Received: 2 November 2022 | Revised: 13 June 2023 | Accepted: 12 August 2023

DOI: 10.1002/smr.2611

RESEARCH ARTICLE - METHODOLOGY

Journal of Software: Evolution and Process

WILEY

Incorporating statistical and machine learning techniques into the optimization of correction factors for software development effort estimation

Ho Le Thi Kim Nhung[1] | Vo Van Hai[2] | Petr Silhavy[3] | Zdenka Prokopova[3] | Radek Silhavy[3]

[1]Faculty of Information Technology, University of Science–Vietnam National University, Ho Chi Minh City, Vietnam

[2]Faculty of Information Technology, Industrial University of Ho Chi Minh City, Ho Chi Minh City, Vietnam

[3]Faculty of Applied Informatics, Tomas Bata University in Zlín, Zlín, Czech Republic

Correspondence
Radek Silhavy, Tomas Bata University in Zlin, Faculty of Applied Informatics, Nad Stranemi 4511, 76001, Zlín, Czech Republic.
Email: rsilhavy@utb.cz

Funding information
Tomas Bata University in Zlín, Grant/Award Numbers: RVO/FAI/2021/002, IGA/CebiaTech/2022/001

Abstract

Accurate effort estimation is necessary for efficient management of software development projects, as it relates to human resource management. Ensemble methods, which employ multiple statistical and machine learning techniques, are more robust, reliable, and accurate effort estimation techniques. This study develops a stacking ensemble model based on optimization correction factors by integrating seven statistical and machine learning techniques (*K*-nearest neighbor, random forest, support vector regression, multilayer perception, gradient boosting, linear regression, and decision tree). The grid search optimization method is used to obtain valid search ranges and optimal configuration values, allowing more accurate estimation. We conducted experiments to compare the proposed method with related methods, such as use case points-based single methods, optimization correction factors-based single methods, and ensemble methods. The estimation accuracies of the methods were evaluated using statistical tests and unbiased performance measures on a total of four datasets, thus demonstrating the effectiveness of the proposed method more clearly. The proposed method successfully maintained its estimation accuracy across the four experimental datasets and gave the best results in terms of the sum of squares errors, mean absolute error, root mean square error, mean balance relative error, mean inverted balance relative error, median of magnitude of relative error, and percentage of prediction (0.25). The *p*-value for the *t*-test showed that the proposed method is statistically superior to other methods in terms of estimation accuracy. The results show that the proposed method is a comprehensive approach for improving estimation accuracy and minimizing project risks in the early stages of software development.

KEYWORDS
optimizing correction factors, software development effort estimation, staked generalization ensemble, statistical and machine learning techniques

1 | INTRODUCTION

The complexity of software project development has increased, and this industry demands a high level of competence from its employees, who must possess particular skills. Project managers typically need such early estimates to bid on a project contract and make informed planning decisions.[1] However, they often encounter difficulties in estimating effort, cost, and schedule correctly in advance. Customer requirements are volatile, inconsistent, and incomplete, that is, unknown. Therefore, a project manager must select an appropriate method and adapt or configure it to the software project the company wants to undertake to obtain accurate estimates. However, since insufficient information is usually available, the estimation process leads to a result subject to significant uncertainties.[2]

Software development effort estimation (SDEE) is one of the most challenging tasks in the early stages of software development. Effort estimation methods are used to reduce project risk and minimize the risk of surprises during the project. They provide project managers with informed control decisions to ensure that an appropriate amount of work is allocated to the various phases of the project development lifecycle. Therefore, accurate effort estimation is critical to minimizing project risk.[3] Several SDEE methods have been proposed, which can be classified into three main categories: (1) algorithmic, (2) non-algorithmic, and (3) statistical and machine learning (ML) methods.[4,5] Algorithmic methods are popular in the literature and use statistical and mathematical equations for SDEE, for example, use case point (UCP), functional point analysis (FPA), the cost constructive model (COCOMO-II), source line of code (SLOC), and the Putnam software life cycle model (SLIM). Non-algorithmic methods rely on analytical comparisons and historical projects for estimation, for example, analogy-based, expert judgment, and planning poker strategies. Statistical and ML models include fuzzy logic, artificial neural network, and hybrid models.[6] These models can be used as stand-alone models and require input variables to estimate software effort.

In the early phases of software development, the UCP method[7] was extensively studied as a functionally sized metric for predicting software effort.[8] Most researchers today focus on developing new methods based on this original method or validating existing methods in industrial applications, with an emphasis on improving accuracy. Basically, they apply statistical and ML techniques in these model variations to optimize estimation accuracy. The techniques are used to model the relationship between effort and software variables, especially when this relationship is non-linear. In recent decades, several statistical and ML techniques have been developed for effort estimation. Many of the proposed models have achieved high estimation accuracy.[9] Jorgensen et al[4] identified 11 ML techniques used in studies published up until 2004 and noted that regression techniques were used in 49% of the studies reviewed. Wen et al[10] also performed a systematic literature review of ML techniques used in SDEE covering the period 1990 to 2010. Their review indicated that the estimation accuracies obtained via ML techniques were greater than those obtained using non-ML-based estimation methods. According to Kumar et al,[11] the overall estimation accuracies of SDEE methods based on statistical and ML techniques are nearly in the acceptable range, as they are within 25% of the percent error (PRED [0.25]). Given the complexity of software development projects today, effort estimation requires ML assistance.[12] Therefore, based on their review, we summarize 21 selected recent studies[13–40] over the past 7 years (2016–2021) on software effort estimation using statistical and ML techniques or just ML techniques. Specifically, seven statistical and ML techniques, namely, multilayer perceptron (MLP), support vector regression (SVR), decision tree (DT), random forest (RF), multiple linear regression (MLR), *K*-nearest neighbor (KNN), and gradient boosting (GB), were discovered to have been most frequently used in SDEE at 16%, 13%, 13%, 11%, 9%, 7%, and 4% of studies, respectively (see Figure 1). The list of abbreviations in Figure 1 is presented in Appendix A.

Although statistical and ML techniques have been handled remarkably well, there have been some difficulties in choosing unbiased approaches and appropriate algorithms. First, selecting the proper statistical and ML techniques for SDEE is challenging. Generally, single statistical and ML methods are unreliable. Specifically, their estimation accuracies are inconsistent and unstable across different datasets and evaluation criteria.[41–43] According to Cabral et al,[44] the use of a single model does not lead to optimal results for SDEE. Priya et al[45] also pointed out that combining multiple models is more accurate. Second, it is well known that the accuracy of a single method depends on its parameter configurations.[46] Moreover, very few studies have used statistical tests to validate their results. It is not valid to claim that one model is better than another when adequate statistical tests are not performed.[47]

With such research motivation, we recently developed a parametric software effort estimation model based on optimizing correction factors (OCFs).[48,49] Specifically, the MLR model is applied to the OCF method to efficiently minimize the estimation error in the integration or recursion process. However, the method still needs to be improved to reach more comprehensive methods. The difference between previous works is that we continue developing our method OCF. The OCF method has investigated the least absolute shrinkage and selection operator (LASSO) method[50,51] to determine the best technical and environmental complexity factors that significantly affect the estimation accuracy of the UCP method. The novel in this paper is that the new ensemble-based OCF approach is studied. Its improvements proposed in this paper are put under a specific situation where popular statistical and ML techniques are incorporated into an ensemble effort estimation (EEE) based on the OCF method. The EEE approach combines at least two different single models to address the weaknesses of single models for estimation tasks through a unique aggregation mechanism and generate the final solution by weighted voting over their solutions.[52] Compared to the previous related methods, the new ensemble-based OCF approach will be unbiased in estimating the effort needed for a new software project. Our results confirm the findings of the previous review that ML remains the most common technique for generating EEE and that ensemble techniques have outperformed single models. Thus, the following three research questions will be addressed:

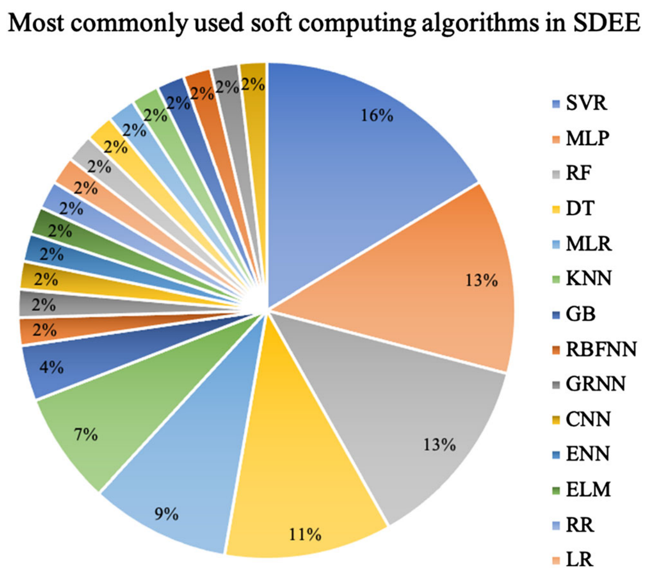

FIGURE 1 Most commonly used statistical and ML algorithms for SDEE over the past 7 years (2016–2021).

- **RQ1:** How much does the proposed ensemble-based OCF method improve upon the single methods used to produce it?
- **RQ2:** Are the differences in estimation accuracy between the proposed method and other methods statistically significant?
- **RQ3:** How much do the effects of the core components of the proposed method the estimation accuracy?

To answer the research questions, we conducted an empirical study to evaluate the estimation accuracies of the proposed method and methods found in the literature. We used evaluation criteria that yield unbiased and symmetric distributions,[49,53] such as the sum of squares errors (SSE), the mean absolute error (MAE), mean balance relative error (MBRE), mean inverted balance relative error (MIBRE), median of magnitude of relative error (MdMRE), root mean square error (RMSE), and the percentage of prediction within *x*% (PRED[*x*]). Finally, all experimental groups are compared using a statistical comparison. In this way, we aim to draw the most accurate conclusions about comparing methods. The statistical comparison includes parametric and non-parametric methods. In this study, we used both the *t*-test, a parametric statistical comparison, and the Mann–Whitney U test, a non-parametric statistical comparison.[54–56] These pairwise statistical comparisons include the averages (μs) of the evaluation results (SSE, MAE, MBRE, MIBRE, MdMRE, and RMSE) from the five-fold cross-validations of the four experimental datasets. The following statistical hypotheses were tested:

- $H_0 : \mu_{\text{the proposed method}} = \mu_{\text{the other tested methods}}$.

  In other words, the estimation ability of the proposed method is not significantly different from the estimation abilities of the other tested methods. In particular, the proposed method does not outperform the other tested methods in estimating software effort.
- $H_1 : \mu_{\text{the proposed method}} < \mu_{\text{the other tested methods}}$.

  In other words, the estimation ability of the proposed method is significantly different from the estimation abilities of the other tested methods. In particular, the proposed method outperforms the other tested methods in estimating software effort.

  Specifically, our main contributions are as follows:

- This study presents a novel SOCF method rooted in the Effort Estimation Ensemble concept. The SOCF method uniquely integrates the capabilities of seven established statistical and ML techniques: MLR, KNN, SVR, MLP, RF, GB, and DTs. The main aim is to lessen the biases and variability errors that are often found in individual models.
- A key aspect of the accuracy of this ensemble method lies in parameter tuning. The grid search (GS)[57] optimization method is used to determine the best parameters for each technique and dataset, with 20% of the training set serving as the validation set. Detailed information on the post-tuning parameters can be found in Section 3.
- The effectiveness of our proposed method is then compared with other estimation methods mentioned in previous studies.[58] This comparison utilizes four historical datasets from administrative, healthcare, and business sectors.[38] We carry out the comparison by implementing a five-fold cross-validation, leading to five random splits of the training and testing data. The findings are based on the average results each model obtains across all evaluation criteria.

- The results indicate that our innovative SOCF method significantly improves the accuracy of effort estimation in the early stages of software development while minimizing project risks. This showcases its comprehensive ability to enhance effort estimation.

The remainder of this paper is organized as follows: Section 2 presents the related works. Section 3 provides the methodologies used, such as an overview of the UCP and OCF methods, a background on the statistical and ML techniques used, and the configuration parameters for the statistical and ML techniques used. Next, Section 4 presents the proposed method for estimating effort. Then, Section 5 describes the experimental design, including the experimental process, the dataset, and the evaluation criteria/metrics. Section 6 focuses on the results, and Section 7 discusses the threats to validity. Finally, Section 8 discusses the conclusions and future work.

2 | RELATED WORKS

2.1 | Ensemble effort estimation problem formulation

Previous studies on statistical and ML techniques have shown that ensemble methods provide more accurate results than single methods.[59] These studies focused on various aspects of effort estimation, such as the diversity of base models, the ranking of models within the ensemble, aggregation techniques, and model selection. The ensemble learning approach in SDEE does well when the base models perform differently on different datasets,[60] that is, it minimizes model limitations and leads to more accurate estimates. The authors investigated the ranking stability and ensemble approach across 90 methods and 20 datasets. They concluded that the ensemble approaches were consistently better, were more reliable, and had lower error estimates.[61] Pahariya et al[62] agreed that ensemble models are superior to single methods. Azzed et al mentioned the importance of the ensemble approach in analogy-based estimation.[13] The findings demonstrate that these ensemble methods perform better than single models and produce more accurate estimates of error measurements.

The problem with the ensemble approach is selecting single appropriate methods that must meet the high accuracy and diversity criteria to receive a high estimate.[13,42,43,63,64] In other words, a single method must be versatile and accurate under certain conditions. In this way, every single method compensates for the estimation errors of the others. Otherwise, an ensemble approach that does not contain different single methods may have a lower estimation accuracy than its single method. The EEE architecture is shown in Figure 2, where X denotes the feature vector of the underestimated project. All other single estimation algorithms $M_1, M_2, \ldots, M_n$ are given the same feature vectors and estimates $\hat{y}_{M_1}, \hat{y}_{M_2}, \ldots, \hat{y}_{M_n}$. Based on the estimates provided by each single estimation algorithm, the ensemble aggregator $\int$ aggregates the estimates using combination rules (mean, median, and IRWM with weights $w_{M_1}, w_{M_2}, \ldots, w_{M_n}$). Finally, an overall ensemble estimate $\hat{Y}_{ensemble}$ is provided for the project.

2.2 | Ensemble effort estimation methods

An ensemble method is proposed for identifying the best-performing regression-based ML model across various datasets.[14] In this method, the AdaBoost ensemble approach is used to create combinations of two statistical and ML techniques (KNN, SVR, and DT) with which to estimate the UCP-based effort. These adaptive UCP (AUCP) models are then compared with the ML models to determine whether they can improve estimation accuracy. The results show that the best ensemble model performs best overall, with a regression rate of over 98% across two datasets.

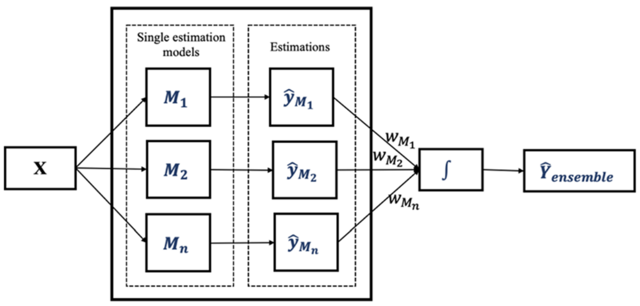

FIGURE 2 The architecture of the ensemble effort estimation.

Effective and practical approaches are proposed for deploying and maintaining ML.[15] Specifically, an ensemble of three statistical and ML algorithms (SVR, MLP, and GLM) is presented for estimating the effort put forth during and duration of the initial phase of a project. The results show that the ensemble model is more accurate than other approaches and suitable for practical use. An ensemble of optimal trees is developed for SDEE.[16] The results show that the RF model outperforms the random tree (RT) model for all datasets except the Desharnais dataset, where their PRED(0.25) values are equal. However, the developed ensemble model consistently has a smaller mean magnitude relative error (MMRE) than the RT and RF models across five datasets.

An ensemble model that combines UCP, expert judgment, and case-based reasoning (CBR) techniques is proposed to improve estimation accuracy in software development.[17] Specifically, UCP, expert judgment, and CBR produce independent variables, whereas effort is the dependent variable. The estimation results of the three basic models are combined into an ensemble using combination rules (mean, median, and inverse rank-weighted mean). An ensemble framework is presented for effort estimation using ML algorithms to obtain better accuracy estimates for error measurements.[18] The framework, which is based on an enhanced RF algorithm, succeeded in this task when it was compared to existing effort estimation methods.

The authors compared five statistical and ML models (MLP, RF, RT, KNN, and SVR) with a voting ensemble model for estimating software development effort over five datasets.[19] For the ensemble model, these models are combined using a combination rule based on the median of their estimated values. The results confirmed that the single models are unreliable because their estimation accuracies are inconsistent and unstable across different datasets. However, the ensemble model outperformed the single models on three of the five datasets. The authors proposed and evaluated heterogeneous ensembles based on KNN, SVR, MLP, and M5Prime using three combination rules (average, median, and inverse rank-weighted mean).[20] The methods were evaluated based on standardized accuracy (SA), effect size, and PRED(0.25) using the leave-one-out cross-validation (LOOCV) method.[21,22] The Scott–Knott statistical test was also conducted to determine significant differences in accuracy among the methods.

The authors experimented with SVR, RR, KNN, DT, and Bayesian networks to determine the method that provides better accuracy in estimating software effort.[42] The results show that none of the methods uniformly performs better. Therefore, an ensemble-based approach was proposed that outperformed the other similar approaches in terms of estimation accuracy. The article of Kumar et al[65] proposed an ensemble learning method, a gradient-boosted regression model. Accuracy comparisons are made with regression models such as KNN, DT, RF, and AdaBoosted regressors. The models are evaluated using evaluation metrics such as MAE, MSE, RMSE, and R^2. The results show that the ensemble learning method performs well on all the individual models used compared to both datasets, achieving 98% accuracy on COCOMO81 and 93% on the CHINA dataset. The authors conducted a comparative study of 12 ensemble methods for effort estimation. With an MMRE value of 10% and a PRED(0.25) of 97%, the M5 rule ensemble was found to be the best way to estimate effort.[66]

The above are some experimental studies on ensemble methods that are performed from time to time. Table 1 summarizes other related work on software effort estimation of various single methods using known datasets and real-time industrial projects, and the performance metrics were evaluated to determine the best model for estimating effort accuracy.

3 | METHODOLOGIES USED

The methodologies used in this paper to estimate the required effort are described below.

3.1 | Use case points

The UCP method[7] is used to estimate the size of object-oriented software projects. The UCP is calculated by converting the elements of the UML use case diagram into size measures according to a well-defined procedure. In the first step, the actor elements are categorized according to their level of difficulty: simple, average, and complex, as shown in Table 2. The unadjusted actor weight (UAW) is calculated in Equation (1).

$$UAW = \sum_{i=1}^{3} at_i \times w_i \tag{1}$$

The use case elements are categorized into three categories (simple, average, and complex) according to the number of transactions mentioned in the use case description, as shown in Table 3. The unadjusted use case weight (UUCW) is calculated in Equation (2).

$$UUCW = \sum_{j=1}^{3} uc_j \times w_j, \tag{2}$$

TABLE 1 Summaries of some important published research works on software effort estimation using statistical and ML methods or just ML techniques (2016 onward).

Statistical and ML techniques used	Algorithm comparison	Evaluation criteria	Best algorithm	Databases used
KNN, DT, RF, AB, and GB	Algorithms were compared to each other over two datasets	MAE, MSE, RMSE, and R^2	GB has optimal performance of 98% with COCOMO781 and 93% with CHINA.[66]	COCOMO81 (63 projects) and CHINA (499 projects)
KNN, SVR, DT, and GB	Algorithms were compared to each other over two datasets	MSE and R^2	The ensemble model GB performs best, with a regression rate of over 98% for both datasets.[14]	DS1 (71 projects) and DS2 (29 projects)
SVR, DT, and MLR	Algorithms were compared to each other over four datasets	MAE, MSE, MAPE, and RMSE	SVR (COCOMO NASA1 with MAE: 31.5, MSE: 2755.5, MAPE: 0.3, and RMSE: 52.5), SVR (COCOMO NASA2 with MAE: 262.6, MSE: 158576.6, MAPE: 2.1, and RMSE: 398.2), DT (COCOMO81 with MAE: 943.3, MSE: 5193249, MAPE: 4.8, and RMSE: 2278.9), MLR (Kaushik with MAE: 14.8, MSE: 468.8, MAPE: 0.1, and RMSE: 26.6).[23]	COCOMO NASA1 (60 projects), COCOMO NASA2 (93 projects), COCOMO81 (63 projects), and Kaushik (15 projects)
KNN, CNN, and ENN	Algorithms were compared to each other	MMRE, RMSE, and BRE	KNN (with MMRE: 0.1, RMSE: 0.5, and BRE: 0.2).[24]	NASA (60 projects)
ELM	Algorithm was compared with SVR, MLP, DT, and RF	MAE, RMSE, MBRE, MIBRE, and SA	ELM (with MAE: 2310.7, RMSE: 3350.8, MBRE: 1.619, MIBRE: 0.432, and SA: 60.1).[25]	ISBSG (926 projects)
SR	Algorithm was compared with UCP	MAR, MMRE, PRED(0.25), MAPE, SSE, and MSE	SR (with MAR: 17.9, MMRE: 0.05, PRED(0.25): 1, MAPE: 5.9, SSE: 13.6, and MSE: 649.7).[26]	70 projects
MLR, MLP, and RF	Algorithms were compared to each other	MAR, RMSE, RRAE, and RSE	MLR (with MAE: 2135.3, RMSE: 2906.7, RRAE: 66.9%, and RSE: 65.55%).[27]	Desharnais dataset
MLR, RR, LR, ENN, RF, SVR, DT, and MLP	Algorithms were compared to each other over four datasets	MAE, MSE, RMSE, and R^2	SVR (Desharnais with MAE: 1888.1, MSE: 5576003, RMSE: 2361.4, R^2: −0.02), SVR (Albrecht with MAE: 3.5, MSE: 15.2, RMSE: 3.9, R^2: 0.94), ENN (Maxwell with MAE: 3113.2, MSE: 20578229, RMSE: 4536.3, R^2: 0.73), LR (China with MAE: 330.7, MSE: 293412.4, RMSE: 541.6, R^2: 0.99).[28]	Desharnais (81 projects), Maxwell (62 projects), China (499 projects), and Albrecht (24 projects)
SVR, NN, and GLM	Algorithms were compared to each other	MMRE, PRED(0.25), and RMSE	The ensemble methods of SVR, NN, and GLM perform best, with MMRE: 12.5, PRED(0.25): 1, RMSE: 0.19.[29]	ISBSG
MLR, KNN, SVR, and MLP	Algorithms were compared to each other	R^2	The accuracy of MLP is better than the other methods with R^2: 79%.[30]	Desharnais (81 projects)
MLP, GRNN, RBFNN, CCNN, ANFIS, and SVR	Algorithms were compared to each other	MAE, MBRE, and MIBRE	GRNN with MAE: 1243.9, MBRE: 28.7, and MIBRE: 17.8.[31]	234 industrial and educational software projects
DT and RF	Algorithms were compared to each other	MMRE, MdMRE, and PRED(0.25)	RF with MMRE: 1.29, MdMRE: 0.37, and PRED(0.25): 0.4.[32]	456 projects

TABLE 1 (Continued)

Statistical and ML techniques used	Algorithm comparison	Evaluation criteria	Best algorithm	Databases used
SVR, MLP, and DT	Algorithms were compared to each other	MAR and MdAR	MLP with MAR: 0.42, and MdAR: 0.34.[33]	ISBSG
NBL and RF	Algorithms were compared to each other	MRE	RF with MRE: 0.14.[34]	NASA (93 projects)
RF, MLP, and SVR	Algorithms were compared to each other	MAE, MMRE, and PRED(0.25)	RF with MAE: 0.10, MMRE: 0.30, and PRED(0.25): 0.72.[35]	214 projects
SVR, KNN, and AB	Algorithms were compared to each other	R^2	AB ensemble has good accuracy, with 91.35% on Desharnais and 85.48% on Maxwell.[36]	Desharnais (81 projects) and Maxwell (62 projects)
DT, GB, and RF	Algorithms were compared to each other	MAE, MMRE, and PRED(0.25)	GB with MAE: 0.16, MMRE: 0.11, and PRED(0.25): 0.85.[37]	21 projects
SR and MLR	Algorithm was compared to UCP	R^2, MSE, SSE, and RMSE	SR (DS1 with R^2: 0.73, MSE: 1843.5, SSE: 23966.1, RMSE: 42.9), SR (DS2 with R^2: 0.90, MSE: 126.2, SSE: 7069.1, RMSE: 11.2).[38]	DS1(28 projects) and DS2 (71 projects)
RF, MLP, and RBFN	Algorithm was compared to MLP, RBFN, SGB, and log-linear regression	MMRE and PRED(0.25)	RF with MMRE: 0.33, PRED(0.25): 0.68.[39]	149 projects
Ensemble model (SVR and RBFNN)	Algorithm was compared to UCP	MAE, MBRE, and MIBRE	Ensemble model (D1 with MAE: 1219.8, MBRE: 15.6, MIBRE: 11.8), Ensemble model (D2 with MAE: 201.1, MBRE: 17.4, MIBRE: 12.7), Ensemble model (D3 with MAE: 1564.8, MBRE: 19.7, MIBRE: 15.1).[40]	D1 (45 industrial projects), DS2 (65 educational projects), DS3 (merging DS1 & DS2)

TABLE 2 Actor classifications and their complexity weights.

Actor classification	Description	Weight
Simple	The system through an API	1
Average	The system through a protocol	2
Complex	The system through a GUI	3

TABLE 3 Use case classifications and their complexity weights.

Use case classification	Description	Weight
Simple	(0, 4)	1
Average	<4, 7>	2
Complex	(7, ∞)	3

TABLE 4 Technical complexity factors.

T_i	Description	Weight (Wt_i)
T_1	Distributed system	2
T_2	Response adjectives	2
T_3	End-use efficiency	1
T_4	Complex processing	1
T_5	Reusable code	1
T_6	Easy to install	0.5
T_7	Easy to use	0.5
T_8	Portability	2
T_9	Easy to change	1
T_{10}	Concurrency	1
T_{11}	Security features	1
T_{12}	Access for third parties	1
T_{13}	Special training facilities	1

where αt_i is the number of actors in actor element i, w_i is the complexity weight of actor i, uc_j is the number of use cases in use case element i, and w_j is the complexity weight of use case j.

The TCF and ECF correction factors are used to describe the experience level of the software development team. The TCF is calculated based on 13 technical factors (F1, F2, …, F13) significantly affecting project performance (see Table 4). The ECF is calculated based on eight environmental factors (E1, E2, …, E8) significantly affecting productivity (see Table 5). Each element in both groups can take an influence value between 0 and 5 and predefined weights representing the influence of each factor. Equations (3) and (4) show how to calculate TCF and ECF.

$$TCF = 0.6 + 0.01\sum_{i=1}^{13} T_i \times Wt_i, \quad (3)$$

$$ECF = 1.4 - 0.03\sum_{i=1}^{8} E_i \times We_i, \quad (4)$$

where T_i is the value of technical factor i, Wt_i is the complexity weight of technical factor i, E_i is the value of environmental factor i, and We_j is the complexity weight of environmental factor i.

The final UCP is computed according to Equation (5).

$$UCP = (UAW + UUCW) \times TCF \times ECF \quad (5)$$

For SDEE, Karner proposed a factor of 20 person-hours per UCP to measure software effort, as shown in the following Equation (6).

TABLE 5 Environmental complexity factors.

E_i	Description	Weight (We_i)
E_1	Family with RUP	1.5
E_2	Application experience	0.5
E_3	Object-oriented experience	1
E_4	Lead analyst capability	0.5
E_5	Motivation	1
E_6	Stable requirements	2
E_7	Part-time workers	−1
E_8	Difficult programming language	2

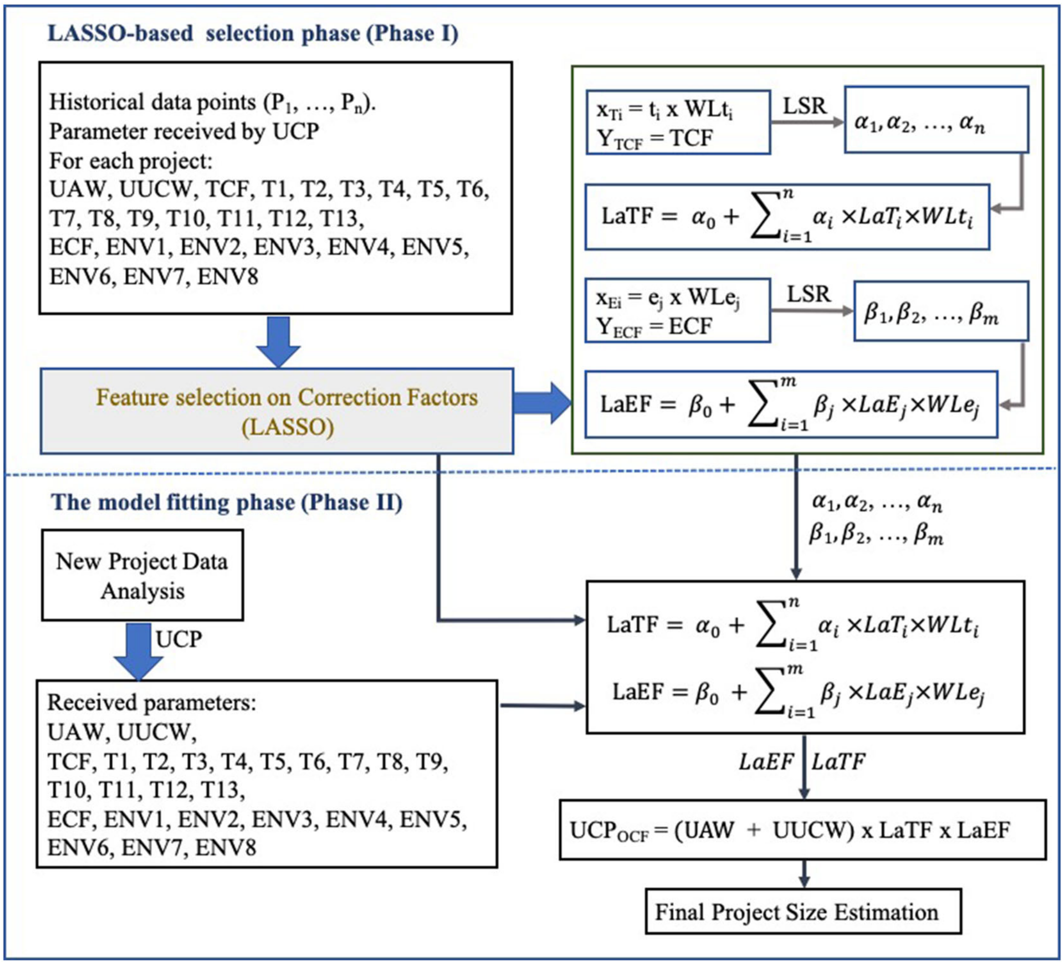
LASSO-based selection phase (Phase I)
Historical data points (P1, …, Pn).
Parameter received by UCP
For each project:
UAW, UUCW, TCF, T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12, T13, ECF, ENV1, ENV2, ENV3, ENV4, ENV5, ENV6, ENV7, ENV8
Feature selection on Correction Factors (LASSO)
x_Ti = t_i x WLt_i
Y_TCF = TCF
LSR
α1, α2, …, αn
LaTF = α0 + Σ αi × LaTi × WLti
x_Ei = e_j x WLe_j
Y_ECF = ECF
LSR
β1, β2, …, βm
LaEF = β0 + Σ βj × LaEj × WLej
The model fitting phase (Phase II)
New Project Data Analysis
UCP
Received parameters:
UAW, UUCW,
TCF, T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12, T13,
ECF, ENV1, ENV2, ENV3, ENV4, ENV5, ENV6, ENV7, ENV8
α1, α2, …, αn
β1, β2, …, βm
LaTF = α0 + Σ αi × LaTi × WLti
LaEF = β0 + Σ βj × LaEj × WLej
LaEF LaTF
UCP_OCF = (UAW + UUCW) x LaTF x LaEF
Final Project Size Estimation

FIGURE 3 Detailed illustration of the optimization correction factor (OCF) method.

$$\textbf{\textit{Effort}} = \textbf{\textit{UCP}} \times 20 \tag{6}$$

3.2 | OCFs

Our method, the OCF method,[48] uses the LASSO method[50,51] to select the best correction factors, thus reducing the risk involved in evaluating these factors using the UCP method. Figure 3 represents a detailed illustration of the OCF method.

Phase 1: The LASSO regression model is used to determine the correction factors for regression analysis. The LASSO estimate $\widehat{\beta}(\lambda)$ is given as follows:

$$\widehat{\beta}(\lambda) = \underset{\beta}{argmin}\left(\frac{\| Y - X\beta\|_2^2}{n} + \lambda \| \beta\|_1\right)$$
$$subject\ to \sum_{j=1}^{k} |\beta_j| < t, \tag{7}$$

where

$$\| Y - X\beta\|_2^2 = \sum_{i=0}^{n} (Y_i - (X\beta)_i)^2, \tag{8}$$

$$\| \beta\|_1 = \sum_{j=1}^{k} |\beta_j|, \tag{9}$$

where $\lambda \geq 0$ is the LASSO parameter, which controls the strength of the penalty determined by the LOOCV method.[21,22] LASSO parameter selection is based on the lowest possible estimation error and a lack of bias with respect to the correction factors for the observations in the training set. The LASSO parameter is directly related to the number of correction factors selected over non-zero β.

Next, least squares regression (LSR) is used to obtain the regression coefficients for the selected technical and environmental variables. Based on LASSO, the lines for the n selected technical factors (LaTF) and m environmental factors (LaEF) are given in Equations (10) and (11), respectively.

$$LaTF = \alpha_0 + \sum_{i=1}^{n} \alpha_i \times LaT_i \times WLt_i, \tag{10}$$

$$LaEF = \beta_0 + \sum_{i=1}^{m} \beta_i \times LaE_i \times WLe_i, \tag{11}$$

where LaT_i and LaE_j are the technical and environmental factors, respectively, that take values from the interval [0, 5]; WLt_i and WLe_j are the weight of these factors; and α_0, α_i, β_0, and β_i are the regression coefficients for the LSR model.

Phase 2: The OCF size is calculated as follows:

$$UCP_{OCF} = (UAW + UUCW) \times LaTF \times LaEF. \tag{12}$$

3.3 | Statistical and ML techniques

3.3.1 | MLP

MLP is a feedforward neural network used to solve regression problems that is usually trained with a backpropagation algorithm. The simplest MLP model consists of at least three nodes, an input layer, a hidden layer, and an output layer.[67] The number of independent variables in the input pattern is equal to the number of nodes in the input layer. Each neuron in the hidden layer converts the values from the previous layer via a weighted linear summation utilizing a nonlinear activation function. The number of nodes in the output layer depends on the problem under consideration and the number of dependent variables.

In this work, the OCF&MLP structure includes an input layer, a hidden layer MLP, and an output layer. The input layer neurons represent the variables identified with OCF. The output layer is the software size ($UCP_{OCF\&MLP}$) and receives values from the hidden layer to calculate the output value. One of the essential steps in developing the MLP is the optimization of its configuration parameters, such as the number of neurons in the hidden layer, and the three parameters of the learning algorithm (initial learning rate, momentum, and the regularization term). According to Linoff et al,[68] the number of nodes in the hidden layer should be between the number of nodes in the input layer and twice this number. In the OCF&MLP, the number of hidden nodes is between five and eight because four OCF variables are input. In this study, the Stochastic Gradient Descent (SGD) algorithm is used to train the MLP model.[69] The critical parameters for constructing the MLP model and their values for preliminary execution are depicted in Table 6.

TABLE 6 The parameters for constructing the MLP model and their values for preliminary execution.

Model parameter	Search range
Initial learning rate	$L = \{0.01, 0.02, 0.03, 0.04, 0.05\}$
Number of hidden nodes	$H = \{5, 6, 7, 8\}$
Momentum	$M = \{0.1, 0.2, 0.3, 0.4, 0.5\}$
Regularization term	$\alpha = \{0.00001, 0.0001, 0.001, 0.01\}$

3.3.2 | Support vector regression

Support vector machine (SVM) is a supervised learning method based on statistical learning theory.[70] SV regression (SVR) is a special form of SVM used to model the input–output functional relationship or regression. Assume that the training dataset is $D = \{(x_i, y_i)\}_1^n$, where $x_i \in \mathbb{R}^m$ denotes the input values, $y_i \in \mathbb{R}$ denotes the corresponding output values, n is the number of samples in the training dataset, and m is the dimension of the input dataset.

The goal of SVR is to approximate the nonlinear relationship shown in Equation (13) that brings $f(x_i)$ as close as possible to the obtained target value (y_i).

$$y_i = f(x_i) = \langle w, \Phi(x_i) \rangle + b, \tag{13}$$

where $w \in \mathbb{R}^m$ and $b \in \mathbb{R}$ are the weight vector and threshold, respectively; $\langle .,. \rangle$ denotes the dot product, and $\Phi(x_i)$ is the transformation function that maps the input values from $\mathbb{R}^m$ space to a feature space of higher dimension. The values w and b are reduced to ensure that the approximated function satisfies the above objective.

$$min_{w,\xi,\xi^*} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \xi_i + \xi_i^*, \tag{14}$$

$$\begin{gathered} \text{subject to} \\ y_i - \langle w, \Phi(x_i) \rangle - b \le \varepsilon + \xi, i = 1, \dots, n \\ \langle w, \Phi(x_i) \rangle + b - y_i \le \varepsilon + \xi^*, i = 1, \dots, n \\ \xi \ge 0, i = 1, \dots, n \\ \xi^* \ge 0, i = 1, \dots, n, \end{gathered} \tag{15}$$

where ε is the deviation of function $f(x_i)$ and ξ and ξ^* are slack variables used to measure ε. The regularization parameter C defines the error tolerance over ε.

In this work, $\varepsilon - SVR$ is used as a variant of SVR, and the radial basis function (RBF) is used as a kernel function.[71] The RBF kernel is calculated as

$$K(x_i, x_j) = \exp\left(-\gamma \|x_i - x_j\|^2\right), \gamma > 0. \tag{16}$$

Three parameters that significantly affect the performance of the $\varepsilon - SVR$ generalization, namely, the C, ε, and γ, must be carefully selected. Table 7 shows the details of these configuration parameters and their search ranges for the SVR method.

3.3.3 | DT

DTs are supervised ML algorithms used to solve regression and classification problems.[72] A DT creates a flowchart in an inverted tree-like structure, where the internal nodes illustrate the test, the branches define the test results, and each leaf node denotes a class label.[73] The output of a given DT is partitioned into distinguishable leaf nodes, following certain conditions, such as an if/else loop. There are many DT algorithms, such as ID3, CART, CHAID, C4.5, M5P, and REPTrees.[74,75] The DTs used in this study are optimized versions of the CART algorithm.

For any DT, we looked at four parameters: (1) the maximum depth (max_depth)—if this depth is not specified, the tree expands until the last leaf nodes contain a single value, resulting in overfitting; (2) the minimum number of leaf nodes (min_samples_leaf) in a decision tree, which is used to control the complexity of the model; (3) the minimum weighted fraction of the sum total of weights (min_weight_fraction_leaf) required

TABLE 7 The parameters for constructing the SVR model and their values for preliminary execution.

Model parameter	Search range
Regularization term (C)	$C = \{5, 10, 100, 150\}$
Epsilon for $\varepsilon - SVR$ (ε)	$\varepsilon = \{1, 0.1, 0.01, 0.001, 0.0001\}$
Kernel coefficient (γ)	$\gamma = \{1, 0.1, 0.01, 0.001, 0.0001\}$

TABLE 8 The parameters for constructing the DT model and their values for preliminary execution.

Model parameter	Search range
The maximum depth of the tree	max_depth = {3, 5, 7, 9, 11, 12}
The minimum weighted fraction	min_weight_fraction_leaf = {0.1, 0.2, 0.3, 0.4, 0.5}
The number of leaf nodes	max_leaf_nodes = {10, 20, 30, 40, 50, 60, 70, 80, 90}
The minimum number of samples	min_sample_nodes = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

TABLE 9 The parameters for constructing the RF model and their values for preliminary executions.

Model parameter	Search range
The number of trees	n_estimators = {100, 150, 200, 150, 300, 350, 400, 450}
The minimum number of samples	min_sample_nodes = {1, 2, 4}
The maximum depth of the tree	max_depth = {10, 20, 30, 40, 50, 60, 70, 80, 90, 100}

TABLE 10 The parameter for constructing the KNN model and its values for preliminary executions.

Model parameter	Search range
Number of neighbors (*K*)	n_neighbors = {2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15}

at a leaf node; and (4) the number of leaf nodes (max_leaf_nodes) to control overfitting. Values that are too high can lead to under-fitting. Table 8 provides details concerning the parameters for the DT model and their search ranges.

3.3.4 | RF

The RF technique uses a supervised nonparametric approach for regression and classification.[76] It creates multiple DTs and combines them to obtain a more accurate and stable prediction. The RF result is the maximum vote from a panel of independent judges, which makes the final prediction better than the best judge. In this research, we also focus on the parameters used in building an RF model, as in the DT model. Optimal RF parameters either increase the model's predictive power or facilitate its training. The robustness and stability of the prediction depend on these parameters.[39] The optimal parameters for the RF method for each experimental dataset are listed in Table 9.

3.3.5 | KNN

KNN is a non-parametric ML method used in classifications and regressions. KNN collects historical data, called the training dataset, and produces estimates for new test data.[76] The *K*-nearest data from the training data set is determined, and then, based on the data attributes of these data, an estimate is made of the new data. In KNN, the selection of *K* (number of neighbors) is very crucial. If the value of *K* is too small, the algorithm becomes sensitive to noise, whereas if the value of *K* is too large, data from other classes can be counted as nearest neighbors.[77] We apply GS to optimize *K* in this study. Table 10 shows the values of its search range. The Euclidean, Manhattan, and Minkowski distance metrics can all be used to measure the distance between points in KNN. We use the default Euclidean distance in scikit-learn. The Euclidean distance $d(p_i, q_i)$ between one vector $p = (p_1, p_2, \ldots, p_n)$ and another vector $q = (q_1, q_2, \ldots, q_n)$ can be computed as follows:

$$d(p_i, q_i) = \left[\sum_{i=1}^{n} (p_i - q_i)^2\right]^{1/2}. \tag{17}$$

3.3.6 | GB

GB is an ML technique used in regression and classification tasks. It is basically an ensemble method based on DTs.[78] In GB, the number of decision trees (number of estimators) is a crucial parameter. The higher the number of trees, the better the data will be learned. However, a large number of trees can significantly slow the training process down. Therefore, a parameter search is necessary. In this study, the three other parameters of interest in GB are the number of boosting stages (n_estimators), the minimum number of leaf nodes (min_samples_leaf), and the maximum depth of the single regression estimators (max_depth), which is used to control model overfitting. The details of the search range for, and optimal values of, the parameters for the OCF&GB method over GS for all datasets can be found in Table 11.

3.4 | Setting configuration parameters

The accuracy of a particular statistical or ML technique depends on the configuration parameters describing the characteristics of a specific dataset. Determining a technique's optimal parameter values gives it a high predictive capacity. In this study, we use GS[57] to optimize the configuration parameters of each statistical and ML technique. Specifically, GS exhaustively searches each empirical method's parameter set across a predefined range of values for each dataset and then selects the configuration that yields the "optimal" estimates. The parameter search ranges are derived from previous analyses.[30,31,35] In each case, we broadened the search range to consider as many possible configurations as possible (see Tables 3–8). Each method's optimization convergence depends on the mean square error (MSE) reaching 0 or the maximum number of iterations reaching 10,000.[79] The parameters are tuned to the validation set, which represents 30% of the training set. The detailed optimal parameter values for the estimation methods for each dataset are listed in Tables 12, 13, 14, and 15.

4 | PROPOSED STACKED OFC METHOD (IN FULL)

In this section, we present our proposed OCF-based stacked generalization ensemble method of statistical and ML models, which we have named stacked OCF (SOCF). In this study, the staked generalization (staking) ensemble[41,42] was used to estimate the OCF-based size. Recall that the main goal of this study was to use the capabilities of a group of robust single estimators in a regression task to provide estimates that are more accurate than those produced by any single model in the ensemble. The ensemble was trained and tested on the four datasets, D1–D4.

Figure 4 shows the detailed SOCF architecture, which consists of steps to clean the data, split the data into training and test datasets, and apply the stacking model to estimate the OCF-based size. The following methodology was used:

TABLE 11 The parameters for constructing the GB model and their values for preliminary executions.

Model parameter	Search range
Number of boosting stages	$n\_estimators = \{20, 40, 60, 80, 100\}$
Minimum number of leaf nodes	$min\_samples\_leaf = \{10, 20, 30, 40, 50, 60, 70\}$
Maximum depth	$max\_depth = \{5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16\}$

TABLE 12 The optimal values of the method parameters for the D1 dataset.

Method	Parameters settings
OCF&MLP	$L = 0.05, H = 7, M = 0.2, \alpha = 0.0001$
UCP&MLP	$L = 0.04, H = 7, M = 0.5, \alpha = 0.001$
OCF&SVR	$C = 10, \gamma = 0.001, \varepsilon = 0.001$
UCP&SVR	$C = 10, \gamma = 1, \varepsilon = 1$
OCF&DT	max_depth = 7, min_weight_fraction_leaf = 0.4, max_leaf_node = 40, min_sample_leaf = 10
UCP&DT	max_depth = 5, min_weight_fraction_leaf = 0.3, max_leaf_node = 20, min_sample_leaf = 6
OCF&RF	n_estimators = 100, min_sample_leaf = 2, max_depth = 10
UCP&RF	n_estimators = 150, min_sample_leaf = 2, max_depth = 20
OCF&GB	n_estimators = 60, min_sample_leaf = 60, max_depth = 5
OCF&KNN	neighbors = 5
UCP&KNN	neighbors = 10
UCP&GRNN	$\sigma = 0.1$

TABLE 13 The optimal values of the method parameters for the D2 dataset.

Method	Parameters settings
OCF&MLP	$L=0.02, H=8, M=0.5, \alpha=0.001$
UCP&MLP	$L=0.03, H=6, M=0.5, \alpha=0.0001$
OCF&SVR	$C=100, \gamma=0.1, \varepsilon=0.001$
UCP&SVR	$C=10, \gamma=1, \varepsilon=0.1$
OCF&DT	max_depth = 5, min_weight_fraction_leaf = 0.5, max_leaf_node = 30, min_sample_leaf = 4
UCP&DT	max_depth = 3, min_weight_fraction_leaf = 0.3, max_leaf_node = 40, min_sample_leaf = 2
OCF&RF	n_estimators = 200, min_sample_leaf = 1, max_depth = 50
UCP&RF	n_estimators = 100, min_sample_leaf = 1, max_depth = 30
OCF&GB	n_estimators = 20, min_sample_leaf = 40, max_depth = 6
OCF&KNN	neighbors = 8
UCP&KNN	neighbors = 9
UCP&GRNN	$\sigma=0.3$

TABLE 14 The optimal values of the method parameters for the D3 dataset.

Method	Parameters settings
OCF&MLP	$L=0.01, H=6, M=0.2, \alpha=0.01$
UCP&MLP	$L=0.04, H=6, M=0.2, \alpha=0.01$
OCF&SVR	$C=50, \gamma=1, \varepsilon=1$
UCP&SVR	$C=10, \gamma=0.01, \varepsilon=0.01$
OCF&DT	max_depth = 9, min_weight_fraction_leaf = 0.3, max_leaf_node = 10, min_sample_leaf = 5
UCP&DT	max_depth = 5, min_weight_fraction_leaf = 0.1, max_leaf_node = 30, min_sample_leaf = 7
OCF&RF	n_estimators = 300, min_sample_leaf = 4, max_depth = 80
UCP&RF	n_estimator = 400, min_sample_leaf = 2, max_depth = 50
OCF&GB	n_estimators = 30, min_sample_leaf = 30, max_depth = 7
OCF&KNN	neighbors = 6
UCP&KNN	neighbors = 10
UCP&GRNN	$\sigma=0.6$

1. LASSO regression is used to determine the best correction factors for the UCP method (details are presented in Section 3.2). A list of the best correction factors for each dataset is presented in Appendix B (Tables 36 and 37).
2. The input and output vectors are determined.
3. The data is divided into a training set $S^{(-j)}$ and a test set S_j. $S^{(-j)}$ is used to create the Level 0 models (regressors) via seven learning algorithms (SVM, KNN, DT, MLP, MLR, GB, and RF).
4. The configuration parameters for the seven regression models (Level 0 models) SVM, KNN, DT, MLP, MLR, GB, and RF are tuned on the validation set (30% of the training set) to produce their optimal settings (see Section 3.4).
5. Create an ensemble model with the stacking method. The estimator's predictions are stacked and fed into a final estimator, which computes the final estimation. More precisely, each of the Level 0 models in the first stage undergo five-fold cross-validation in $S^{(-j)}$ to output its prediction and generate a prediction for S_j by taking the average of the seven estimation results generated by the five CV models in the training phase. Then, these Level 0 models create a vector of predictions to input into the Level 1 model (in the second stage). RF was selected as the meta-regressor to train a new model for the final project size estimation.

TABLE 15 The optimal values of the method parameters for the D4 dataset.

Method	Parameters settings
OCF&MLP	$L = 0.02, H = 6, M = 0.3, \alpha = 0.01$
UCP&MLP	$L = 0.03, H = 6, M = 0.4, \alpha = 0.001$
OCF&SVR	$C = 100, \gamma = 1, \varepsilon = 0.01$
UCP&SVR	$C = 10, \gamma = 0.01, \varepsilon = 0.01$
OCF&DT	max_depth = 3, min_weight_fraction_leaf = 0.1, max_leaf_node = 50, min_sample_leaf = 2
UCP&DT	max_depth = 5, min_weight_fraction_leaf = 0.5, max_leaf_node = 30, min_sample_leaf = 3
OCF&RF	n_estimators = 250, min_sample_leaf = 4, max_depth = 10
UCP&RF	n_estimators = 300, min_sample_leaf = 4, max_depth = 20
OCF&GB	n_estimators = 40, min_sample_leaf = 50, max_depth = 6
OCF&KNN	neighbors = 7
UCP&KNN	neighbors = 9
UCP&GRNN	$\sigma = 0.4$

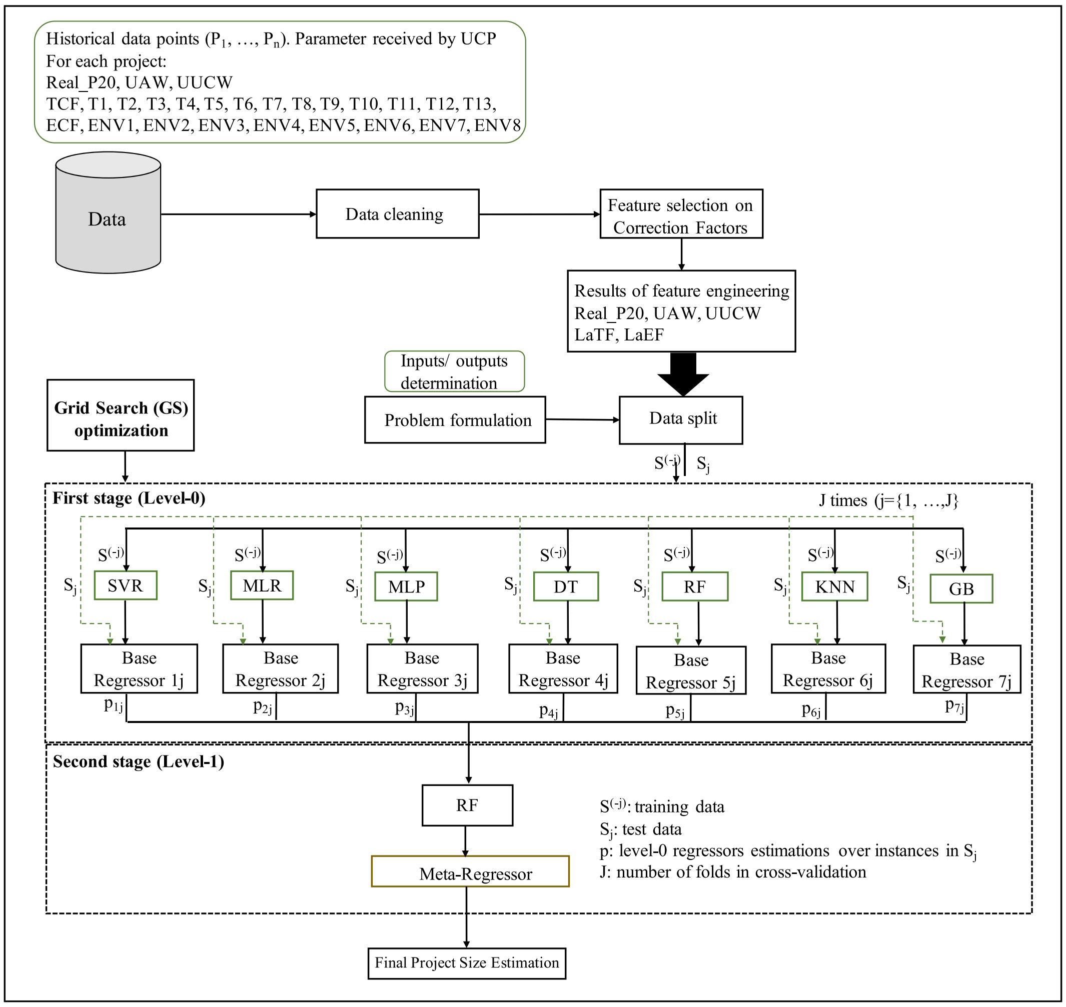
Historical data points (P1, …, Pn). Parameter received by UCP
For each project:
Real_P20, UAW, UUCW
TCF, T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12, T13,
ECF, ENV1, ENV2, ENV3, ENV4, ENV5, ENV6, ENV7, ENV8
Data
Data cleaning
Feature selection on Correction Factors
Results of feature engineering
Real_P20, UAW, UUCW
LaTF, LaEF
Inputs/ outputs determination
Grid Search (GS) optimization
Problem formulation
Data split
S(-j) Sj
First stage (Level-0)
J times (j={1, …,J}
SVR
MLR
MLP
DT
RF
KNN
GB
Base Regressor 1j
Base Regressor 2j
Base Regressor 3j
Base Regressor 4j
Base Regressor 5j
Base Regressor 6j
Base Regressor 7j
p1j p2j p3j p4j p5j p6j p7j
Second stage (Level-1)
RF
Meta-Regressor
S(-j): training data
Sj: test data
p: level-0 regressors estimations over instances in Sj
J: number of folds in cross-validation
Final Project Size Estimation

FIGURE 4 The architecture of the proposed SOCF model.

5 | EXPERIMENTAL DESIGN

This section presents the experimental design, which consists of (1) an experimental process for evaluating the SDEE methods, (2) descriptions of the datasets for the experiment, and (3) the evaluation criteria for assessing SDEE method accuracy.

5.1 | Experimental process

The experimental process for evaluating the accuracies of SDEE methods is shown in Figure 5. We performed experiments to compare our proposed SOCF method with related methods, such as the UCP-based single methods (described in Table 16), OCF-based single methods (described in Table 17), and ensemble methods (described in Table 18). In addition, we experimented with pure estimation methods, such as a baseline UCP method,[7] and an OCF method.[48]

We ran each experiment five times under five different random training and testing splits. The comparisons of the methods' estimation accuracies were based on the average results of these five runs and seven evaluation criteria, namely, the SSE, MAE, RMSE, MBRE, MIBRE, MdMRE, and PRED(0.25), which are defined in Equations (18)–(24) of Section 6.3. A statistical comparison was also used to validate method accuracy.

5.2 | Dataset descriptions

The UCP methodology is a promising tool for early effort estimation in the software industry, but it still requires refinements in certain areas as per our prior works.[48,49] The evaluation of correction factors (TCF and ECF), which inherently possess a degree of uncertainty, significantly influences the precision of the UCP method. This research examines explicitly the dataset quality, encompassing the number of projects incorporated.

Consequently, we utilized the UCP benchmark dataset.[38] This dataset's significance lies in its comprehensive coverage of technical (T1–T13) and environmental factors (E1–E8), enabling a thorough evaluation of effort estimation methodologies. While some studies utilized multiple datasets,[83,84] to the best of our understanding, no alternate dataset aligns with our focus area.

A critical challenge is the relatively small dataset size, but the implementation previously mentioned LOOCV approach (as discussed in the Threat to validity section) helps overcome this challenge. Other datasets that are publicly accessible for experiment replication and result generalization, unlike in other research domains of effort estimation, cannot be used in the study, which evaluates TCF and ECF. This accessibility issue significantly hinders broader community participation and progress.

A total of 70 projects from three repositories were used. Figure 6 shows boxplots of Real_P20 for each data repository, where Real_P20 is a real effort in person-hours divided by productivity (PF – person-hours per 1 UCP). The repositories have significantly different Real_P20 values. Specifically, the D1 data repository has the largest Real_P20 for projects, whereas the D3 data repository has the smallest Real_P20 for projects. The D4 data repository, which combines D1–D3, was used to evaluate the impact of mixing projects from different data repositories.

Table 19 summarizes the descriptive statistics for the dataset's Real_P20 variables, including dataset size, as well as the median, mean, minimum, and maximum Real_P20 values. For all datasets, the median Real_P20 uses PF = 20 because it is assumed that 20 person-hours equals 1 UCP.[7] Minimum Real_P20 and maximum Real_P20 describe the smallest and largest project sizes in each case.

5.3 | Evaluation criteria

In SDEE, different criteria are used to evaluate the accuracy of the estimation methods. The accuracy of the SDEE method in terms of MMRE and MMER[1,9,49] are the most commonly used measures. However, these measurements can be biased.[8,26,85,86] Therefore, to evaluate the proposed estimation method, in this study, alternative criteria are used that provide an unbiased symmetric distribution as follows: SSE (Equation [18]), MAE (Equation [19]), RMSE (Equation [20]), MBRE (Equation [21]), MIBRE (Equation [22]), MdMRE (Equation [23]), and PRED(x) (where $x = 0.25$ in this study; Equation [24]). All of these criteria have been proven to be effective.[53]

Specifically, SSE and PRED(0.25) are used to evaluate the accuracy of the estimated model. SSE is the most important metric to assess the variability of modeling errors.[87] This metric can describe errors in selected datasets. PRED(0.25) is less biased towards underestimation. This usually identifies the best method as standardized accuracy. The SDEE method with high estimation accuracy (when the value of PRED(x) is high) is also suitable (when the value of SA is high).[53]

$$SSE = \sum_{i=1}^{n} (y_i - \widehat{y}_i)^2, \tag{18}$$

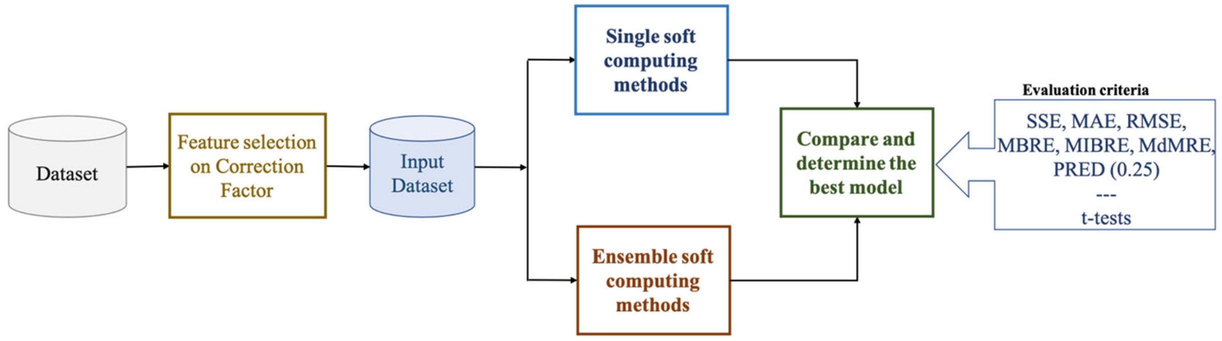
Dataset
Feature selection on Correction Factor
Input Dataset
Single soft computing methods
Ensemble soft computing methods
Compare and determine the best model
Evaluation criteria
SSE, MAE, RMSE, MBRE, MIBRE, MdMRE, PRED (0.25)
---
t-tests

FIGURE 5 Description of experimental process.

TABLE 16 UCP-based single methods implemented for experiments.

No.	Effort estimation method	ML technique	Summary	Notation
1	Use case point	MLR	• Uses MLR to estimate the software size based on UCP variables (UAW, UUCW, TCF, and ECF).	UCP&MLR
2	Use case point	SVR	• Uses SVR to estimate the software size based on UCP variables (UAW, UUCW, TCF, and ECF).	UCP&SVR
3	Use case point	KNN	• Uses KNN to estimate the software size based on UCP variables (UAW, UUCW, TCF, and ECF).	UCP&KNN
4	Use case point	DT	• Uses DT to estimate the software size based on UCP variables (UAW, UUCW, TCF, and ECF).	UCP&DT
5	Use case point	GRNN	• Uses GRNN to estimate the software size based on UCP variables (UAW, UUCW, TCF, and ECF).	UCP&GRNN
6	Use case point	MLP	• Uses MLP to estimate the software size based on UCP variables (UAW, UUCW, TCF, and ECF).	UCP&MLP
7	Use case point	RF	• Uses RF to estimate the software size based on UCP variables (UAW, UUCW, TCF, and ECF).	UCP&RF

TABLE 17 OCF-based single methods implemented for experiments.

No.	Effort estimation method	Statistical and ML technique	Summary	Notation
1	Optimization correction factor	SVR	• Uses SVR to estimate the software size based on OCF variables (UAW, UUCW, LaTF, and LaEF).	OCF&SVR
2	Optimization correction factor	MLP	• Uses MLP to estimate the software size based on OCF variables (UAW, UUCW, LaTF, and LaEF).	OCF&MLP
3	Optimization correction factor	GB	• Uses GB to estimate the software size based on OCF variables (UAW, UUCW, LaTF, and LaEF).	OCF&GB
4	Optimization correction factor	MLR	• Uses RF to estimate the software size based on OCF variables (UAW, UUCW, LaTF, and LaEF).	OCF&MLR
5	Optimization correction factor	KNN	• Uses KNN to estimate the software size based on OCF variables (UAW, UUCW, LaTF, and LaEF).	OCF&KNN
6	Optimization correction factor	DT	• Uses DT to estimate the software size based on OCF variables (UAW, UUCW, LaTF, and LaEF).	OCF&DT
7	Optimization correction factor	RF	• Uses RF to estimate the software size based on OCF variables (UAW, UUCW, LaTF, and LaEF).	OCF&RF

TABLE 18 Ensemble methods implemented for experiments.

No.	Effort estimation method	Statistical and ML technique	Summary	Notation
1	Use case point	Majority voting ensemble[80]	• Uses an ensemble of the MLR, SVR, and MLP models with the majority voting method to estimate the software size based on UCP variables (UAW, UUCW, TCF, and ECF).	VUCP[81]
2	Optimization correction factor	Stacked generalization ensemble[82]	• Uses an ensemble of the SVM, KNN, DT, MLP, MLR, GB, and RF models with the stacked generalization method to estimate the software size based on OCF variables (UAW, UUCW, LaTF, and LaEF).	SOCF (proposed in Section 4)

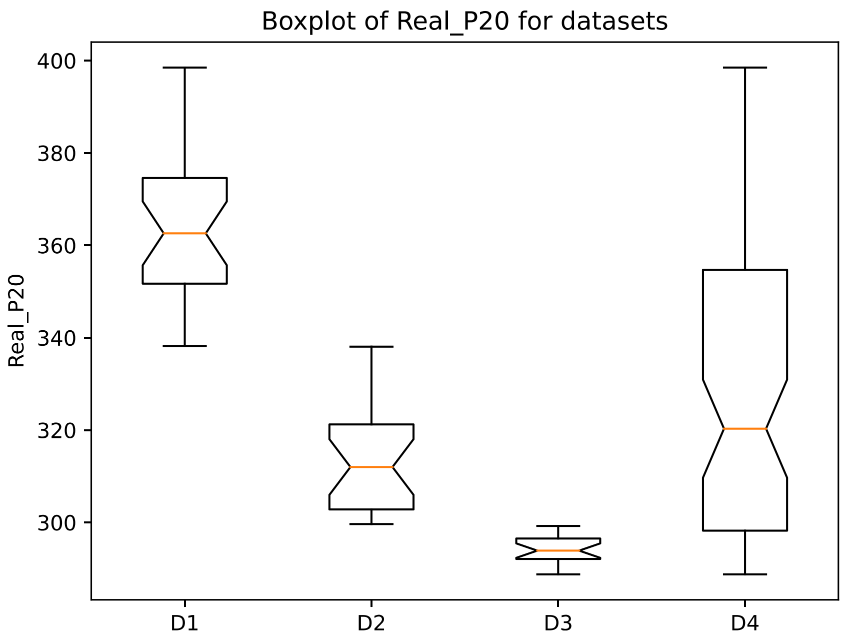

FIGURE 6 Boxplots of Real_P20 for four datasets.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|, \tag{19}$$

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}}, \tag{20}$$

$$MBRE = \frac{1}{n}\sum_{i=1}^{n}\frac{|(y_i - \hat{y}_i)|}{min(y_i - \hat{y}_i)}, \tag{21}$$

$$MIBRE = \frac{1}{n}\sum_{i=1}^{n}\frac{|(y_i - \hat{y}_i)|}{max(y_i - \hat{y}_i)}, \tag{22}$$

$$MdMRE = median_i\left(\frac{|y_i - \hat{y}_i|}{y_i}\right), \tag{23}$$

$$PRED(x) = \frac{1}{n}\sum_{i=1}^{n}\begin{cases}1 \text{ if } \frac{|y_i - \hat{y}_i|}{y_i} \le x\\ 0 \text{ otherwise}\end{cases}, \tag{24}$$

where n is the number of observations, y_i is the known real value, and $\hat{y}_i$ is the estimated value.

TABLE 19 Descriptive statistics for the datasets.

		Real_P20				
Dataset	**Size**	**Min**	**Max**	**Mean**	**Median**	**Standard deviation**
D1	27	338.200	398.500	364.500	362.600	18.820
D2	23	299.650	338.050	314.708	312.000	12.156
D3	20	288.750	299.250	293.787	293.900	3.287
D4	70	288.750	398.500	327.936	320.300	33.212

6 | RESULTS AND DISCUSSION

This section presents the results obtained for our proposed effort estimation method as well as the related methods and answers our research questions.

6.1 | RQ1

To answer this question, we will first evaluate and rank the six UCP-based and seven OCF-based SDEE methods for each dataset. Second, we will consider the ensemble methods and compare them with their component methods for each dataset. Finally, the ensemble methods will be compared with each other.

The first step in assessing these statistical and ML techniques consisted of building and tuning them using the GS optimization technique. The optimal settings for the datasets are listed in Section 3.4. Tables 20 and 21 present the estimation accuracies of the six UCP-based single methods and seven OCF-based single methods across the four datasets.

The first observation from these results is that the OCF-based estimation methods, that is, OCF&SVR, OCF&MLP, OCF&DT, OCF&KNN, and OCF&RF, minimize the errors more effectively than the traditional UCP model-based estimation methods, that is, UCP&SVR, UCP&MLP, UCP&DT, UCP&KNN, and UCP&RF. This further reinforces the effectiveness of OCF variables when they are leveraged in estimation methods. In addition, the determination of the technical and environmental complexity factors helped our OCF-based methods to give better experimental in terms of average SSE, MAE, RMSE, MdMRE, MBRE, and MIBRE results in all experimental datasets, as shown in Figure 7. Based on the SSE, MAE, RMSE, MdMRE, MBRE, and MIBRE results in Tables 20 and 21, we present in Tables 22, 23, 24, and 25 the percentage improvements in SSE, MAE, RMSE, MdMRE, MBRE, and MIBRE results of the OCF-based estimation methods over the UCP-based estimation methods. Following are some comments on the most significant improvements between the SSE results of the OCF-based and UCP-based models: the SSE results of OCF&KNN are 133.39% and 166.71% better than those of UCP&KNN in data sets D1 and D2, respectively. For datasets D3 and D4, the SSE results of OCF&RF are better than those of UCP&RF by 36.93% and 116%, respectively. Based on this finding, we can conclude that approaches that use OCF variables outperform those that use UCP variables.

The second observation from these results is that the estimation accuracies of the single methods vary from one dataset to another, making them unstable across these datasets and the evaluation criteria. In particular, the best model for the D1 dataset among the UCP-based single models was UCP&GRNN. UCP&KNN had the lowest accuracy according to the SSE, whereas UCP&SVR had the most insufficient accuracy according to the MAE, RMSE, MBRE, MIBRE, and MdMRE. For the D2 dataset, UCP&GRNN had the highest accuracy, whereas UCP&MLP had the lowest. For the D3 dataset, UCP&SVR achieved the best accuracy according to the SSE, MAE, RMSE, MBRE, and MIBRE, whereas UCP&DT achieved the best accuracy according to the MdMRE. UCP&RF was the worst model. For the D4 dataset, UCP&DT achieved the best accuracy according to the SSE, whereas UCP&SVR had the lowest accuracy according to the MAE, RMSE, MBRE, MIBRE, and MdMRE. UCP&RF was the worst model according to the SSE, RMSE, MBRE, MIBRE, and MdMRE, whereas UCP&MLP was the worst model according to the MAE. Similarly, among the OCF-based single models for the D1 dataset, OCF&RF performed best according to the SSE and RMSE, whereas OCF&KNN performed best according to the MAE, MBRE, MIBRE, and MdMRE. OCF&SVR was the worst model. For the D2 dataset, OCF&KNN had the highest accuracy, whereas UCP&MLP had the lowest. For the D3 dataset, OCF&SVR had the best accuracy according to the SSE, MAE, RMSE, MBRE, and MIBRE, whereas UCP&GB achieved the best accuracy according to the MdMRE. OCF&MLP was the worst model. For the D4 dataset, OCF&DT had the best accuracy according to the SSE, whereas OCF&SVR had the lowest accuracy according to the MAE, RMSE, MBRE, MIBRE, and MdMRE. OCF&MLP was the worst model. Table 26 ranks the UCP-based single methods from 1 to 6 based on the SSE metric across the datasets, with "1" being the best, and "6" being the worst method in terms of the SSE metric. Similarly, in Table 27, the OCF-based methods are ranked from 1 to 7 based on the SSE metric across the datasets, where "1" represents the best method, and "7" represents the worst method based on the SSE metric. From these results, we can conclude that there is no single absolutely best method, meaning a single model can provide superior estimation accuracy for one dataset while doing poorly on another dataset.

TABLE 20 Estimation results for the UCP-based single methods. The best results are in bold. The worst results are italicized.

Method	SSE	MAE	RMSE	MBRE	MIBRE	MdMRE	PRED
	D1 dataset						
UCP&SVR	1866.171	*16.732*	*18.711*	*0.048*	*0.045*	*0.045*	1.00
UCP&MLP	1515.529	14.082	16.768	0.040	0.038	0.036	1.00
UCP&GRNN	**1493.428**	**12.770**	**15.553**	**0.039**	**0.036**	**0.035**	1.00
UCP&KNN	*1942.105*	16.564	18.532	0.047	0.044	0.048	1.00
UCP&DT	1520.841	13.539	16.619	0.038	0.036	0.030	1.00
UCP&RF	1526.650	14.048	16.440	0.040	0.037	0.029	1.00
	D2 dataset						
UCP&SVR	768.535	10.180	13.168	0.034	0.032	0.026	1.00
UCP&MLP	*1546.821*	*14.854*	*17.333*	*0.050*	*0.046*	*0.040*	1.00
UCP&GRNN	**392.452**	**8.382**	**9.736**	**0.028**	**0.026**	**0.023**	1.00
UCP&KNN	651.119	11.122	12.459	0.036	0.035	0.032	1.00
UCP&DT	528.280	9.497	11.151	0.031	0.030	0.027	1.00
UCP&RF	405.550	8.054	9.640	0.026	0.025	0.021	1.00
	D3 dataset						
UCP&SVR	**41.978**	**3.066**	**3.573**	**0.011**	**0.010**	0.014	1.00
UCP&MLP	56.090	3.629	4.050	0.012	0.012	0.015	1.00
UCP&GRNN	51.268	3.517	4.020	0.012	0.012	0.015	1.00
UCP&KNN	54.621	3.780	4.210	0.013	0.013	0.016	1.00
UCP&DT	46.617	3.305	3.767	0.011	0.011	**0.013**	1.00
UCP&RF	*60.420*	*3.941*	*4.407*	*0.014*	*0.013*	*0.017*	1.00
	D4 dataset						
UCP&SVR	10,935.116	**25.628**	**30.962**	**0.082**	**0.074**	**0.073**	1.00
UCP&MLP	11,890.211	*25.894*	31.395	0.081	0.072	0.062	0.98
UCP&GRNN	11,105.597	23.822	29.951	0.076	0.067	0.056	1.00
UCP&KNN	11,074.020	24.558	30.942	0.077	0.068	0.060	0.98
UCP&DT	**10,878.228**	26.588	31.223	0.086	0.077	0.085	1.00
UCP&RF	*13,470.085*	25.689	*32.905*	*0.083*	*0.072*	*0.073*	0.98

The third observation from these results is that the ensemble methods outperform all their components. We compare the experimental results for the two ensemble methods (VUCP and SOCF) with their component methods across the datasets (Tables 20–21 show the estimation accuracies for the single methods, which can then be compared with the results for their respective ensemble methods in Table 28). Specifically, the SOCF ensemble method leads to the average SSE result better than OCF&SVR, OCF&MLP, OCF&DT, OCF&MLR, OCF&GB, OCF&RF, and OCF&KNN, respectively, at 79.42%, 87.19%, 148.69%, 73.86%, 152.09%, 51.68%, and 56.01%. The results also show that the VUCP ensemble method produces the average SSE results better than UCP&SVR, UCP&KNN, and UCP&DT with 41.92%, 43.07%, and 35.27%, respectively. The comparison between the ensemble methods and their single approaches is shown in Figures 8 and 9.

In this straight line, we delved into the obtained experimental results to investigate the distinctiveness of our proposed approach. We have proposed a new EEE-based OCF approach by combining the results of seven commonly used statistical and ML techniques with the OCF method. The techniques are MLR, KNN, SVR, MLP, RF, GB, and DT. In this work, one of the strengths of our proposed SOCF is that we find the correct or optimal hyperparameter values, which is the optimal model and uncertain training computational cost and test estimation models with different values of hyperparameters and propose hyperparameter optimization with GS to optimize the parameters of the seven models in SOCF and find the best parameters to estimate the effort of four datasets. We compared the experimental results with the ensemble algorithms commonly used in the literature (AdaBoost ensemble, AUCP; random forest ensemble, ROCF; and voting ensemble, VUCP) and other related methods (see details in Table 28). Figure 10 shows the improvement where SOCF outperforms all other methods regarding SSE, MAE, RMSE, MdMRE, MBRE, and MIBRE. It can be seen that SOCF produced better SSE, MAE, MdMRE, MBRE, MIBRE, and RMSE results than VUCP by 1.969, 1.561, 1.791, 1.621, 1.217, and 1.448 times, respectively. Compared with AUCP, SOCF results were better than 2.914, 2.113, 2.344, 2.202, 1.650, and 1.885 times, respectively. Similarly, SOCF results were better than ROCF results of 1.517, 1.367, 1.529, 1.405, 1.067, and 1.323, respectively. Generally, SOCF also provides better SSE, MAE, RMSE, MdMRE, MBRE, and MIBRE results than the remaining methods.

TABLE 21 The estimation results for the OCF-based single methods.

Method	SSE	MAE	RMSE	MBRE	MIBRE	MdMRE	PRED
	D1 dataset						
OCF&SVR	*1410.337*	*13.900*	*16.350*	*0.039*	*0.037*	*0.029*	1.00
OCF&MLP	1197.735	12.887	15.362	0.036	0.035	0.031	1.00
OCF&DT	1343.018	13.470	16.021	0.038	0.036	0.030	1.00
OCF&MLR	1018.045	11.545	13.719	0.032	0.031	0.025	1.00
OCF&GB	1314.082	13.434	15.903	0.038	0.036	0.030	1.00
OCF&RF	**747.095**	9.520	**11.700**	0.027	0.026	0.022	1.00
OCF&KNN	832.111	**9.245**	12.356	**0.026**	**0.024**	**0.017**	1.00
	D2 dataset						
OCF&SVR	649.039	9.434	12.399	0.032	0.030	0.025	1.00
OCF&MLP	*994.084*	*12.096*	*15.033*	*0.040*	*0.038*	*0.033*	1.00
OCF&DT	278.476	7.203	7.949	0.023	0.023	0.022	1.00
OCF&MLR	493.827	9.479	11.017	0.031	0.029	0.026	1.00
OCF&GB	279.682	7.203	7.972	0.023	0.023	0.022	1.00
OCF&RF	360.796	7.371	9.340	0.024	0.023	0.019	1.00
OCF&KNN	**244.127**	**6.405**	**7.673**	**0.020**	**0.022**	**0.018**	1.00
	D3 dataset						
OCF&SVR	**36.965**	**2.891**	**3.387**	**0.010**	**0.010**	0.013	1.00
OCF&MLP	*51.081*	*3.634*	*3.968*	*0.012*	*0.012*	*0.014*	1.00
OCF&DT	37.345	2.887	3.408	0.010	0.010	0.013	1.00
OCF&MLR	46.500	3.417	3.806	0.012	0.012	0.013	1.00
OCF&GB	37.893	2.899	3.437	0.010	0.010	**0.012**	1.00
OCF&RF	44.123	3.132	3.700	0.011	0.011	0.013	1.00
OCF&KNN	47.462	3.312	3.830	0.011	0.011	0.014	1.00
	D4 dataset						
OCF&SVR	6642.853	18.577	23.772	0.059	0.054	0.047	1.00
OCF&MLP	6874.876	18.429	24.192	0.058	0.053	0.047	1.00
OCF&DT	10,454.394	25.932	30.547	0.083	0.075	0.077	1.00
OCF&MLR	6909.895	19.053	24.246	0.060	0.054	0.054	1.00
OCF&GB	*10,647.201*	*26.230*	*30.810*	*0.085*	*0.076*	*0.084*	1.00
OCF&RF	**6236.123**	**17.973**	**23.387**	**0.056**	**0.051**	**0.050**	1.00
OCF&KNN	6475.196	18.208	24.113	0.057	0.052	0.044	1.00

Note: The best results are in bold. The worst results are italicized.

Table 29 shows the processing time (in seconds) of the different experimental methods. It can be seen that the methods using neural network techniques, that is, SOCF, UCP&MLP, and OCF&MLP, have longer training time than other conventional models. In particular, their average training time is longer than that of UCP&RF, OCF&RF, UCP&DT, VUCP, and OCF&DT: 47.61, 42.77, 36.24, 36.01, and 30.98 times in the D1 dataset; 10.40, 10.07, 15.02, 14.97, and 13.09 times in the D2 dataset; 25.61, 15.76, 35.05, 34.48, and 26.91 times in the D3 dataset; and 54.59, 52.19, 70.79, 69.82, and 50.75 times in the D4 dataset. The significantly higher time consumption of these methods is explained as follows by GS performing in the step of tuning the hyperparameters. The main drawback of GS is its ineffectiveness in the configuration space of high-dimensional hyperparameters since the number of evaluations increases exponentially with the frequency of hyperparameters. Assuming that k parameters exist and each has n distinct values, the computational complexity increases exponentially at a rate of $O(n^k)$.[57] Is it, therefore, necessary to perform hyperparameter tuning in ML methods? How about using these methods with the default parameters of the models, referred to as SOCFwithoutGS? Figure 10 sheds light on these two questions in terms of SSE, MAE, MBRE, MIBRE, MdMRE, and RMSE results. We experimented with the default parameters of the models. Specifically, compared to the other methods, we tested the estimation performance of all seven models in SOCF without applying the grid search hyperparameter optimization. It can be seen that the ratio of improvement where SOCFwithoutGS outperforms the other methods is not better than SOCF. We found that most hyperparameter values are changed during tuning,

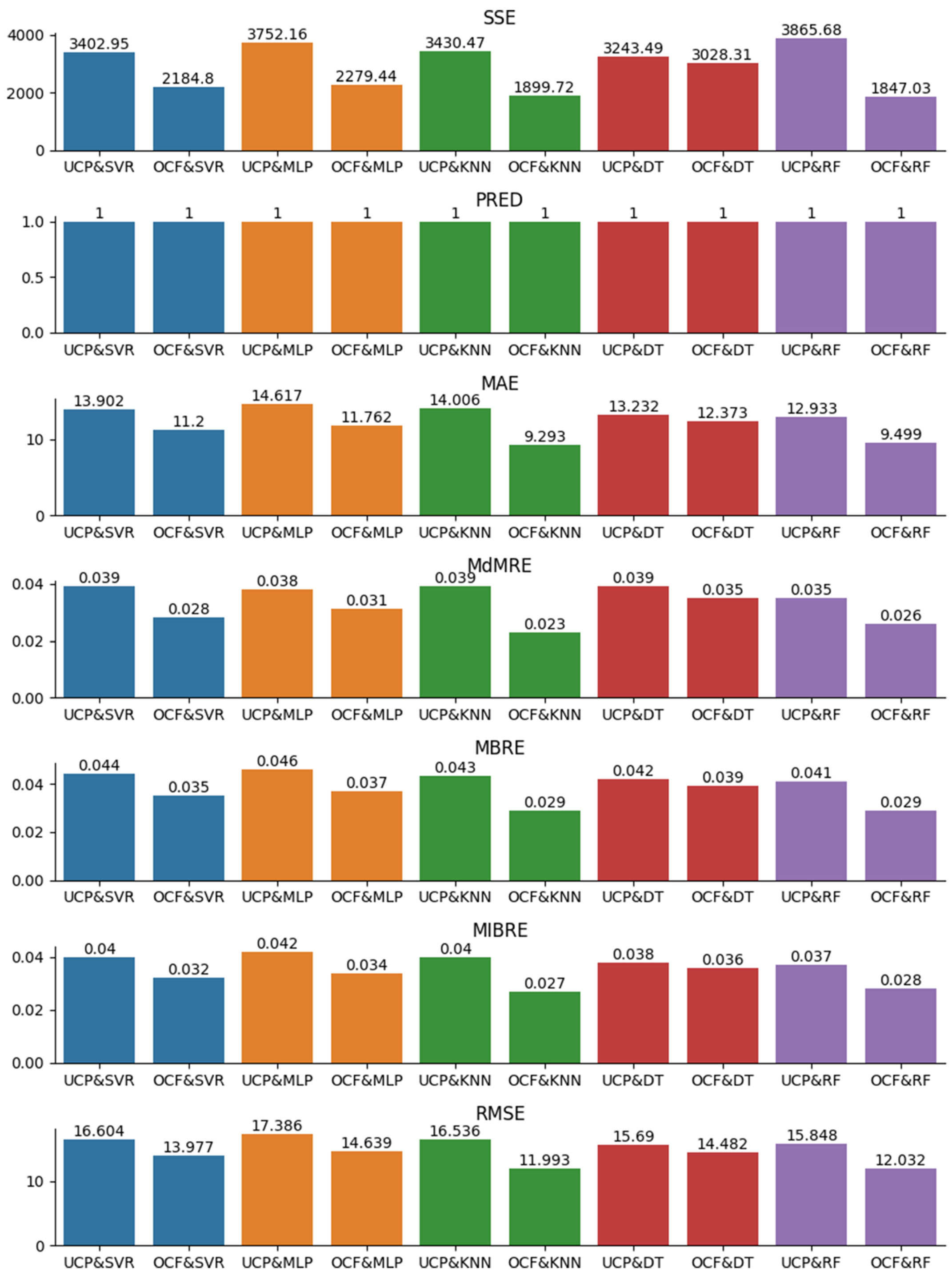

FIGURE 7 The average estimation results of the UCP-based and OCF-based single methods on all datasets.

indicating that the default values are suboptimal. The SSE result of our proposed SOCF method was more than 16.022%, 16.032%, 11.765%, 19.048%, and 13.358% of the SSE, MAE, MdMRE, MBRE, and RMSE of SOCFwithoutGS, respectively. These results showed that the tuning process of the model's hyperparameters has a statistically significant positive impact on the estimation accuracy of the models. The methods in this study performed well with optimally configured hyperparameter values. Moreover, these results show that when applying statistical and ML methods, the optimization of the hyperparameters must be considered in the estimation process, as this theoretically increases the prediction efficiency of ML methods. Based on the experimental results, conclusions can be drawn that the SOCF is a comprehensive approach to complex algorithms based on the exploration of technical requirements for more accurate software effort estimation.

TABLE 22 The percentage improvements of OCF&SVR, OCF&MLP, OCF&KNN, OCF&DT, and OCF&RF over UCP&SVR, UCP &MLP, UCP &KNN, UCP &DT, and UCP &RF on the D1 dataset.

	OCF&SVR vs. UCP&SVR	OCF&MLP vs. UCP&MLP	OCF&KNN vs. UCP&KNN	OCF&DT vs. UCP&DT	OCF&RF vs. UCP&RF
SSE	32.32%	26.53%	133.39%	13.24%	104.34%
MAE	20.38%	9.28%	79.16%	0.51%	47.56%
RMSE	14.44%	9.15%	49.99%	3.73%	40.51%
MdMRE	57.64%	16.03%	186.90%	0.00%	28.83%
MBRE	21.43%	9.39%	82.17%	1.59%	48.87%
MIBRE	21.08%	8.67%	82.64%	0.00%	46.09%

TABLE 23 The percentage improvements of OCF&SVR, OCF&MLP, OCF&KNN, OCF&DT, and OCF&RF over UCP&SVR, UCP &MLP, UCP &KNN, UCP &DT, and UCP &RF on the D2 dataset.

	OCF&SVR vs. UCP&SVR	OCF&MLP vs. UCP&MLP	OCF&KNN vs. UCP&KNN	OCF&DT vs. UCP&DT	OCF&RF vs. UCP&RF
SSE	18.41%	55.60%	166.71%	89.70%	12.40%
MAE	7.91%	22.80%	73.64%	31.85%	9.27%
RMSE	6.20%	15.30%	62.37%	40.28%	3.21%
MdMRE	4.0%	21.47%	73.91%	21.82%	15.05%
MBRE	8.23%	24.75%	77.45%	33.91%	9.09%
MIBRE	7.43%	21.69%	60.19%	29.82%	8.62%

TABLE 24 The percentage improvements of OCF&SVR, OCF&MLP, OCF&KNN, OCF&DT, and OCF&RF over UCP&SVR, UCP &MLP, UCP &KNN, UCP &DT, and UCP &RF on the D3 dataset.

	OCF&SVR vs. UCP&SVR	OCF&MLP vs. UCP&MLP	OCF&KNN vs. UCP&KNN	OCF&DT vs. UCP&DT	OCF&RF vs. UCP&RF
SSE	13.56%	9.81%	15.08%	24.83%	36.93%
MAE	6.07%	0.04%	14.13%	14.49%	25.86%
RMSE	5.49%	2.08%	9.86%	10.52%	19.10%
MdMRE	7.94%	5.80%	16.07%	3.08%	31.25%
MBRE	12.50%	0.00%	14.29%	12.00%	25.93%
MIBRE	6.25%	0.00%	9.91%	12.00%	24.53%

TABLE 25 The percentage improvements of OCF&SVR, OCF&MLP, OCF&KNN, OCF&DT, and OCF&RF over UCP&SVR, UCP &MLP, UCP &KNN, UCP &DT, and UCP &RF on the D4 dataset.

	OCF&SVR vs. UCP&SVR	OCF&MLP vs. UCP&MLP	OCF&KNN vs. UCP&KNN	OCF&DT vs. UCP&DT	OCF&RF vs. UCP&RF
SSE	64.61%	72.95%	71.02%	4.05%	116.00%
MAE	37.96%	40.51%	34.87%	2.53%	34.87%
RMSE	30.25%	29.77%	28.32%	2.21%	28.32%
MdMRE	55.32%	31.91%	36.36%	10.39%	46.00%
MBRE	38.51%	39.66%	35.09%	3.61%	48.21%
MIBRE	37.04%	35.85%	30.77%	2.67%	41.18%

6.2 | RQ2

To answer RQ2, we statistically compared the methods with a significance level of 0.05. This study used the *t*-test (parametric statistical comparison) and the Mann–Whitney U test (non-parametric statistical comparison). The *t*-test depends on the *t*-values, whereas the Mann–Whitney U test depends on the *z*-values. The *t*- and *z*-values were used to calculate the *p*-values. If the *p*-value is less than 0.05, then the two methods used in the statistical comparison are significantly different. The results of the comparison are shown in Tables 29, 30, 31, and 32. The tables show the statistical significance between our proposed and other methods. Specifically, our proposed SOCF method is statistically superior to the baseline

TABLE 26 Rankings of the UCP-based single methods based on the SSE metric.

Methods	D1	D2	D3	D4
UCP&SVR	5	5	1	2
UCP&MLP	2	6	5	5
UCP&GRNN	1	1	3	4
UCP&KNN	6	4	4	3
UCP&DT	3	3	2	1
UCP&RF	4	2	6	6

TABLE 27 Rankings of the OCF-based single methods based on the SSE metric.

Methods	D1	D2	D3	D4
OCF&SVR	7	6	1	3
OCF&MLP	4	7	7	4
OCF&DT	6	2	2	6
OCF&MLR	3	5	5	5
OCF&GB	5	3	3	7
OCF&RF	1	4	4	1
OCF&KNN	2	1	6	2

TABLE 28 Ensemble estimation methods.

Method	Base regressor	SSE	MAE	RMSE	MBRE	MIBRE	MdMRE	PRED
		D1 dataset						
VUCP	KNN, SVR, DT	1173.227	11.575	13.970	0.033	0.031	0.027	1.00
AUCP	AdaBoost	1803.058	17.012	18.614	0.048	0.046	0.046	1.00
SOCF	SVR, MLP, DT, MLR, GB, RF, KNN	**491.627**	**7.168**	**9.186**	**0.020**	**0.023**	**0.016**	**1.00**
ROCF	RF	747.095	9.520	11.700	0.027	0.026	0.022	1.00
		D2 dataset						
VUCP	KNN, SVR, DT	335.898	7.618	8.937	0.025	0.024	0.023	1.00
AUCP	AdaBoost	705.518	10.633	13.077	0.035	0.033	0.026	1.00
SOCF	SVR, MLP, DT, MLR, GB, RF, and KNN	**125.236**	**4.322**	**5.386**	**0.014**	**0.022**	**0.013**	**1.00**
ROCF	RF	360.796	7.371	9.340	0.024	0.023	0.019	1.00
		D3 dataset						
VUCP	KNN, SVR, and DT	38.537	2.865	3.431	0.010	0.010	0.013	1.00
AUCP	AdaBoost	84.212	4.542	4.876	0.015	0.015	0.013	
SOCF	SVR, MLP, DT, MLR, GB, RF, and KNN	**31.496**	**2.486**	**3.106**	**0.009**	**0.010**	**0.008**	**1.00**
ROCF	RF	44.123	3.312	3.700	0.011	0.011	0.013	**1.00**
		D4 dataset						
VUCP	KNN, SVR, and DT	8043.400	21.536	26.338	0.069	0.062	0.059	1.00
AUCP	AdaBoost	11,598.995	26.816	32.008	0.086	0.077	0.074	
SOCF	SVR, MLP, DT, MLR, GB, RF, and KNN	**4222.464**	**13.944**	**21.706**	**0.043**	**0.049**	**0.030**	**1.00**
ROCF	RF	6236.123	17.973	23.387	0.056	0.051	0.050	1.00

Note: The methods with the best results are in bold.

UCP method and its component methods (OCF&SVR, OCF&MLP, OCF&DT, OCF&MLR, OCF&GB, OCF&RF, and OCF&KNN), the VUCP ensemble method and its component methods (UCP&SVR, UCP&KNN, and UCP&DT), and the other methods tested (OCF, UCP&MLP, and UCP&GRNN) (Table 33). We can conclude that the proposed SOCF model achieves the best results regarding the number of wins compared to other models. This is due to the effectiveness of each of the three main components of the SOCF model, including optimizing model parameters

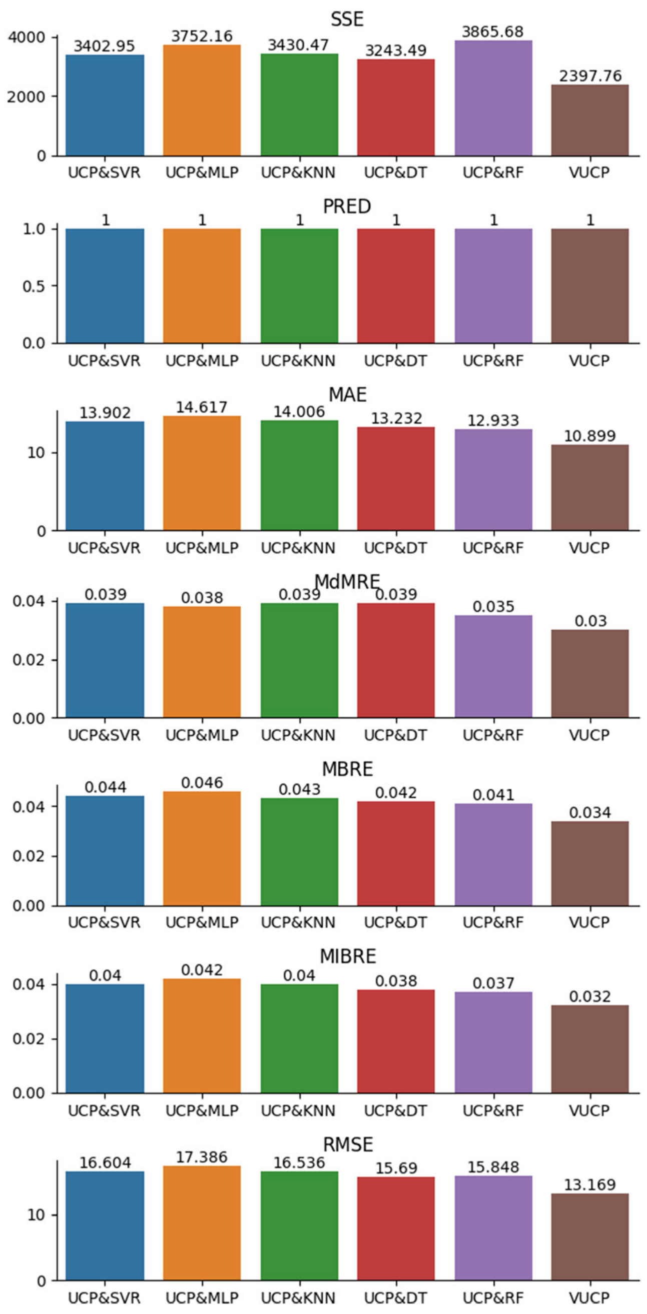

FIGURE 8 The comparison between the ensemble method VUCP and its single approaches.

using GS methods, reducing generalization errors using stacked ensembles, and selecting seven appropriate individual models for stacked ensembles. All three components strongly support our conclusion above. Therefore, we accept the alternative hypothesis H_1.

6.3 | RQ3

To address RQ3, we conducted effect analyses to assess the effectiveness of each of SOCF's three core components: (1) optimizing model parameters using the GS technique, (2) reducing the generalization error using the stacking ensemble, and (3) the selection of seven individual models for the stacking ensemble.

- Case 1: Removing the first component (optimizing model parameters using the GS technique) and substituting the default parameters for SOCF's single models. We named this method SOCF-Case1.

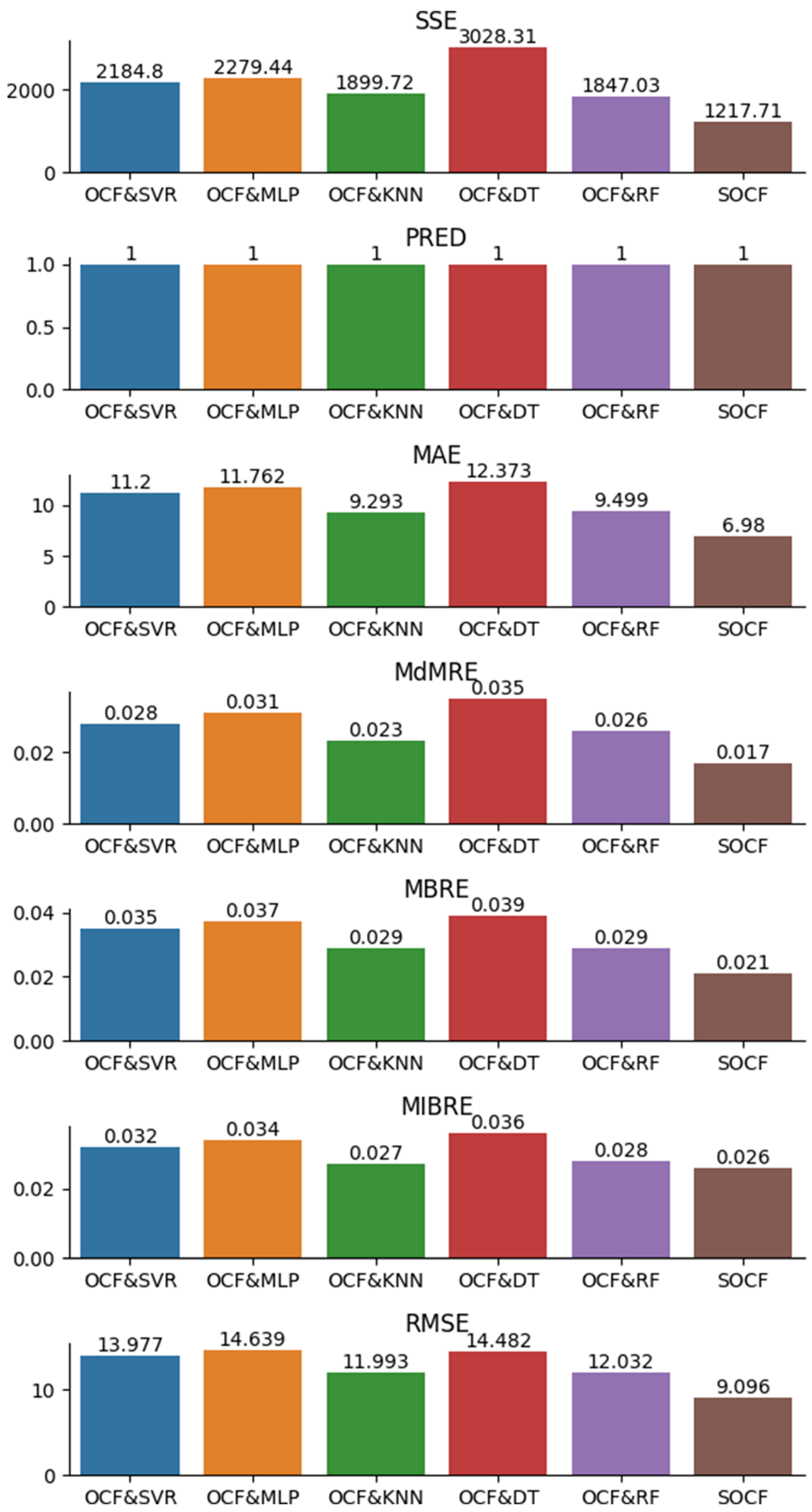

FIGURE 9 The comparison between the ensemble SOCF and its single approaches.

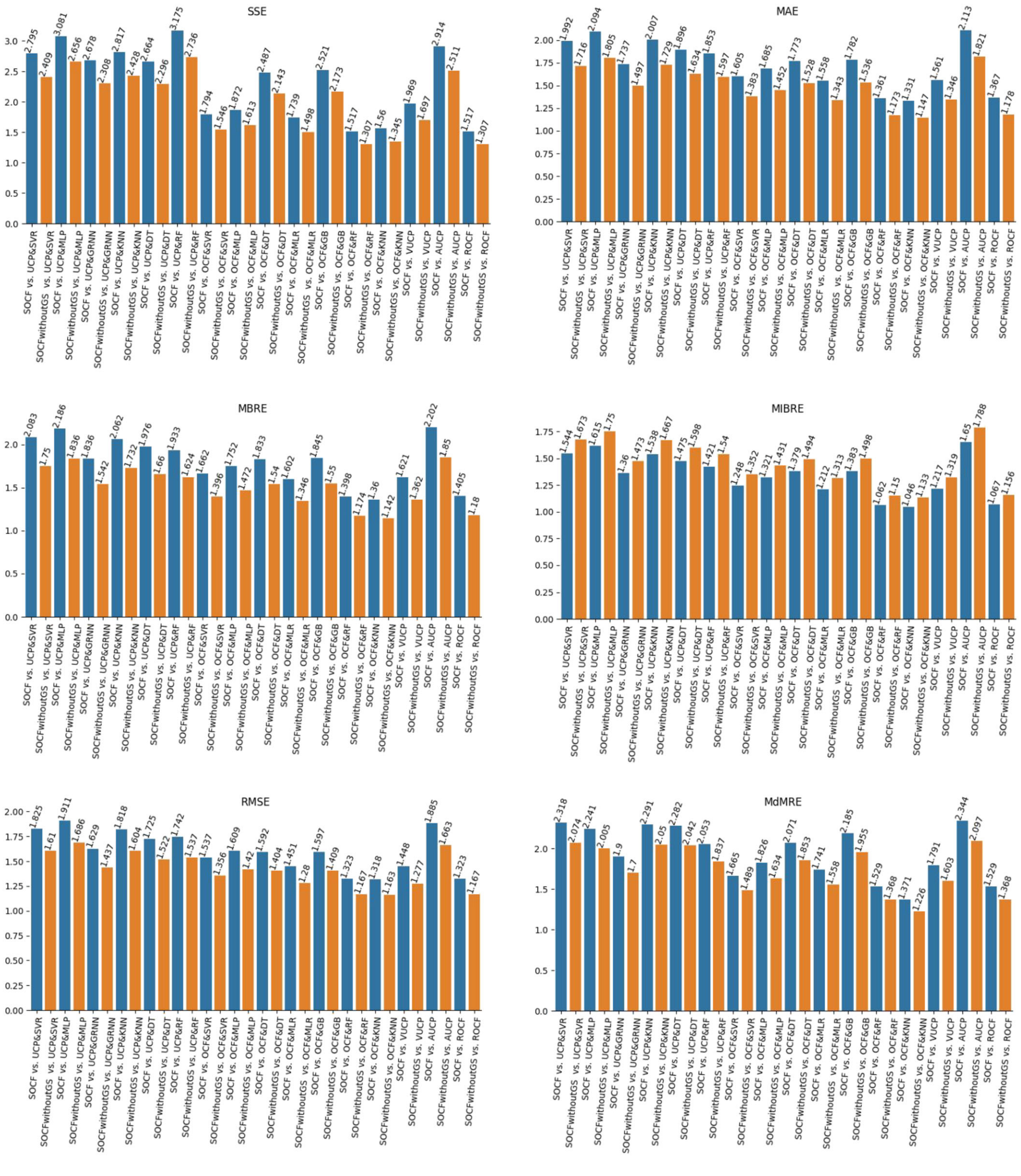

FIGURE 10 The ratio of improvement for which SOCF outperforms each other methods in terms of SSE, MAE, MBRE, MIBRE, MdMRE, and RMSE.

- Case 2: Removing the second component (reducing generalization error using the stacking ensemble) and substituting in the voting ensemble. We named this method SOCF-Case2.
- Case 3: Removing the third component (the selection of seven single models) and substituting in the three single models (MLR, SVR, and MLP). We named this method SOCF-Case3.

TABLE 29 The processing time (in seconds) for different experimental methods.

	The training time				The average response time of the estimation for a data record			
Methods	**D1**	**D2**	**D3**	**D4**	**D1**	**D2**	**D3**	**D4**
UCP&KNN	1.321	0.776	1.437	0.981	0.004	0.003	0.003	0.001
OCF&KNN	1.360	1.944	2.951	1.737	0.035	0.037	0.018	0.008
UCP&SVR	14.620	11.588	65.354	26.414	0.002	0.002	0.002	0.001
OCF&SVR	19.248	14.549	86.044	29.970	0.036	0.036	0.016	0.008
OCF&GB	148.807	199.070	292.73	94.551	0.035	0.035	0.015	0.008
UCP&RF	1874.78	5620.99	5599.09	2539.39	0.003	0.005	0.004	0.001
OCF&RF	2087.17	5807.34	9099.68	2655.77	0.039	0.039	0.018	0.009
UCP&DT	2463.08	3894.02	4090.90	1957.94	0.001	0.002	0.002	0.001
VUCP	2479.04	3906.42	4157.75	1985.35	0.002	0.005	0.004	0.001
OCF&DT	2881.87	4467.11	5327.96	2731.12	0.036	0.036	0.015	0.008
UCP&MLP	86,504.7	54,753.3	136,296.1	117,837.9	0.002	0.001	0.001	0.001
OCF&MLP	88,066.5	55,098.2	139,488.8	146,233.6	0.035	0.035	0.016	0.008
SOCF	93,229.5	65,605.7	154,317.9	151,766.2	0.040	0.039	0.018	0.012

Table 34 shows that as each of the three components was removed from the model and replaced by their substitutes, the average SSE, MAE, and RMSE results for SOCF across all experimental datasets increased, implying a decrease in estimation accuracy in each case. Specifically, the SSE results increase the most when the seven single models (MLR, SVR, MLP, KNN, RF, GB, and DT) are replaced by MLR, SVR, and MLP. Table 35 indicates that removing any of the three SOCF core components increases the SSE, MAE, and RMSE results (t-test p-value of less than 0.05). As a result, the estimation accuracy of SOCF decreased in each case. Thus, the ablation analyses provided an answer to our RQ3.

7 | THREATS TO VALIDITY

This section presents the threats to the validity of this study, specifically internal, construct, conclusion, and external validity.

In terms of internal validity, we highlight each statistical and ML algorithm's unbiased performance evaluation methodology, which should correct for any overfitting of the proposed method.[88,89] The LOOCV method was used in the experiments to select optimal configuration parameters for the statistical and ML algorithms. Because it produces a lower bias and a higher variance estimate than cross-validation, the LOOCV method appears to be a better evaluation method. All of the configuration settings for this study were provided by the GS fine-tuning technique. Adding an additional tuning step would significantly increase the cost of the experiments, and most of the methods in this study performed well with optimally configured parameter values. However, these parameters might not perform well for larger datasets.

Measurement validity is the most serious threat to construct validity. The credibility/reliability of measures was chosen to assess the estimation accuracies of the methods. The accuracy of the SDEE with regard to the MMRE is the most commonly used measure,[9,49] but this measure can be biased.[88,89] As a result, we evaluated the estimation methods using alternative criteria that produced unbiased and symmetric distributions: the SSE, MAE, MAE, RMSE, MIBRE, MBRE, MdMRE, and PRED(0.25).[53]

Conclusion validity is about the ability to draw significant correct conclusions. We carefully applied statistical tests and tested all necessary assumptions. In particular, t-tests (parametric statistical comparison) and Mann–Whitney U tests (non-parametric statistical comparison) were used to demonstrate statistical significance, as presented in Section 6. This research aimed to form the most accurate conclusions regarding the methods. As a result, we can conclude that this study's experimental results are highly generalizable. In addition, we used a medium-sized dataset to mitigate the risk associated with the number of observations that make up the dataset.

The most significant external threat to the study's validity is the generalizability of the ensemble and single techniques' estimation accuracy results. Four datasets were chosen to assess the effectiveness of the ensemble and single techniques in mitigating external threats. These projects were divided into four datasets and covered many domains, including the government, healthcare provider, and organizational domains.[38] One external threat concerns the use of only one GS technique to fine-tune the configuration parameters of each statistical and ML technique. To generalize the results of this study, it is suggested that research be conducted on other optimization techniques.

TABLE 30 The *t*-test results for five different runs of the proposed SOCF method in comparison with the other methods.

Pairs of methods		SOCF vs. UCP	SOCF vs. OCF	SOCF vs. OCF&SVR	SOCF vs. OCF&MLP	SOCF vs. OCF&DT	SOCF vs. OCF&MLR	SOCF vs. OCF&GB	SOCF vs. OCF&RF	SOCF vs. OCF&KNN
SSE	Avg. SSE	1217.71 vs. 44,947.4	1217.71 vs. 41,038.7	1217.71 vs. 2184.80	1217.71 vs. 2279.44	1217.71 vs. 3028.31	1217.71 vs. 2117.07	1217.71 vs. 3,069.72	1217.71 vs. 1847.03	1217.71 vs. 1899.72
	Avg. *p*-value	0.00000	0.00000	0.00190	0.00076	0.00440	0.0514	0.00460	0.00583	0.01199
	St. conc.	>>	>>	>>	>>	>>	<<	>>	>>	>>
MAE	Avg. MAE	6.98 vs. 71.99	6.98 vs. 71.54	6.98 vs. 11.20	6.98 vs. 11.76	6.98 vs. 12.37	6.98 vs. 10.87	6.98 vs. 12.44	6.98 vs. 9.49	6.98 vs. 9.29
	Avg. *p*-value	0.00000	0.00000	0.00000	0.00000	0.00005	0.00001	0.00005	0.00000	0.00005
	St. conc.	>>	>>	>>	>>	>>	>>	>>	>>	>>
RMSE	Avg. RMSE	9.09 vs. 83.14	9.09 vs. 81.09	9.09 vs. 13.97	9.09 vs. 14.63	9.09 vs. 14.63	9.09 vs. 13.19	9.09 vs. 14.53	9.09 vs. 12.03	9.09 vs. 11.99
	Avg. *p*-value	0.00000	0.00000	0.00005	0.00002	0.00006	0.00000	0.00006	0.00005	0.00010
	St. conc.	>>	>>	>>	>>	>>	>>	>>	>>	>>
MBRE	Avg. MBRE	0.02 vs. 0.29	0.02 vs. 0.293	0.02 vs. 0.035	0.021 vs. 0.037	0.02 vs. 0.039	0.021 vs. 0.034	0.02 vs. 0.039	0.02 vs. 0.029	0.02 vs. 0.029
	Avg. *p*-value	0.00000	0.00000	0.00000	0.00000	0.00000	0.00008	0.00009	0.00007	0.00007
	St. conc.	>>	>>	>>	>>	>>	>>	>>	>>	>>
MIBRE	Avg. MIBRE	0.02 vs. 0.199	0.02 vs. 0.203	0.02 vs. 0.032	0.026 vs. 0.034	0.02 vs. 0.036	0.026 vs. 0.032	0.02 vs. 0.036	0.02 vs. 0.028	0.02 vs. 0.027
	Avg. *p*-value	0.00000	0.00000	0.00430	0.00094	0.00126	0.00119	0.00132	0.12496	0.21646
	St. conc.	>>	>>	>>	>>	>>	>>	>>	<<	<<
MdMRE	Avg. MdMRE	0.01 vs. 0.220	0.01 vs. 0.219	0.01 vs. 0.028	0.017 vs. 0.031	0.01 vs. 0.035	0.017 vs. 0.030	0.01 vs. 0.037	0.01 vs. 0.026	0.01 vs. 0.023
	Avg. *p*-value	0.00000	0.00000	0.00011	0.00080	0.00038	0.00057	0.00063	0.00023	0.00019
	St. conc.	>>	>>	>>	>>	>>	>>	>>	>>	>>

Note: A>>B means that A is statistically superior to B.

TABLE 31 The *t*-test results for five different runs of the proposed SOCF method in comparison with the other methods.

Pairs of methods		SOCF Vs. UCP&SVR	SOCF Vs. UCP&MLP	SOCF Vs. UCP&GRNN	SOCF Vs. UCP&KNN	SOCF Vs. UCP&DT	SOCF Vs. UCP&RF	SOCF Vs. UCP&VUCP
SSE	Avg. SSE	1217.71 vs. 3402.95	1217.71 vs. 3752.1	1217.71 vs. 3260.68	1217.71 vs. 3430.46	1217.7 vs. 3243.4	1217.7 vs. 3865.6	1217.71 vs. 2397.76
	Avg. *p*-value	0.00195	0.01032	0.04457	0.00635	0.00251	0.03050	0.00764
	St. conc.	>>	>>	>>	>>	>>	>>	>>
MAE	Avg. MAE	6.980 vs. 13.902	6.980 vs. 14.616	6.980 vs. 12.123	6.980 vs. 14.006	6.980 vs. 13.232	6.980 vs. 12.933	6.980 vs 10.899
	Avg. *p*-value	0.00000	0.00009	0.00038	0.00000	0.00000	0.00020	0.00003
	St. conc.	>>	>>	>>	>>	>>	>>	>>
RMSE	Avg. RMSE	9.096 vs. 16.604	9.096 vs. 17.386	9.096 vs. 14.815	9.096 vs. 16.536	9.096 vs. 15.690	9.096 vs. 15.848	9.096 vs. 13.169
	Avg. *p*-value	0.00000	0.00009	0.00072	0.00001	0.00000	0.00037	0.00007
	St. conc.	>>	>>	>>	>>	>>	>>	>>
MBRE	Avg. MBRE	0.021 vs. 0.044	0.021 vs. 0.046	0.021 vs. 0.039	0.021 vs. 0.043	0.021 vs. 0.042	0.021 vs. 0.041	0.021 vs. 0.034
	Avg. *p*-value	0.00000	0.00017	0.00048	0.00000	0.00001	0.00038	0.00004
	St. conc.	>>	>>	>>	>>	>>	>>	>>
MIBRE	Avg. MIBRE	0.026 vs. 0.040	0.026 vs. 0.042	0.026 vs. 0.035	0.026 vs. 0.040	0.026 vs. 0.038	0.026 vs. 0.037	0.026 vs. 0.032
	Avg. *p*-value	0.00006	0.00070	0.00112	0.00005	0.00010	0.00061	0.00136
	St. conc.	>>	>>	>>	>>	>>	>>	>>
MdMRE	Avg. MdMRE	0.017 vs. 0.039	0.017 vs. 0.038	0.017 vs. 0.032	0.017 vs. 0.039	0.017 vs. 0.039	0.017 vs. 0.035	0.017 vs. 0.030
	Avg. *p*-value	0.00002	0.00011	0.00085	0.00001	0.00016	0.00085	0.000131
	St. conc.	>>	>>	>>	>>	>>	>>	>>

Note: A>>B means that A is statistically superior to B. (cont.).

TABLE 32 The Mann–Whitney U test results for five different runs of the proposed SOCF method in comparison with the other methods.

Pairs of methods		SOCF vs. UCP	SOCF vs. OCF	SOCF vs. OCF&SVR	SOCF vs. OCF&MLP	SOCF vs. OCF&DT	SOCF vs. OCF&MLR	SOCF vs. OCF&GB	SOCF vs. OCF&RF	SOCF vs. OCF&KNN
SSE	Avg. *p*-value	0.00000	0.00000	0.00060	0.00480	0.02280	0.00740	0.01880	0.02280	0.04320
	St. conc.	>>	>>	>>	>>	>>	>>	>>	>>	>>
MAE	Avg. *p*-value	0.00000	0.00000	0.00140	0.00100	0.00340	0.00140	0.00340	0.00480	0.02280
	St. conc.	>>	>>	>>	>>	>>	>>	>>	>>	>>
RMSE	Avg. *p*-value	0.00000	0.00000	0.00210	0.00110	0.01020	0.00340	0.01020	0.01390	0.02070
	St. conc.	>>	>>	>>	>>	>>	>>	>>	>>	>>
MBRE	Avg. *p*-value	0.00000	0.00000	0.00080	0.00050	0.00020	0.00110	0.00200	0.00530	0.02390
	St. conc.	>>	>>	>>	>>	>>	>>	>>	>>	>>
MIBRE	Avg. *p*-value	0.00000	0.00000	0.00400	0.00380	0.01470	0.00920	0.01260	0.09030	0.23650
	St. conc.	>>	>>	>>	>>	>>	>>	>>	<<	<<
MdMRE	Avg. *p*-value	0.00000	0.00000	0.00170	0.00090	0.00040	0.00160	0.00030	0.00530	0.05350
	St. conc.	>>	>>	>>	>>	>>	>>	>>	>>	<<

Note: A>>B means that A is statistically superior to B.

TABLE 33 The Mann–Whitney U test results for five different runs of the proposed SOCF method in comparison with the other methods.

Pairs of methods		SOCF vs. UCP&SVR	SOCF vs. UCP&MLP	SOCF vs. UCP&GRNN	SOCF vs. UCP&KNN	SOCF vs. UCP&DT	SOCF vs. UCP&RF	SOCF vs. UCP&VUCP
SSE	Avg. *p*-value	0.0034	0.0038	0.0092	0.0038	0.0042	0.0092	0.0126
	St. conc.	>>	>>	>>	>>	>>	>>	>>
MAE	Avg. *p*-value	0.0003	0.0003	0.0018	0.0002	0.0006	0.0018	0.0034
	St. conc.	>>	>>	>>	>>	>>	>>	>>
RMSE	Avg. *p*-value	0.0009	0.0005	0.0027	0.0010	0.0013	0.0027	0.0074
	St. conc.	>>	>>	>>	>>	>>	>>	>>
MBRE	Avg. *p*-value	0.0002	0.0002	0.0014	0.0002	0.0003	0.0020	0.0030
	St. conc.	>>	>>	>>	>>	>>	>>	>>
MIBRE	Avg. *p*-value	0.0017	0.0009	0.0133	0.0012	0.0022	0.0139	0.0396
	St. conc.	>>	>>	>>	>>	>>	>>	>>
MdMRE	Avg. *p*-value	0.0003	0.0004	0.0006	0.0000	0.0003	0.0007	0.0013
	St. conc.	>>	>>	>>	>>	>>	>>	>>

Note: A>>B means that A is statistically superior to B. (cont.).

TABLE 34 The results for SOCF-Case1, SOCF-Case2, and SOCF-Case3.

Methods	SSE	MAE	RMSE	MBRE	MIBRE	MdMRE	PRED
SOCF (in Full)	1217.71	6.98	9.10	0.021	0.026	0.017	1.00
SOCF-Case1	↑ 1412.80	↑ 8.10	↑ 10.31	↑ 0.025	↑ 0.024	↑ 0.019	1.00
SOCF-Case2	↑ 1643.75	↑ 8.91	↑ 11.16	↑ 0.028	↑ 0.026	↑ 0.023	1.00
SOCF-Case3	↑ 2146.13	↑ 11.00	↑ 14.03	↑ 0.035	↑ 0.032	↑ 0.030	1.00

Note: The ↑ sign denotes an increase in SSE, MAE, RMSE, MBRE, MIBRE, or MdMRE results, implying a decrease in estimation accuracy compared to the SOCF (in full) model.

TABLE 35 The ablation analyses.

Models for ablation analyses			*p*-value of *t*-test
SOCF-Case1	SSE increase	195.096	0.01166 << Full SOCF model
	MAE increase	1.119	0.00000 << Full SOCF model
	RMSE increase	1.215	0.00008 << Full SOCF model
SOCF-Case2	SSE increase	426.047	0.01914 << Full SOCF model
	MAE increase	1.932	0.00035 << Full SOCF model
	RMSE increase	2.061	0.00011 << Full SOCF model
SOCF-Case3	SSE increase	928.428	0.00070 << Full SOCF model
	MAE increase	4.029	0.00001 << Full SOCF model
	RMSE increase	4.930	0.00001 << Full SOCF model

Note: The term "<< Full SOCF model" refers to the full SOCF model's statistical superiority over models that exclude one of the three core components.

8 | CONCLUSION AND FUTURE WORK

This work introduces a comprehensive approach to complex algorithms based on engineering requirements research for a more accurate estimation of software effort. Specifically, we detailed software effort estimation using ensemble techniques and statistical and ML algorithms on the OCF method. The proposed method incorporates standard statistical and ML techniques into an ensemble design to achieve higher estimation accuracy with the OCF method. In particular, our proposed method combines three key components: optimizing the model parameters with a GS technique, reducing the generalization error with a stacking ensemble, and including seven single models in the stacking ensemble. The stacking ensemble was created by combining RF, KNN, SVR, MLR, MLP, GB, and DT. The GS method was used to find the optimal parameters for each technique for the validation set. These regression-based single learners were then trained using the stacked learner. We conducted experiments on a total of four datasets to demonstrate the effectiveness of our SOCF method more clearly. The estimation accuracy of the proposed method and other methods were evaluated using unbiased performance measures, namely, the SSE, MAE, MAE, RMSE, MIBRE, MBRE, and PRED(0.25).

Based on the experimental results, no single model outperformed the other single models across all experimental datasets. Instead, our new ensemble-based approach, which is an unbiased method for estimating the effort for a new software project, produced the best results across all four experimental datasets. In other words, the SOCF method is highly stable. To provide robust method comparisons, we conducted statistical comparisons using both the *t*-test and the Mann–Whitney U test, which indicated that our SOCF method is statistically superior to the other models we considered. In addition, we performed ablation analyses to evaluate the effectiveness of each of the three core components of the SOCF. The results showed that the average evaluation results increased across all experimental datasets when the three components were progressively removed from the model and replaced by substitutes, implying a decrease in the estimation accuracy compared to the full SOCF model. Overall, our method outperformed the other models tested. We believe that the SOCF method developed in this study will benefit project managers in terms of the pricing process, project planning, iteration planning, budgeting, and investment analysis. In summary, the results obtained can be considered beneficial for industrial applications, as they show that the proposed approach leads to more accurate estimates of the size and complexity of the software.

In the future, we believe that our proposed method can be further improved by identifying specific correction factor components that will help inexperienced developers better design their correction factors. One of our initial ideas for this work is to incorporate the program evaluation and review technique (PERT) into the estimation problem, especially the correction factors. Another possibility is to calibrate the weighting values of the correction factors to reflect the latest trend in the software development industry and improve the accuracy of the proposed methods. Therefore, an approach to calibrate the weights of the correction factors using an artificial neural network will be performed in the future.

ACKNOWLEDGMENTS

This work was supported by the Faculty of Applied Informatics, Tomas Bata University in Zlín, under projects IGA/CebiaTech/2022/001 and RVO/FAI/2021/002.

CONFLICT OF INTEREST STATEMENT

The authors declare that they have no conflicts of interest.

DATA AVAILABILITY STATEMENT

Data and code are provided on the site https://github.com/hltknhung/SOCF.

ORCID

Vo Van Hai https://orcid.org/0000-0002-5427-1960
Radek Silhavy https://orcid.org/0000-0002-5637-8796

REFERENCES

1. Boehm B. *Software engineering economics*. Prentice-Hall; 1981.
2. Boehm B, Abts C, Chulani S. Software development cost estimation approaches—a survey. *Annals of Software Engineering*. 2000;10(1/4):177-205. doi:10.1023/A:1018991717352
3. Trendowicz A, Jeffery R. Software project effort estimation. In: *Foundations and best practice guidelines of success*. Springer; 2014. doi:10.1007/978-3-319-03629-8
4. Jorgensen M, Shepperd M. A systematic review of software development cost estimation studies. *IEEE Transactions on Software Engineering*. 2007; 33(1):33-53. doi:10.1109/TSE.2007.256943
5. Nhung HLTK, Hoc HT, Hai VV. A review of use case-based development effort estimation methods in the system development context. In: *Intelligent systems applications in software engineering*. CoMeSySo; 2019. doi:10.1007/978-3-030-30329-7_44
6. Srichandan S. A new approach of software effort estimation using radial basis function neural networks. *International Journal on Advanced Computer Theory and Engineering*. 2012;1:113-120.
7. Karner G. *Resource estimation for objector projects*. Objective Systems SFAB; 1993.
8. Mahmood Y, Kama N, Azmi A. A systematic review of studies on use case points and experts-based estimation of software development effort. *J Softw Evol Proc*. 2020;32:e2245. doi:10.1002/smr.2245
9. Azzeh M, Nassif AB, Attili IB. Predicting software effort from use case points: a systematic review. *Science of Computer Programming*. 2021;204: 102596. doi:10.1016/j.scico.2020.102596
10. Wen J, Li S, Lin Z. Systematic literature review of machine learning based software development effort estimation models. *Information Software Technology*. 2012;54(1):41-59. doi:10.1016/j.infsof.2011.09.002
11. Kumar PS, Behera HS, Kumari A, Nayak J, Naik B. Advancement from neural networks to deep learning in software effort estimation: perspective of two decades. *Comp Sci Rev*. 2020;38:100288.
12. Silhavy R, Silhavy P, Prokopova Z. Algorithmic optimisation method for improving use case points estimation. *PLoS ONE*. 2015;10(11):e0141887. doi:10.1371/journal.pone.0141887
13. Azzeh M, Nassif AB, Minku LL. An empirical evaluation of ensemble adjustment methods for analogy-based effort estimation. *Journal of Systems and Software*. 2015;103:36-52. doi:10.1016/j.jss.2015.01.028
14. Marapelli B, Carie A, Islam SMN. Software effort estimation with use case points using ensemble machine learning models. In: *International conference on electrical, computer and energy technologies (ICECET)*; 2021:1-6.
15. Pospieszny P, Chrobot BC, Kobylinski A. An effective approach for software project effort and duration estimation with machine learning algorithms. *Journal of Systems and Software*. 2018;137:184-196. doi:10.1016/j.jss.2017.11.066
16. Abdelali Z, Hicham M, Abdelwahed N. An ensemble of optimal trees for software development effort estimation. In: *AIT2S 2018*; 2019:55-68.
17. Mahmood Y, Kama N, Azmi A. Improving estimation accuracy prediction of software development effort: a proposed ensemble model. In: *International Conference on Electrical, Communication, and Computer Engineering (ICECCE). IEEE*; 2020:1-6.
18. Hussain A, Raja M, Vellaisamy P, Krishnan S, Rajendran L. Enhanced framework for ensemble effort estimation by using recursive-based classification. *IET Software*. 2021;15(3):230-238. doi:10.1049/sfw2.12020
19. Elish MO. Assessment of voting ensemble for estimating software development effort. In: *IEEE Symposium on Computational Intelligence and Data Mining (CIDM)*; 2013:316-321.

20. Hosni M, Idri A, Nassif AB, Abran A. Heterogeneous ensembles for software development effort estimation. In: *International Conference on Soft Computing & Machine Intelligence (ISCMI)*; 2016:174-178.
21. van der Vaart AW, Dudoit S, van der Laan MJ. Oracle inequalities for multi-fold cross-validation. *Statistics and Decision*. 2006;24(3):351-371. doi:10.1524/stnd.2006.24.3.351
22. Van der Lann MJ, Dudoit S. Unified cross-validation methodology for selection among estimators and a general cross-validated adaptive epsilon-net estimator: finite sample oracle inequalities and examples. In: *U. C. Berkeley division of biostatistics working paper series*; 2003.
23. Zakaria NA, Ismail AR, Ali AY, Khalid NHM, Abidin NZ. Software project estimation with machine learning. *Int J Adv Comput Sci Appl*. 2021;12(6):726-734.
24. Abdulmajeed AA, Al-Jawaherry MA, Tawfeeq TM. Predict the required cost to develop software engineering projects by using machine learning. *Journal of Physics: Conference Series*. 2021;1897(1):012029. doi:10.1088/1742-6596/1897/1/012029
25. Shukla S, Kumar S. An extreme learning machine based approach for software effort estimation. In: *International conference on evaluation of novel approaches to software engineering*; 2021.
26. Silhavy R, Silhavy P, Prokopova Z. Using actors and use cases for software size estimation. *Electronics*. 2021;10(5):2021. doi:10.3390/electronics10050592
27. Singh AJ, Kumar M. Comparative analysis on prediction of software effort estimation using machine learning techniques. In: *Intelligent communication and computational research (ICICCR-2020)*; 2020.
28. Varshini AGP, Kumari KA. Predictive analytics approaches for software effort estimation: a review. *Indian J Sci Technol*. 2020;13(21):2094-2103. doi:10.17485/IJST/v13i21.573
29. Marapelli B. Software development effort duration and cost estimation using linear regression and k-nearest neighbors machine learning algorithms. *Int J Innov Technol Explor Eng*. 2019;9(2):1043-1047.
30. Shukla S, Kumar S. *Applicability of neural network based models for software effort estimation.* IEEE World Congress on Services (SERVICES); 2019:339-342.
31. Azzeh M, Nassif AB, Banitaan S. Comparative analysis of soft computing techniques for predicting software effort based use case points. *IET Software*. 2018;12(1):19-29. doi:10.1049/iet-sen.2016.0322
32. Abdelalia Z, Mustaphaa H, Abdelwahedb N. Investigating the use of random forest in software effort estimation. *Procedia Computer Science*. 2019; 148:343-352. doi:10.1016/j.procs.2019.01.042
33. Florianoa AG, Martínb CL, Márquezc CY, Abrand A. Support vector regression for predicting software enhancement effort. *Information and Software Technology*. 2018;97:99-109. doi:10.1016/j.infsof.2018.01.003
34. Mustafa AB. *Predicting software effort estimation using machine learning techniques*. Computer Science and Information Technology (CSIT); 2018.
35. Sharma P, Singh J. Machine learning based effort estimation using standardization. In: *International conference on computing, power and communication technologies (GUCON)*; 2018:716-720.
36. Hidmi O, Erdogdu Sakar B. Software development effort estimation using ensemble machine learning. *Int J Comput Commun Instrum Eng*. 2017;4(1):143-147.
37. Satapathy SS, Rath SK. Empirical assessment of machine learning models for agile software development effort estimation using story points. *Innovations Syst Software Engineering*. 2017;13(2-3):191-200. doi:10.1007/s11334-017-0288-z
38. Silhavy R, Silhavy P, Prokopova Z. Analysis and selection of a regression model for the use case points method using a stepwise approach. *Journal of Systems and Software*. 2017;125:1-14. doi:10.1016/j.jss.2016.11.029
39. Satapathy SM, Acharya BP, Rath SK. Early stage software effort estimation using random forest technique based on use case points. *IET Software*. 2016;10(1):10-17. doi:10.1049/iet-sen.2014.0122
40. Azzeh M, Nassif AB. A hybrid model for estimating software project effort from use case points. *Applied Soft Computing Journal*. 2016;49:981-989. doi:10.1016/j.asoc.2016.05.008
41. Minku LL, Yao X. Ensembles and locality: insight on improving software effort estimation. *Information and Software Technology*. 2013;55(8):1512-1528. doi:10.1016/j.infsof.2012.09.012
42. Malgonde O, Chari K. An ensemble-based model for predicting agile software development effort. *Empire Software Engineering*. 2019;24(2):1017-1055. doi:10.1007/s10664-018-9647-0
43. Shukla S, Kumar S, Bal PR. Analyzing effect of ensemble models on multi-layer perceptron network for software effort estimation. In: *IEEE world congress on SERVICES (SERVICES)*; 2019.
44. de Cabral JJTHA, Oliveira ALI. Ensemble effort estimation using dynamic selection. *J Syst Softw*. 2021;175:110904.
45. Priya AG, Anitha KK, Varadarajan V. Estimating software development efforts using a random forest-based stacked ensemble approach. *Electronics*. 2021;10(10):1195.
46. Kumar TM, Jayaram MA. Comparison of hard limiting and soft computing methods for predicting software effort estimation: in reference to small scale visualization projects. *Int J Eng Technol*. 2018;7(4.6):294-298.
47. Myrtveit I, Stensrud E. Validity and reliability of evaluation procedures in comparative studies of effort prediction models. *Empir Softw Eng*. 2012; 17(1):23-33. doi:10.1007/s10664-011-9183-7
48. Nhung HLTK, Hoc HT, Hai VV. An evaluation of technical and environmental complexity factors for improving use case points estimation. In: *Advances in Intelligent Systems and Computing*. Springer; 2020.
49. Nhung HLTK, Hai VV, Silhavy R, Prokopova Z, Silhavy P. Parametric software effort estimation based on optimizing correction factors and multiple linear regression. *IEEE Access*. 2022;10:2963-2986.
50. Fonti V., *"Feature Selection Using LASSO," Research Paper in Business Analysis*, 2017.
51. Muthukrishnan R, Rohini R. LASSO: A feature selection technique in predictive modeling for machine learning. In: *IEEE International Conference on Advances in Computer Applications (ICACA)*; 2016:18-20.
52. Chandraa A, Yao X. Ensemble learning using multi-objective evolutionary algorithms. *Journal of Math Model Algorithms*. 2006;5(4):417-445. doi:10.1007/s10852-005-9020-3
53. Idri A, Abnane I, Abran A. Evaluating PRED(p) and standardized accuracy criteria in software development effort estimation. *J Softw Evol Process*. 2017;30(4):e1925.
54. Nam LNH, Quoc HB. The hybrid filter feature selection methods for improving high-dimensional text categorization. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*. 2017;25(02):235-265. doi:10.1142/S021848851750009X

55. Tulai AF, Oppacher F. Multiple species weighted voting—A genetics-based machine learning system. In: *Genetic and Evolutionary Computation Conference*; 2004:1263-1274.
56. Azzeh M, Nassif AB, Elsheikh Y, Angelis L. On the value of project productivity for early effort estimation. *Science of Computer Programming*. 2022;219:102819. doi:10.1016/j.scico.2022.102819
57. Huang CL, Wang CJ. A GA-based feature selection and parameters optimization for support vector machines. *Expert System Application*. 2006;31(2):231-240. doi:10.1016/j.eswa.2005.09.024
58. Pourjafari E, Reformat M. A support vector regression based model: predictive control for volt-var optimization of distribution systems. In: *IEEE Access*. Vol.7; 2019.
59. Ali A, Gravino C. A systematic literature review of software effort prediction using machine learning methods. *J Softw Evol Process*. 2019;31(10):1-25.
60. Kocaguneli E, Kultur Y, Bener A. Combining multiple learners induced on multiple datasets for software effort prediction. In: *Symposium on software reliability engineering (ISSRE)*; 2009.
61. Kocaguneli E, Menzies T, Keung JW. On the value of ensemble effort estimation. *IEEE Transactions on Software Engineering*. 2012;38(6):1403-1416. doi:10.1109/TSE.2011.111
62. Pahariya JS, Vadlamani R, Mahil C. Software cost estimation using computational intelligence techniques. In: *World Congress on Nature & Biologically Inspired Computing*. IEEE; 2009:849-854.
63. Elish MO, Helmy T, Hussain MI. Empirical study of homogeneous and heterogeneous ensemble models for software development effort estimation. *Math Probl Eng*. 2013;2013:1-21. doi:10.1155/2013/312067
64. Malgonde O, Chari K. An ensemble-based model for predicting agile software development effort. *Empirical Software Engineering*. 2019;24(2):1017-1055. doi:10.1007/s10664-018-9647-0
65. Kumar PS, Behera HS, Nayak J, Naik B. A pragmatic ensemble learning approach for effective software effort estimation. *Innovations System Software Engineering*. 2022;18(2):283-299. doi:10.1007/s11334-020-00379-y
66. Alhazmi O, Khan M. Software effort prediction using ensemble learning methods. *Journal of Software Engineering and Applications*. 2020;13(07):143-160. doi:10.4236/jsea.2020.137010
67. Haykin S. *Neural networks: a comprehensive foundation*. Prentice Hall; 1999.
68. Linoff GS, Berry MJ. *Data mining techniques: for marketing, sales and customer relationship management*. Wiley; 2011.
69. Moller MF. A scaled conjugate gradient algorithm for fast supervised learning. *Neural Netw*. 1993;6(4):525-533. doi:10.1016/S0893-6080(05)80056-5
70. Cortes C, Vapnik V. Support-vector networks. *Machine Learning*. 1995;20(3):273-297. doi:10.1007/BF00994018
71. Hosni M, Idri A, Abran A, Nassif AB. On the value of parameter tuning in heterogeneous ensembles effort estimation. *Soft Computing*. 2017;1-34.
72. Najm A, Zakrani A, Marzak A. Decision trees based software development effort estimation: A systematic mapping study. In: *International Conference of Computer Science and Renewable Energies (ICCSRE)*; 2019:1-6.
73. Quinlan JR. *C4.5: program for machine learning*. Morgan Kaufmann; 1993.
74. Breiman L, Friedman J, Olshen R, Stone C. *Classification and Regression Trees*. Wadsworth; 1984.
75. Hastie T, Tibshirani R, Friedman J. *Elements of Statistical Learning*. Springer; 2009. doi:10.1007/978-0-387-84858-7
76. Breiman L. Random forests. *Machine Learning*. 2001;45(1):5-32. doi:10.1023/A:1010933404324
77. Bailey T, Jain AK. A note on distance weighted k-nearest neighbor rules. *IEEE Transaction Systems, Man Cybernatics*. 1978;8:311-313.
78. Khoshgoftaar TM, Zhong S, Joshi V. Enhancing software quality estimation using ensemble-classifier based noise filtering. *Intelligent Data Analysis*. 2005;9(1):3-27. doi:10.3233/IDA-2005-9102
79. Gollapudi S. *Practical machine learning*. Packt Publishing Ltd.; 2016.
80. Xu L, Krzyzak A, Suen CY. Methods of combining multiple classifiers and their applications to handwriting recognition. *IEEE Trans Syst Man Cybern*. 1992;22(3):418-435. doi:10.1109/21.155943
81. Schapire RE, Freund Y, Bartlett P, Lee WS. Boosting the margin: a new explanation for the effectiveness of voting methods. *The Annals of Statistics*. 1998;26:1651-1686.
82. Wolpert D. Stacked generalization. *Neural Network*. 1992;5(2):241-259. doi:10.1016/S0893-6080(05)80023-1
83. Ochodek M, Nawrocki J, Kwarciak K. Simplifying effort estimation based on use case points. *Information and Software Technology*. 2011;53(3):200-213. doi:10.1016/j.infsof.2010.10.005
84. Subriadi AP, Sholiq PAN. Critical review of the effort rate value in use case point method for estimating software development effort. *J Theor Appl Inf Technol*. 2014;59(3):735-744.
85. Shepper M, MacDonell S. Evaluating prediction systems in software project estimation. *Information and Software Technology*. 2012;54(8):820-827. doi:10.1016/j.infsof.2011.12.008
86. Azzeh M, Nassif AB, Banitaan S, Almasalha F. Pareto efficient multi-objective optimization for local tuning of analog-based estimations. *Neural Computing and Applications*. 2016;27(8):2241-2265. doi:10.1007/s00521-015-2004-y
87. Golberg M, Cho H. *Introduction to Regression Analysis*. University of Nevada; 2010.
88. Cawley GC, Talbot NLC. On over-fitting in model selection and subsequent selection bias in performance evaluation. *Journal of Machine Learning Research*. 2010;11:2079-2107.
89. Krstajic D, Buturovic LJ, Leahy DE, Thomas S. Cross-validation pitfalls when selecting and assessing regression and classification models. *J Chem*. 2014;6:1-15.

How to cite this article: Nhung HLTK, Van Hai V, Silhavy P, Prokopova Z, Silhavy R. Incorporating statistical and machine learning techniques into the optimization of correction factors for software development effort estimation. *J Softw Evol Proc*. 2023;e2611. doi:10.1002/smr.2611

APPENDIX A: LIST OF ABBREVIATIONS

Abbreviation	Meaning of abbreviation	Abbreviation	Meaning of abbreviation
MLP	Multilayer perceptron	ELM	Extreme learning machine
SVR	Support vector regression	SR	Stepwise regression
DT	Decision tree	RR	Ridge regression
RF	Random forest	LR	Lasso regression
MLR	Multiple linear regression	ENR	Elastic net regression
KNN	*K*-nearest neighbor	PNN	Probabilistic neural network
GB	Gradient boosting	RNN	Recurrent neural network
GLM	Generalized linear model	GRNN	General regression neural network
RT	Regression tree	RBFNN	Radial basis function neural network
CNN	Cascade neural network	CCNN	Cascade correlation neural network
ENN	Elman neural network	ANFIS	Adaptive neuro-fuzzy inference system
NBL	Naïve Bayes, Logistic	MMRE	Mean Magnitude Relative Error
BRE	Balanced Relative Error	MAR	Mean Absolute Residual
MSE	Mean Squared Error	RSE	Relative Squared Error
RRAE	Root Relative Absolute Error	RRSE	Root Relative Squared Error
MAR	Mean of Absolute Residual	MdAR	Median of Absolute Residual
AB	Adaptive Boosting ensemble		

APPENDIX B: THE TECHNICAL AND ENVIRONMENTAL FACTORS SELECTED IN EACH DATASET WITH THE λ DETERMINED AND THEIR COEFFICIENT ESTIMATES ARE PRESENTED IN TABLES 35 AND 36

TABLE 36 The estimated TCF coefficients in the LASSO regression.

	D1	D2	D3	D4
λ	0.000231	0.000268	0.000227	0.000236
intercept	0.690619	0.693400	0.720820	0.695850
T1	0.009451	0.009725	0.009547	0.009505
T2	-	-	-	-
T3	0.010897	0.010902	0.010311	0.010456
T4	0.009330	0.008877	0.009888	0.009556
T5	0.010430	0.011130	0.015199	0.010622
T6	0.009576	0.010157	-	0.009202
T7	0.008536	-	0.007298	0.008989
T8	-	-	-	-
T9	0.010551	0.014018	0.013144	0.010334
T10	0.010526	0.010893	0.009730	0.010902
T11	0.007387	0.006516	-	0.005998
T12	-	-	-	-
T13	-	-	-	-

TABLE 37 The estimated ECF coefficients in the LASSO regression.

	D1	**D2**	**D3**	**D4**
λ	0.000177	0.000192	0.000247	0.000327
intercept	1.373478	1.376197	1.404496	1.387716
ENV1	-	-	-	-
ENV2	-	-	-	-
ENV3	−0.032072	−0.042706	−0.032954	−0.033555
ENV4	−0.042291	−0.037886	−0.025558	−0.033001
ENV5	−0.029170	−0.028453	−0.029931	−0.029393
ENV6	−0.028133	−0.027549	−0.030139	−0.029072
ENV7	−0.027981	−0.026382	−0.029221	−0.028660
ENV8	−0.028193	−0.028713	−0.031169	−0.029333