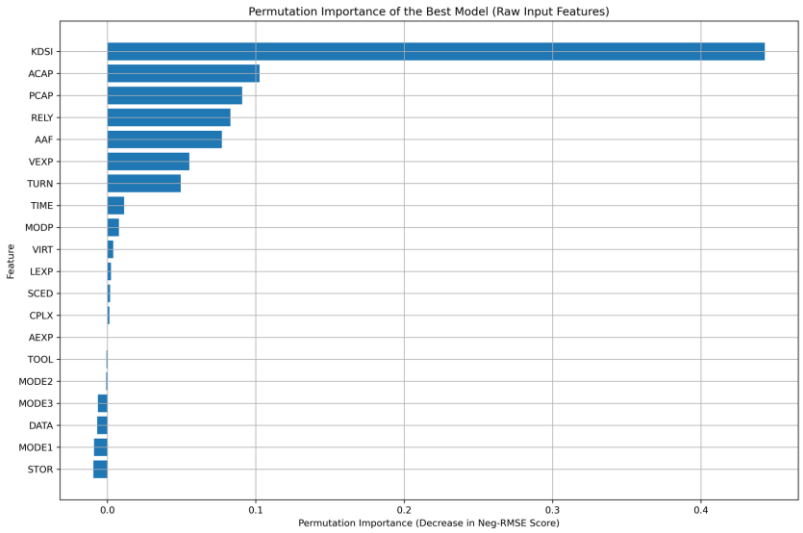

Figure 28. Permutation importance of the best model computed from hold-out RMSE degradation.

Table 6. Top predictors of the best model according to permutation importance.

Rank	Feature	Permutation importance	Std. dev.
1	KDSI	0.4430	0.1011
2	ACAP	0.1025	0.0438
3	PCAP	0.0909	0.0385
4	RELY	0.0828	0.0471
5	AAF	0.0771	0.0422
6	VEXP	0.0552	0.0268
7	TURN	0.0494	0.0328
8	TIME	0.0111	0.0227
9	MODP	0.0077	0.0269
10	VIRT	0.0039	0.0065

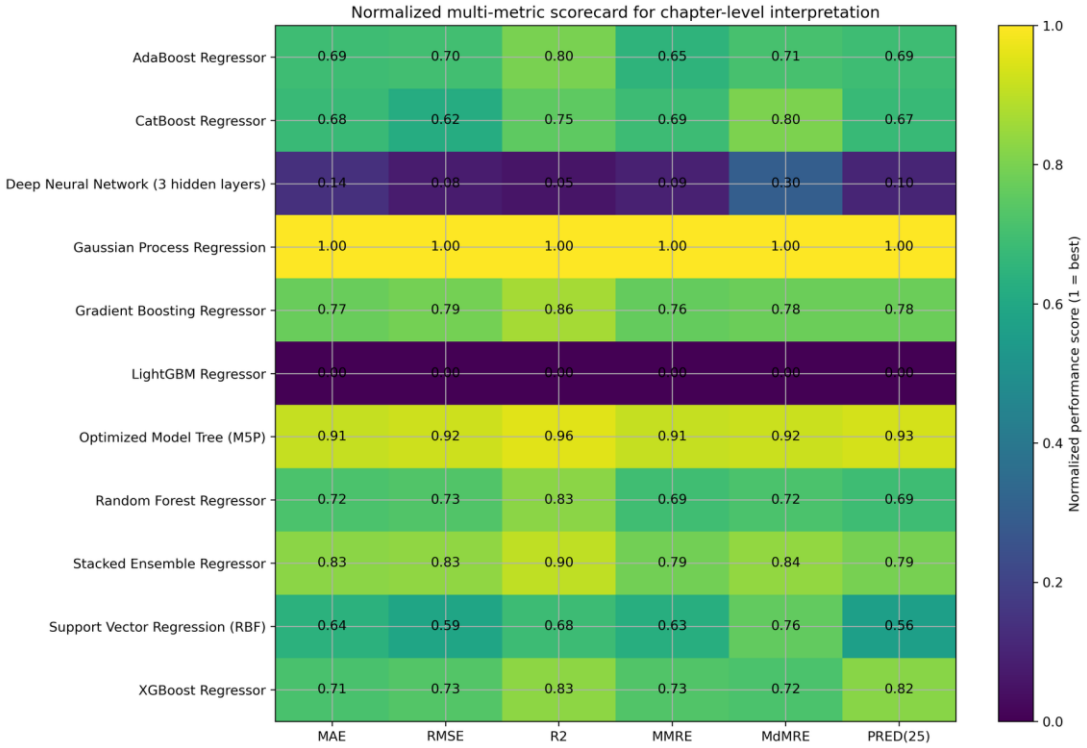

Figure 29. Normalized multi-metric scorecard derived from repeated cross-validation.

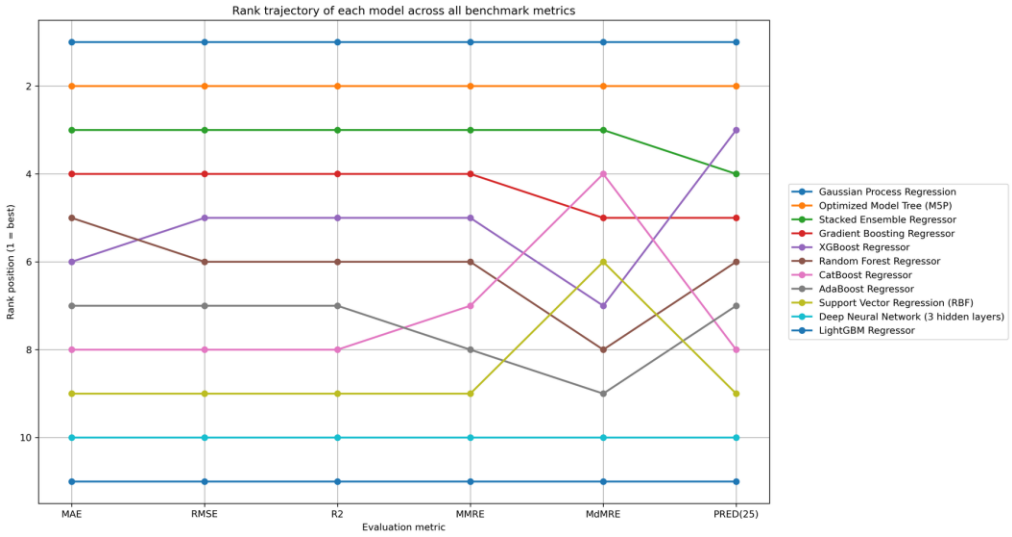

Figure 30. Changes in each model's rank across the six benchmark measures.

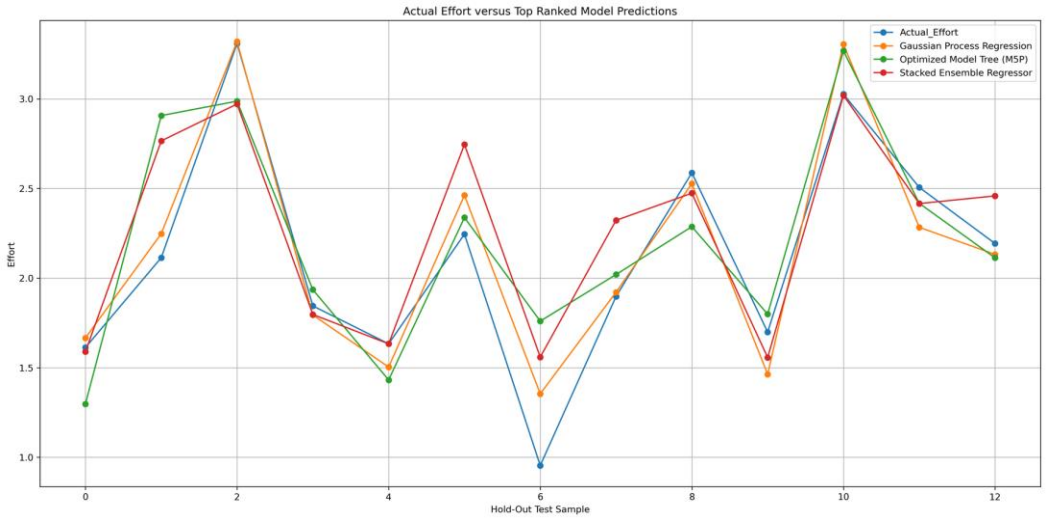

Figure 31. Observed effort and predictions from the leading hold-out models.

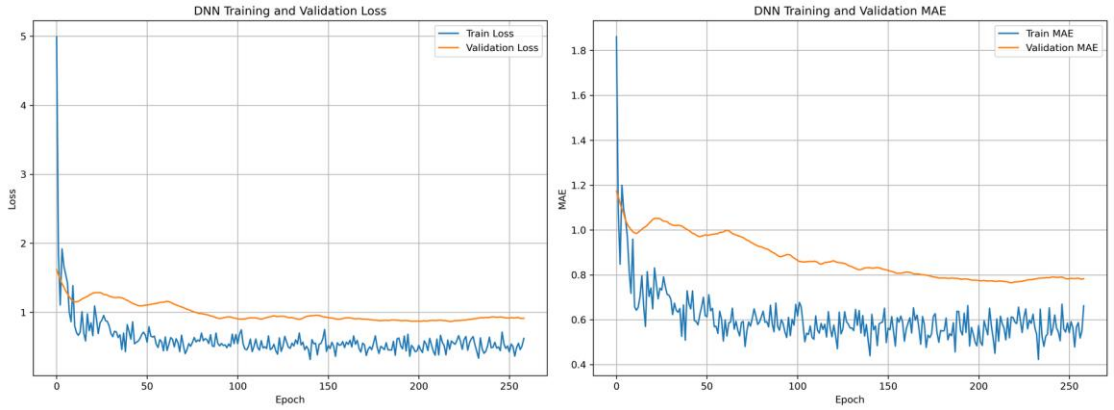

Figure 32. Learning curves for the deep neural-network baseline.

Figure 24 offers the most direct contrast between Gaussian process regression and the neural baseline. The Gaussian-process estimates cluster around the identity line, whereas the neural predictions show substantially larger departures. On the hold-out set, Gaussian process regression reduced RMSE by 69.49% and MAE by 70.93%, increased R^2 from 0.0158 to 0.9084, and raised PRED(25) by 38.46 percentage points. Figure 25 confirms that the advantage extends across the observed effort range.

In Figure 26, Gaussian-process residuals fluctuate around zero without a visible monotonic pattern. One low-effort project accounts for the largest overestimate, while the remaining errors occupy a comparatively narrow interval. Figure 27 tells the same story on a relative scale: most cases fall in the low-error bins and a single observation forms the upper tail. These patterns agree with MMRE = 0.0850, MdMRE = 0.0634, and PRED(25) = 0.9231 on the hold-out set.

Table 6 and Figure 28 attribute the largest permutation importance to KDSI, followed by ACAP, PCAP, RELY, AAF, and VEXP. This ordering is substantively plausible: project size establishes the dominant effort scale, while personnel capability, reliability requirements, and adjustment factors refine the prediction. These values describe predictive dependence, not causal effects. With only 13 hold-out cases, near-zero or slightly negative scores may reflect correlated inputs and sampling variability.

Figures 29 and 30 summarize the benchmark from two complementary views. In the normalized scorecard, Gaussian process regression is the only method that remains at the top for every criterion; M5P forms the second tier, followed by stacking and gradient boosting. The rank trajectories confirm this stability, while most competing methods move across metrics.

Figure 31 compares the leading hold-out predictions case by case. Gaussian process regression follows the observed series closely at low and medium effort levels, whereas the alternatives more often miss local peaks and troughs. Figure 32 helps explain the neural result: training loss declines quickly, but validation loss settles at a distinctly higher level, matching the generalization gap observed in Figures 4 and 5.

3.5 Limitations and scope of inference

The study is constrained by a very small working sample of 62 projects and by reliance on one legacy COCOMO-derived dataset. The 13-project hold-out set yields coarse PRED(25) increments of 1/13 and can produce unstable single-split rankings. In addition, the inputs are inherited effort multipliers and mode indicators rather than contemporary raw industrial descriptors. The reported ordering should therefore be read as evidence for this dataset and protocol, not as a universal hierarchy of regression algorithms. Validation on larger, independently curated industrial repositories, preferably with temporal or cross-organization testing and formal multiple-comparison procedures, is required before operational adoption.

4 Conclusions

The combined evidence supports a dataset-specific conclusion. When all eleven regressors are trained under the same leakage-safe protocol and ranked with the mean-rank rule of Eq. (9), Gaussian process regression is the best model for the present software effort estimation problem. Its superiority is not restricted to one metric: it dominates MAE, RMSE, R^2, MMRE, MdMRE, and PRED(25) in repeated cross-validation and retains first place on the independent hold-out test set.

Repeated cross-validation produced the following order: (1) Gaussian Process Regression, (2) Optimized Model Tree (M5P), (3) Stacked Ensemble Regressor, (4) Gradient Boosting Regressor, (5) XGBoost Regressor, (6) Random Forest Regressor, (7) CatBoost Regressor, (8) AdaBoost Regressor, (9) Support Vector Regression (RBF), (10) the three-hidden-layer Deep Neural Network, and (11) LightGBM Regressor. This sequence is the principal comparative result because it draws on ten validation folds and all six evaluation criteria, rather than on a single partition.

The independent test partition also places Gaussian process regression first, but it exposes the fragility of single-split conclusions. LightGBM moves from last in repeated validation to second on the 13-project hold-out set, a nine-rank shift that indicates substantial partition sensitivity. The neural network likewise underperforms under this protocol despite its expressive architecture and regularization. In a 62-project sample, the kernel and piecewise-linear/tree models control estimation variance more effectively than the dense network.

For portfolios comparable in size and structure to the present sample, Gaussian process regression is the strongest candidate when predictive accuracy is the principal criterion. M5P provides a defensible alternative when local linear explanations are required, while stacking and gradient boosting remain competitive choices where a probabilistic kernel model is operationally inconvenient. These recommendations are conditional on the dataset, preprocessing, and validation design used here.

Subsequent studies should repeat the comparison on larger industrial repositories, add formal multiple-comparison procedures such as Friedman–Nemenyi tests, evaluate predictive intervals, and test whether richer feature engineering or wider hyperparameter searches alter the ordering. For the present 62-project sample, the available evidence favors Gaussian process regression; that conclusion should be treated as conditional on this dataset and protocol rather than as a universal ranking.

5 Code and reproducibility resources

The complete implementation and reproducibility materials are available at:

https://github.com/JAIME6609/AFS/blob/main/CODIGO-AFS-IA-ART-00-VARIOS-MODELOS-V4.py

Declaration on generative AI and AI-assisted technologies

During revision, the authors used an AI-assisted language tool to support linguistic editing. All numerical results, references, methodological descriptions, and scientific interpretations were subsequently reviewed by the authors, who assume full responsibility for the published text.

References

Arora, S., & Mishra, N. (2017). Software cost estimation using single layer artificial neural network. *International Journal of Advanced Engineering Research and Science, 4*(9), 22–26. doi:10.22161/ijaers.4.9.6

Boehm, B. W. (1981). *Software engineering economics*. Prentice-Hall.

Boehm, B. W., Abts, C., Brown, A. W., Chulani, S., Clark, B. K., Horowitz, E., Madachy, R., Reifer, D. J., & Steece, B. (2000). *Software cost estimation with COCOMO II*. Prentice Hall.

Boehm, B. W., Abts, C., Brown, A. W., Chulani, S., Clark, B. K., Horowitz, E., Madachy, R., Reifer, D. J., & Steece, B. (2000). *Software cost estimation with COCOMO II*. Prentice Hall.

Breiman, L. (2001). Random forests. *Machine Learning, 45*(1), 5–32. doi:10.1023/A:1010933404324

Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 785–794). Association for Computing Machinery. doi:10.1145/2939672.2939785

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning, 20*, 273–297. doi:10.1007/BF00994018

Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *The Annals of Statistics, 29*(5), 1189–1232. doi:10.1214/aos/1013203451

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press.

Hoerl, A. E., & Kennard, R. W. (1970). Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics, 12*(1), 55–67. doi:10.1080/00401706.1970.10488634

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In F. Bach & D. Blei (Eds.), *Proceedings of the 32nd International Conference on Machine Learning* (Vol. 37, pp. 448–456). PMLR.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., & Liu, T.-Y. (2017). LightGBM: A highly efficient gradient boosting decision tree. *Advances in Neural Information Processing Systems, 30*, 3146–3154.

Kingma, D. P., & Ba, J. (2015). *Adam: A method for stochastic optimization*. International Conference on Learning Representations.

Kingma, D. P., & Ba, J. (2015). *Adam: A method for stochastic optimization*. International Conference on Learning Representations.

Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A. V., & Gulin, A. (2018). CatBoost: Unbiased boosting with categorical features. *Advances in Neural Information Processing Systems, 31*, 6638–6648.

Quinlan, J. R. (1992). Learning with continuous classes. In A. Adams & L. Sterling (Eds.), *AI '92: Proceedings of the 5th Australian Joint Conference on Artificial Intelligence* (pp. 343–348). World Scientific.

Rasmussen, C. E., & Williams, C. K. I. (2006). *Gaussian processes for machine learning*. MIT Press. doi:10.7551/mitpress/3206.001.0001

Smola, A. J., & Schölkopf, B. (2004). A tutorial on support vector regression. *Statistics and Computing, 14*(3), 199–222. doi:10.1023/B:STCO.0000035301.49549.88

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research, 15*, 1929–1958.

Wang, Y., & Witten, I. H. (1996). *Induction of model trees for predicting continuous classes* (Working Paper 96/23). University of Waikato, Department of Computer Science.

Wang, Y., & Witten, I. H. (1997). Induction of model trees for predicting continuous classes. In M. van Someren & G. Widmer (Eds.), *Poster papers of the 9th European Conference on Machine Learning (ECML'97)* (pp. 128–137). Laboratory of Intelligent Systems.

Wolpert, D. H. (1992). Stacked generalization. *Neural Networks, 5*(2), 241–259. doi:10.1016/S0893-6080(05)80023-1