IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 4 February 2023, accepted 3 March 2023, date of publication 13 March 2023, date of current version 23 March 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3256533

RESEARCH ARTICLE

Heterogeneous Ensemble Model to Optimize Software Effort Estimation Accuracy

SYED SARMAD ALI[1,2], JIAN REN[1], KUI ZHANG[1], JI WU[1], AND CHAO LIU[1]
[1]State Key Laboratory of Software Development Environment, School of Computer Science and Engineering, Beihang University, Beijing 100191, China
[2]Department of Computer Science, Mohammad Ali Jinnah University, Karachi 75400, Pakistan

Corresponding author: Jian Ren (renjian@buaa.edu.cn)

This work was supported in part by the State Key Laboratory of Software Development Environment.

ABSTRACT The software industry has experienced rapid expansion in recent years, with software development now essential to the success of many multinational corporations. The demand for complex software systems has dramatically increased, effective software development has become crucial, given the limitations of resources such as money, time, and labor. Cost and effort calculations significantly impact the development process and client needs, and project failure is often caused by errors in job estimating. Underestimating a project's cost and effort can have severe repercussions, such as exceeding the project's budget. Project overruns, on the other hand, can also have a detrimental impact on software projects' successful completion. Researchers and experts in the software industry are continually exploring ways to keep management and development productivity at high levels. However, standalone estimating models have revealed inadequacies over the last decade, and they have not produced any noteworthy research results. Recent literature suggests that opting for ensemble models would yield better results than standalone models. We have proposed a heterogeneous ensemble effort estimation (EEE) model in this research. Our proposed model comprises standalone estimating models such as Use Case Point, Expert Judgment (EJ), and Artificial Neural Network (ANN). We combined the effort of each unique base model using linear combination rule. To validate our model's effectiveness, we applied it to the benchmark dataset, the International Software Benchmarking Standards Group (ISBSG), using three different variations to avoid biases. We further applied the trained models to industry use cases for cross-validation. Our study's findings demonstrated that, in comparison to stand-alone estimate strategies, the ensemble technique produced better estimation results. Finally, our study proposes a heterogeneous ensemble effort estimation model that outperforms standalone models in terms of accuracy. This model has the potential to aid in effective software development, particularly in project cost and effort estimation.

INDEX TERMS Software effort estimation, ensemble effort estimation (EEE), standalone estimation, use case point (UCP), machine learning algorithms, deep learning, expert judgement.

I. INTRODUCTION

Over the past decade, the software industry has undergone significant development, and software development has become essential to the success of many multinational corporations [1]. The development of software that operates effectively within the constraints of cost, effort, and time has become a crucial factor. Both software practitioners and academics are exploring strategies to maintain high levels of productivity in both development and management. During the development of a new software project, effectively managing issues related to cost, time, and labor is critical. However, the cost of software development has risen considerably, presenting significant challenges for businesses. Additionally, accurately estimating project effort and time has become increasingly important. It is not uncommon for software

The associate editor coordinating the review of this manuscript and approving it for publication was Seifedine Kadry.

development to exceed predicted costs and timelines [2], [3]. According to Stanish Group International, around 60% of IT projects were delayed, and 56% of them exceeded the budget [4], [5]. Software effort estimation (SEE), a methodology for determining the amount of labor required to construct a software system, is critical to the success of software project management and related activities [6], [7], [8], [9]. Several factors, such as project duration, cost, and necessary manpower, need to be considered when estimating the effort required for a software project.

Effort appraisal is a critical aspect of software project management and involves several important considerations, including project duration, cost, and necessary manpower [3]. According to Wen et al. [10], the effort required for a project is often measured in man-months or man-hours. Predicting effort is the primary function of software cost estimation, which is why the research community often refers to cost estimation as effort estimation [11]. Developing software with a precise understanding of the required effort is crucial yet challenging. Therefore, both academics and practitioners are actively researching software development effort estimation and duration, as estimating software efforts and planning for resources are necessary for creating a trustworthy software system [12], [13]. In a 2017 assessment conducted by the Project Management Institute (PMI), it was found that while 69% of software projects met their initial goals and business priorities, a significant number of projects faced challenges. Specifically, 43% of projects failed to adhere to their intended budget, 48% experienced delays, and 32% failed to deliver altogether. Large effort overruns can lead to dissatisfied customers, substandard software, and frustrated software developers [14].

We have proposed Heterogeneous ensemble model which is a combination of Use Case Point (UCP), Artificial Neural Network and Expert Judgement (EJ). In the next section, we are going to discuss several model which were used in the literature to predict software effort estimation. However, our proposed model Heterogeneous ensemble models can be more effective than single models or homogeneous ensemble models for software effort estimation for several reasons:

- Diverse set of models: A heterogeneous ensemble model combines different types of models, such as decision trees, neural networks, and regression models, to leverage the strengths of each model and reduce the weaknesses. This can lead to a more accurate and robust estimation of software effort.
- Improved generalization: A heterogeneous ensemble model can improve the generalization of software effort estimation by reducing the impact of bias and variance. Bias refers to the systematic errors in the model, while variance refers to the sensitivity of the model to changes in the training data. A heterogeneous ensemble model can reduce both bias and variance by combining different models that have different strengths and weaknesses.
- Robustness to noise and outliers: A heterogeneous ensemble model can be more robust to noise and outliers in the training data because it combines multiple models that are trained on different subsets of the data. This can reduce the impact of noisy or outlier data points on the final prediction.
- Scalability: A heterogeneous ensemble model can be more scalable than a single model or a homogeneous ensemble model because it can leverage the strengths of different types of models and handle different types of input data.

Overall, a heterogeneous ensemble model can be more effective than single models or homogeneous ensemble models for software effort estimation because it combines different types of models to leverage their strengths, reduce weaknesses, improve generalization, and be more robust to noise and outliers.

II. LITERATURE REVIEW

Effort estimation, which refers to the prediction of time and resources needed for software development, has been extensively studied by both practitioners and researchers. According to Leung and Fan [12], the process of evaluating software effort has evolved to include the cost and resources required to produce software products. Effort estimation models were first introduced in the early 1950s [15], and since then, numerous models have been presented by Boehm [16], [17], Putnam [18], Albrecht [19], [20], SEER-SEM [21], Rubin [22], and Yan-Chin and McDevitt [23]. These models aim to efficiently allocate resources to meet project requirements. However, none of these estimation models can be recognized as trustworthy and cannot be used as a standard because effort estimation is an unsettled and open-ended subject. Cost estimation models often fail to provide accurate and reliable projections, possibly due to their inadequacy in the firm's environment.

During his research, Heemstra [24] discovered that 30% of companies do not estimate their budget. Furthermore, among the total of 598 organizations surveyed, 50% do not keep track of active projects. Inaccurate estimation has been shown to cause 80% of businesses to overestimate expenses and deadlines. The CHAOS research by The Standish Group [4] suggests that inaccurate effort estimates account for approximately 24% of project failures. In 2015, The Chaos Report from The Standish Group International revealed that 56% of IT projects exceeded their budget, while 60% were completed late. Factors contributing to improper and inaccurate effort estimation include the use of inexperienced estimators or premature estimating. However, the accurate estimation of effort is essential for generating requirements for discussions, planning, bids, contract monitoring, and control. It also helps in the more efficient preparation of resources for stakeholders. Over the past three decades, researchers and practitioners have collaborated to develop cost estimating models and techniques for more accurate effort assessment. A mathematical construct called a cost estimation model combines formulas or algorithms for estimating the amount of time and work

required to complete a software project. For the last thirty years, researchers and practitioners have worked together to develop models and techniques for accurately estimating the costs of software development efforts. One such model is the cost estimation model, which is a mathematical construct that combines formulas or algorithms to estimate the amount of time and work required to complete a software project.

Software effort estimation is a crucial task for software project managers, involving the prediction of the time and resources required to develop a software system. This process is essential for planning and allocating resources, setting project timelines, and communicating project progress to stakeholders. Several approaches to software effort estimation exist, including expert judgment, algorithmic methods, and machine learning techniques.

Expert judgment involves seeking input from experienced software developers or project managers who possess knowledge of the specific work being undertaken. This approach relies on the expertise and knowledge of individuals providing the estimates, which may be subjective in nature. Expert judgment has several variations, including the Delphi Technique, Wide-Delphi Technique, Planning Poker, Work Breakdown Structure, and Activity-Based Model. Delphi is a consensus-based approach to effort estimation that involves a team of professionals, including a software developer, an estimation expert, and an expert from the application area. This approach provides an ample communication channel for the professionals to debate and discuss essential data and information necessary for their cooperation and internal estimations. The Wide-band Delphi technology was introduced by the Rand Corporation and was further improved upon by Barry Boehm and John Farquhar in the 1970s [25]. One of the primary benefits of this strategy is the one-on-one interaction between specialists, which facilitates planning, scheduling, and estimating. The core principles of Wide-band Delphi are grounded in group-based software cost estimation, where required effort is computed using the group's consensus. Planning Poker is another consensus-based estimation method, similar to Wideband Delphi. This strategy was initially proposed by Grenning in 2002 [26] and later promoted by M. Cohn in his book from 2005 [27]. Planning Poker is often used in agile development due to its alignment with the people-focused Agile principles.

Algorithmic methods involve the use of a predetermined set of rules or formulas to calculate estimates based on specific inputs. These methods can be based on historical data or best practices in the field. During the 1970s, Lines of Code (LOC) were considered the foundation for effort estimation, and numerous estimation models were developed based on LOC on various datasets [28]. The Putnam SLIM is one of the earliest algorithmic cost models and is commonly regarded as a macro estimate model based on the Norden/Rayleigh function [18]. Functional Point Analysis (FPA) was established to determine the time and money spent developing new software applications as well as maintaining those that already exist [19], [20]. Statistical modeling involves using statistical techniques to analyze data from past projects to develop models that can be used to predict future effort. This approach may be more accurate than expert judgment or algorithmic methods, but it requires a significant amount of data and may be more complex to implement. The use case point model was first proposed by Karner in 1993 and used to calculate the software development effort with the aid of the use case diagram. An early effort estimate based on use cases can be developed if one has a good understanding of the issue domain, system size, and architecture [29].

Since the 1990s, many researchers have suggested using machine learning (ML) based Software Development Effort Estimation (SDEE) models to improve estimation accuracy. Wen et al. [10] conducted a study that found that eight different types of ML techniques, including Case-Based Reasoning (CBR), Artificial Neural Networks (ANN), Decision Trees (DT), Bayesian Networks (BN), Support Vector Regression (SVR), Genetic Algorithms (GA), Genetic Programming (GP), and Association Rules (AR), have been used to estimate software development effort. Among these algorithms, CBR and ANN were found to be the most dominant in estimating effort for software projects. The aforementioned ML techniques were often used singly or in combination to forecast software development effort. To build a combination form, ML techniques can be mixed with non-ML techniques or two or more ML approaches. Fuzzy logic and general algebra are two popular non-ML methods that are commonly used with other ML methods [10]. Regardless of the approach taken, it is important to carefully consider all relevant factors when estimating software effort, including the complexity of the project, the skills and experience of the team, and the resources available. It is also important to regularly review and update estimates as the project progresses, as changes in project scope or other factors may affect the overall effort required. A comprehensive review of literature on software effort estimation revealed a plethora of models, frameworks, and approaches put forth by researchers and practitioners to achieve high prediction accuracy. However, no single approach has been able to precisely calculate software effort. In their analysis of 304 studies, Jorgensen and Shepard found that the regression method was the most commonly used (49%) approach for assessing effort [6]. Recently, machine learning (ML) methods such as artificial neural networks (ANN) and support vector regression (SVR) have gained attention. Among these, CBR (37%) and ANN (26%) are the most frequently applied ML techniques [10]. When evaluating software effort, ML techniques tend to offer greater accuracy compared to non-ML methods (with means of Pred (25) = 46% and mean of MMRE for CBR, compared to mean of Pred (25) = 64% and mean of MMRE = 37% for ANN). Idri et al. conducted a comprehensive analysis of analogy-based software effort estimation techniques and found that the mean prediction accuracy values of MMRE, MdMRE, and Pred(25) were 49.8%, 29.7%, and 51.23%, respectively. Wu et al. [30]

introduced a hybrid technique that fuses CBR and PSO for estimating software effort, incorporating Euclidean distance, Manhattan distance, and grey relational grade, which are commonly employed CBR techniques in SEE, with optimal weights obtained using the PSO method.

III. PROBLEM STATEMENT

Software practitioners and the academic community have been concerned about accurately assessing efforts for software projects. Since the software projects complexity have been increased over the years, it is becoming increasingly difficult to determine the effort of these enormous software projects, as they also required huge investments for many industries [31]. The research community is currently being compelled to adopt new approaches to optimize the accuracy in predicting the effort due to the significant innovation and expansion in the usage of new techniques and development frameworks/methodologies in the field of software engineering. Despite the significant efforts made to offer new, distinct models and frameworks, there is still much that can be done to improve the accuracy of effort estimation. The inaccuracy of the software industry's estimates of the effort required to create software applications has previously been highlighted by researchers [31], [32]. Unquestionably, substantial over- or under-estimations endanger a software project in several ways. The quality and maintainability of software projects could suffer from an underestimation of tasks (such as testing and documentation) that were to be abandoned or more staff were to be hired [33]. This research study's major goal is to propose an effort estimating ensemble model to increase the precision of software development effort prediction. To maximize effort accuracy, the suggested heterogeneous model has been incorporated with Use Case Points (UCP), Artificial Neural Network (ANN), and Expert Judgment (EJ) approaches. It is analyzed using case studies from software development companies, industry experts, archived data on estimations, and evaluation indicators to use a quantitative methodology. The software development companies and practitioners will utilize the proposed model created after this project as a tool to estimate the effort needed to develop software projects.

IV. PROPOSED SOLUTION

To obtain high-effort estimation accuracy, professionals and academics have offered a variety of estimating approaches based on the before stated strategies. In the beginning, researchers used various separate estimation techniques to gauge the project's first effort. Although there is evidence in the literature to the contrary, according to Wen et al. [10], [34], no standalone/solo estimating model has presented an accurate estimation. Recent proposals for new ensemble estimation arrangements include Kocaguneli et al. [35] and Minku and Yao [36]. The ensemble method in software development effort estimating was developed to address the shortcomings of standalone estimation strategies (SDEE). To create an ensemble estimation model(EEE), approaches combine various classical estimation models. To anticipate the software development effort of a new project using a combination rule, such as mean, median, and Inverse Rank Weighted Mean-IRWM, an ensemble effort estimating technique combines more than one standalone model [37]. Each base model's estimation is integrated to create an ensemble's estimation. To increase the efficiency and accuracy of software development projects, a heterogeneous ensemble model is suggested in this article that combines Artificial Neural Networks (ANN), Use Case Points (UCP), and Expert Judgment (EJ).

A. RESEARCH QUESTIONS

This study's principal research question is: *"How to improve/optimize software effort estimation accuracy"*.

- RQ-1 Does a heterogeneous ensemble effort estimation model using Use Case Points (UCP), Artificial Neural Network (ANN) and Expert Judgement (EJ) produced better results as compared to Standalone ML models?
- RQ-2 Does Heterogeneous Ensemble Model produce better results as compared to Machine Learning Ensemble model?
- RQ-3 Does the variation in Feature selection of the data-set have any impact on result accuracy or comparison

V. METHODOLOGY

In this section, we are going to present the methodology used for the integration of the classical standalone model to form a heterogeneous ensemble model. We have combined ANN, UCP, and EJ to form an ensemble model.

A novel data mining approach based on ensembles or combinations of methods is now being used to address prediction issues. Studies [10], [38], and [35], on data mining show that ensemble methods outperform single methods in terms of accuracy. The software engineering community has been inspired to create and test ensemble techniques across a range of areas as a result. In the literature on methods for estimating software development work, ensemble effort estimation (EEE) is defined as a combination of several single estimation procedures, or base models, under a certain combination rule. The effort prediction of an EEE technique is the total of the estimations from each constituent base model under a specific combination rule. Elish [39] claim that there are two different categories of EEE techniques: homogeneous approaches and heterogeneous approaches. In this study, we are going to employ a heterogeneous approach

A. HETEROGENEOUS ENSEMBLE FRAMEWORK

EEE methodologies combine different classical estimating models to generate an ensemble estimation model. An ensemble effort estimation technique combines more than one standalone model to predict the software development effort of a new project using a combination rule, like mean, median, and Inverse Rank Weighted Mean-IRWM. The estimation of an

ensemble is produced by integrating the estimation of each base model. In this study, a heterogeneous ensemble model that integrates use case points (UCP), expert judgment (EJ), and artificial neural networks (ANN) is developed to improve the accuracy and efficiency of software development projects. The whole framework of ensemble model is presented in Figure 6.

B. DATASETS USED IN EXPERIMENTATL SETUP

We have used two datasets for our experiments i.e. ISBSG dataset (benchmark dataset) and eight industrial projects. Following are the details of the datasets.

1) ISBSG DATASET

The ISBSG (International Software Benchmarking Standards Group) dataset is a collection of data on software development projects that have been compiled and maintained by the ISBSG. The dataset includes information on various aspects of software development projects, including project size, duration, effort, and cost. The goal of the ISBSG is to provide organizations with a source of real-world data that can be used to benchmark the performance of their software development processes and to identify best practices for improving software development efficiency. The ISBSG dataset is widely used in software engineering research and practice to support the development of software cost and effort estimation models, as well as to evaluate the effectiveness of different software development methodologies and tools [31], [32].

Feature Selection and Cleansing of ISBSG Dataset We had approximately 256 features in total. There are a few empty values in the column. Such columns might have a direct impact on the results. Given that NULL values have no description. Practitioners and academics avoid these values as a result. Moreover, we eliminated all columns with missing values bigger than 50. By removing redundant and unnecessary data, feature selection (FS) techniques have been used in the field of SDEE to minimize the dimensionality of a dataset. The SDEE algorithms are trained on a dataset with relevant information, to increase the precision of their estimations. A dataset with N features must be provided to FS for it to select the ''best'' feature subset from among the 2N competing candidate subsets. Which subsets are best depends on the task at hand, so a subset picked by one evaluation function might not be the same as a subset picked by another.

2) PRIMARY DATASET (INDUSTRIAL CASE STUDIES)

The goal of this section is to observe the phenomenon of effort estimation in practical settings. As a result, a specific case study technique is used in the investigation, which encourages the examination of a phenomenon in its natural setting. Software engineering should be improved via the case study method. In a sample case study, two academic projects and two commercial software projects are observed and analyzed using historical data on estimations [40]. Although observational studies are typically carried out in a real-world setting, the researchers used software development projects as target cases and observed them in genuine enterprises. The fundamental benefit of observational studies is the wealth of information they offer, which enables us to analyze a phenomenon in a context that is appropriate to real-world situations.

VI. FRAMEWORK OF HETEROGENEOUS ENSEMBLE MODEL

A. MODEL DEVELOPMENT

A novel data mining approach based on ensembles or combinations of standalone algorithms is now being considered to address the problem of effort prediction. As a result, the community of software engineers has been motivated to develop and evaluate ensemble techniques in a variety of fields. Ensemble effort estimation (EEE) is described as a combination of various single estimation approaches, or base models, under a certain combination rule in the literature on methodologies for estimating software development work [10], [35], and [38]. The sum of the estimations from each constituent base model under a certain combination rule constitutes the effort prediction of an EEE approach.

We have developed three base model i.e.

- BM1: Use Case Point (UCP), as shown in Figure 2
- BM2: Artificial Neural Network (ANN), as shown in Figure 3
- BM3: Expert Judgement (EJ), as shown in Figure 5

We have combined all the three base (BM1, BM2, BM3), using the linear combination rule as suggested by Hosni et al. [38].

In homogeneous ensemble model, base model is combined with at least two alternative configurations, or one ensemble learning approach is combined with one base model, such as bagging [41], negative correlation [42], or randomization [38]. Homogeneous Ensemble Model, according to Idri et al. [43], can be further subdivided into two groups:

- Ensembles made up of at least two configurations of a single SDEE technology
- Ensembles, such as bagging, boosting, negative correlation, and random subspace, that combine a single meta model and a single SDEE strategy.

There is no agreement on the techniques for measuring software effort that result in the most precise models, despite decades of research. Previous studies show that no single strategy consistently beats the others when M-estimating approaches are used. Instead of selecting one estimating method as the best, it could be wiser to develop estimates using ensembles of a few different methods. When the estimates of many estimators are integrated [44], it is found that the combined techniques outperform any one estimator. According to Shepperd, no standalone model compete with the strength and diversity of ensemble models [6], [45].

TABLE 1. Descriptive statistics of use cases.

Case ID	Case Name	No. of Actors	No. of Use Cases
CS1	Inventory Management System	6	30
CS2	Smart Home Automation	2	11
CS3	Pre-Owned Accessories	7	5
CS4	BAM Electra	2	7
CS5	Sign Language Translator	3	8
CS6	Spect AR	2	8
CS7	Digital Identity	7	26
CS8	The Educational Network (TEN)	2	13

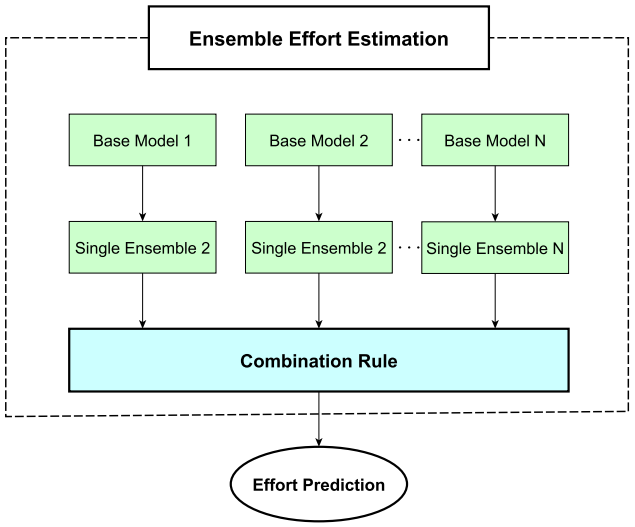


FIGURE 1. Ensemble effort estimation (EEE) processes [14], [38].

B. LINEAR COMBINATION RULE

Combining the outputs of base models yields ensemble effort estimates. To combine our base models, we have used the Median combination rule. Idri et al. [38] identified 18 rules used to get prediction values from an ensemble. Majorly there can be distinguish into two categories:

- Linear Combination Rule: In a linear combination, the output of each base model is multiplied by a weight, and the resulting outputs are added together to produce the final prediction. This can be expressed mathematically as:

$$y = w1 * y1 + w2 * y2 + \ldots + wn * yn \qquad (1)$$

where y is the final prediction, y1, y2, …, yn are the predictions of the base models, and w1, w2, …, wn are the weights assigned to each base model. The weights can be determined using various techniques, such as cross-validation or grid search.

- Non-Linear Model: In a non-linear combination, the output of each base model is first transformed using a non-linear function, and then the resulting outputs are combined to produce the final prediction. This can be expressed mathematically as:

$$y = f(w1 * g(y1) + w2 * g(y2) + \ldots + wn * g(yn)) \qquad (2)$$

where f is the non-linear function and g is the transformation function. The non-linear function can be any function that maps the weighted sum of the base model outputs to the final prediction. The transformation function can be any function that maps the output of the base model to a new space where the non-linear function can better capture the relationships between the base model outputs.

TABLE 2. Combination Rule.

Type	Combination Rule
Linear	Mean, Mean weighted, Median, Inverse ranked weighted mean (IRWM), Weighted adjustment based on criterion, Equally weighted, Median weighted, Outperformance combination, Geometric Mean. Harmonic Mean
Non-linear	MLP, SVR, Adaptive Resonance Theory (ART), Fuzzy inference system using fuzzy c-means (FIS-FCM), Fuzzy inference system using subtractive clustering (FIS-SC), ANFIS-FCM, ANFIS-SC, MLR, RBF, DENFIS

1) MEDIAN COMBINATION RULE

The median linear combination rule is a common method for combining the predictions of different models in a heterogeneous ensemble. In this case, we have three base models: Use Case Point (UCP), Artificial Neural Network (ANN), and Expert Judgement (EJ). Here's how we can use the median linear combination rule to create a heterogeneous ensemble model:

Train the three base models on the same training data set. Generate predictions for the test data set using each of the three models. Combine the predictions by taking the median of the three predictions for each test instance. That is, for each instance in the test set, we take the median of the predicted values from the UCP model, the ANN model, and the EJ model. Use the resulting combined predictions as the final output of the ensemble model. The median linear combination rule is a robust method for combining the predictions of different models, as it is not sensitive to extreme values or outliers in the individual model predictions. Additionally,

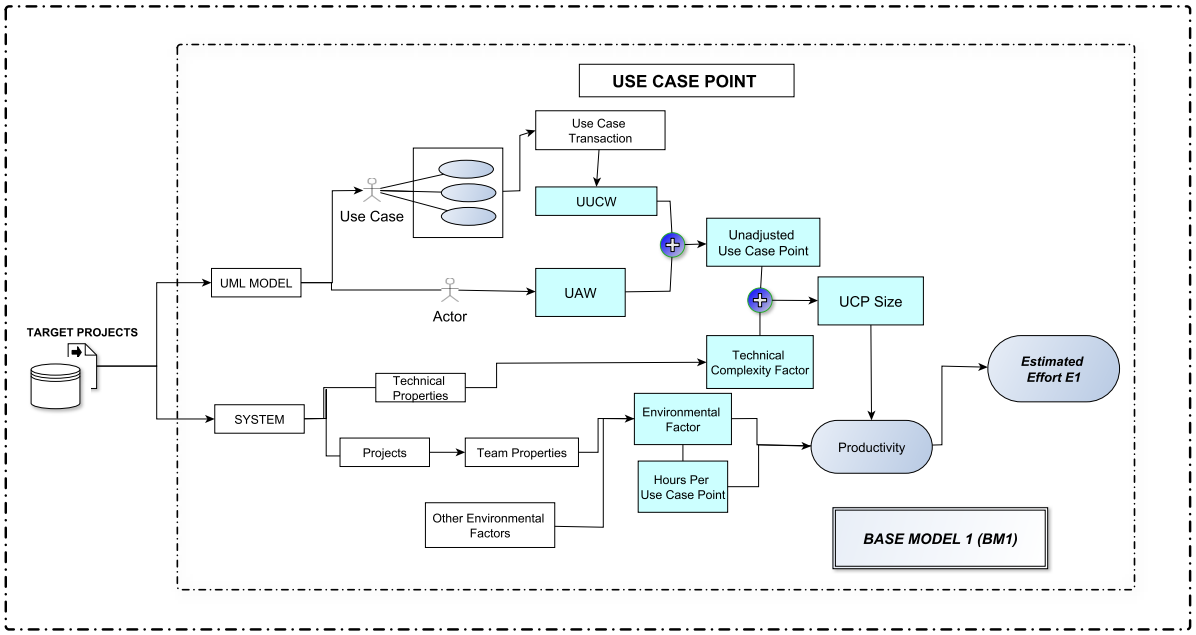


FIGURE 2. Use Case Point Base Model 1 (BM1).

by combining different modeling techniques like UCP, ANN, and EJ, we can leverage the strengths of each method to create a more accurate and robust ensemble model.

C. MACHINE LEARNING MODELS EXPERIMENTAL SETUP ON THE ISBSG DATASET FOR FEATURE SELECTION

The ISBSG dataset served as the basis for our experiments. We used the most well-known machine learning techniques for this experiment, including Support Vector Regressor (SVR), Linear Regression, K-Nearest Neighbor (KNN), XGBoost Regressor, and Artificial Neural Network(ANN). To prevent bias in the outcomes, we will employ many dataset variations. For instance, we employed the ISBSG features that were evaluated in the ways listed below.

1) **DATA VARIATION 1**: Applying all features on ISBSG dataset.
   We are going to incorporate all the features of ISBSG dataset i.e. 256 features.
2) **DATA VARIATION 2**: Using selected features on recommendation of Nassif et al. [46]
3) **DATA VARIATION 3**: Applying Results of Statistical Feature Selection.

We have all undoubtedly faced the difficulty of removing the irrelevant or insignificant elements from a set of data that do not significantly affect our target variable to increase the precision of our model. The performance of a model is significantly impacted by feature selection, as it is considered one of the core principles of machine learning. The data properties to train machine learning models have a significant impact on the performance one can achieve. The performance of the model could be negatively impacted by features that are irrelevant or only partially relevant. Feature selection and data cleansing should be the first and most important steps in the development of any machine learning model. Whether features are chosen manually or automatically will depend on the traits that are crucial to prediction variables or desired results. The model might learn based on irrelevant features in the given data, which could lower model accuracy.

D. USE CASE POINT (UCP)

Base Model (BM1) is presented in Figure 2 i.e. UCP. The use case point model was first introduced by Karner [48]. This model was used to calculate the effort of software development with the help of the use case diagram. If the idea about the problem domain, system size, and architecture is clear, then an early effort estimation focused on use cases could be made [29]. The overall working of our first base model i.e. UCP is shown in Figure 2 and the calculation of UCP is shown in Equation 3. After classifying actors and use cases, we calculate unadjusted weights and points for actors and use cases from Equation 4 and Equation 5. After calculating the unadjusted weights, we combine them to get Un-adjusted Use Case Point (UUCP) in Equation 6.

According to [47], following are the actions that must be taken in order to obtain an estimate using the UCP method:

1) Determine the UUCPs and compute them.
2) Calculate and determine the TCFs.
3) Calculate and determine the ECFs.
4) Calculate the PF.
5) Estimate the number of hours required.

The equation can be used to predict the number of man-hours required to accomplish a project when productivity is added as a time-expressing coefficient. Here's the whole

equation Equation 3, which includes a Productivity Factor (PF):

$$UCP = UUCP * TCF * EF * PF \tag{3}$$

The UUCP value obtained from the above equation is altered based on the weights allotted to 13 Technical Complexity Factors(TCF) and 8 Environmental Factos(EFs). After calculating the Technical Complexity Factor (TCF) and Environmental factor, we can find the final Use Case Point by simply adding Un-adjusted Use Case Point (UCP), Technical and Environmental Factor as in Equation 8 to get Effort 1 i.e. Yi as shown in Figure 2.

E. UNADJUSTED USE CASE WEIGHT (UUCW)

The UUCW is calculated as given in Table 4 using the number of use cases in the three categories (simple, average, and complex). Each use case's scenario's number of stages, including alternate flows, is categorized. It is critical to remember that the estimation is influenced by the number of phases in a scenario. The UUCW will be skewed toward complexity in a use-case scenario with several phases and rising UCPs. The UUCW will be skewed toward simplicity and the UCPs will be kept to a minimum if only a few steps are taken. Countless steps can be skipped, without having an adverse effect on the business process. The UUCW is determined by counting the number of use cases in each category, multiplying each total by the weighting factor indicated, and then adding the products.

F. UNADJUSTED ACTOR WEIGHT (UAW)

A use case becomes slightly more challenging when communicating with actors, who have established APIs. It is slightly more complex when interacting with actors who have established protocols, and significantly more complex when interacting with actors who have developed GUIs. Table 6 lists the actors along with explanations of their complexity, classifications, and numerical weighting.

$$UAW = \sum_{i=1}^{3} N_i * W_i \tag{4}$$

Unadjusted Actor Weight (UAW) is calculated by summing the weights of all actors shown in Equation 4. The detailed calculation of UAW is shown in Table 7.

$$UUCW = \sum_{f=1}^{3} P_j * X_j \tag{5}$$

$$UUCP = UAW * UUCW \tag{6}$$

$$Tfactor = \sum_{i=1}^{13} T_i * W_i \tag{7}$$

$$EFactor = \sum_{f=1}^{8} E_i * W_i \tag{8}$$

$$UCP = UUCP * TCF * EF \tag{9}$$

G. UNADJUSTED USE CASE POINT (UUCP)

The sum of UAW and UUCW is the Unadjusted Use Case Point (UUCP). It supplies the system's unadjusted size, which is indicated in Equation as Unadjusted Use Case Points.

$$UUCP = UAW * UUCW \tag{10}$$

H. TECHNICAL COMPLEXITY FACTOR (TCP)

The Technical Complexity Factor (TCF) is an important criterion for UUCP modification [48]. TCF has an impact on project performance since it is derived using 13 technical parameters (T1-T13) shown in Table 8. These variables describe the project's non-functional requirements, ranging from 0 (insignificant) to 5 (very important) (very relevant). Technical Factor (TFactor) and Technical Complexity Factor (TCF) are determined using Equation 11 and Equation 12. The Technical Factor (TFactor) is calculated by summing the values of the technical factors (T1–T13) multiplied by their weight.

$$Tfactor = \sum_{i=1}^{13} T_i * W_i \tag{11}$$

where,

- Ti=takes values between 0 and 5
- Wi=complexity weight

$$TCF = 0.6 + (0.01\ TFactor) \tag{12}$$

I. ENVIRONMENTAL COMPLEXITY FACTOR (ECF)

The Environmental Complexity Factor is a crucial feature used in UUCP modification (ECF). Eight environmental elements (E1-E8) are used to calculate the effects of environmental factors on productivity, as shown in Table 9. The ranking of the project is determined by these factors, which range from 0 (insignificant) to 5 (very important) (very relevant). Below equations are used to compute the EFactor and ECF in Equation 13. To calculate the ECF, multiply the environmental factors (E1–E8) by their weight, then sum all of the figures to get the EFactor.

$$EFactor = \sum_{f=1}^{8} E_i * W_i \tag{13}$$

where,

- Ei=takes values between 0 and 5
- Wi=complexity weight

$$ECF = 1.4 + (0.03 EFactor) \tag{14}$$

J. PRODUCTIVITY FACTOR

The productivity element is very significant when evaluating software effort. It is described as a ratio of size to effort. Once the modified UCP has been determined, the UCP and productivity are multiplied as given in Equation 15.

$$Effort = Productivity\ UCP \tag{15}$$

TABLE 3. Use Case Categories.

Use Case Category	Description	Weight
Simple	The user interface is simple. Only affects one database entity. Its success scenario consists of three or fewer steps. It is implemented with fewer than five classes.	5
Average	Work on the user interface continues. At the same time, it affects two or more database entities. There are four to seven steps. It is implemented using five to ten classes.	10
Complex	Complex user interface or processing. Touches three or more database entities. More than seven steps. Its implementation involves more than 10 classes	15

TABLE 4. Computing UUCW.

Use Case Category	Description	Weight	No. of use Cases	Results
Simple	The user interface is simple. Only affects one database entity. Its success scenario consists of three or fewer steps. It is implemented with fewer than five classes.	5	7	35
Average	Work on the user interface continues. At the same time, it affects two or more database entities. There are four to seven steps. It is implemented using five to ten classes.	10	13	130
Complex	complex processing or user interface. relates to three or more database objects. additional to seven steps. In order to implement it, more than ten classes are needed.	15	3	35

TABLE 5. Use Case complexity detailed.

Use Case Category	Use Case Weight	No. of Use Case	Product
Simple	5	S_UC	$5 * S_UC$
Average	10	A_UC	$10 * A_UC$
Complex	15	C_UC	$15 * C_UC$
			$UAW = 5 * S_UC + 10 * A_UC + 15 * C_UC$

TABLE 6. Use Case Categories.

Actor Category	Description	Actor Weight
Simple	The actor represents a system that has a well-defined API.	1
Average	The actor represents another system with which it communicates via a protocol (e.g., TCP/IP, FTP, HTTP, database, text-based interface).	2
Complex	The actor is a representation of a human who interacts with a graphical user interface.	3

TABLE 7. Actor complexity detailed.

Actor Category	Actor Weight	No. of Actors	Product
Simple	1	S_A	$1 * S_A$
Average	2	A_A	$2 * A_A$
Complex	3	C_A	$3 * C_A$
			$UAW = 1 * S_A + 2 * A_A + 3 * C_A$

Numerous factors, such as the kind of software process, the expertise of the developers, team communications, the environment, and deliverables, affect the value of the productivity factor. Based on previously finished projects, the productivity factor is computed. However, this productivity can only be used to calculate the labor for a new project when the difficulty of the current project and the old project are both identical.

K. ARTIFICIAL NUERAL NETWORK (ANN)

We have used Artificial Neural Network as our second base model. Inspired by the neural network produced by biological neurons, a model called a neural network (NN) was developed is our Base Model (BM2). The artificial neuron serves as the building block for creating a NN. A vector of numerical values serves as the input for an artificial neuron. Each value or component of the vector is translated by the neuron using its weight, which is a discrete, independent sensitivity. The neuron chooses its internal state after receiving the input vector before deciding on its output value. The inner product of the input vector, weight vector, and bias represents the internal state of the neuron. A transfer function is another name for this function [49]. A prominent criticism of the use of ANNs in prediction is their sensitivity to initial weight values and choices for training, validation, and test sets. When properly constructed, ANN ensembles can lessen these effects and produce dependable findings [50].

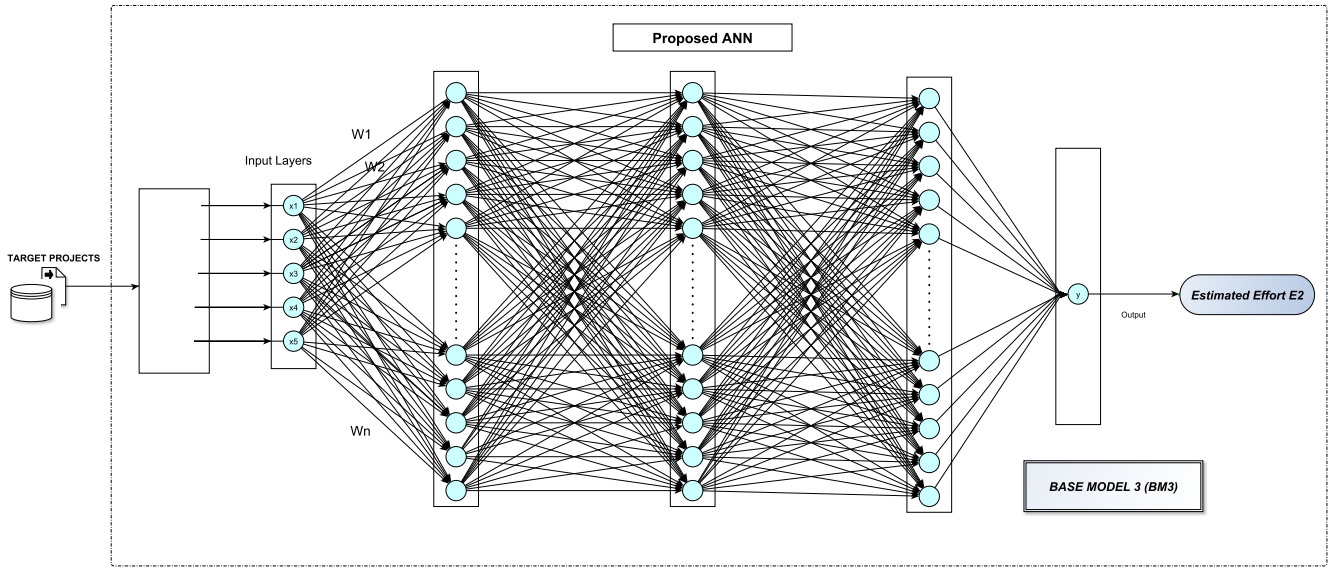


FIGURE 3. Artificial Neural network Base Model 2 (BM2).

TABLE 8. Technical complexity factors [48].

Factor	Description	Weight
T1	Distributed System	2.0
T2	Response time or throughput performance objectives	1.0
T3	Efficiency of end user	1.0
T4	Complicated internal processing	1.0
T5	Reusability of the code	1.0
T6	Easy to install	0.5
T7	Usability	0.5
T8	Portability	1.0
T9	Changeability	1.0
T10	Concurrent	1.0
T11	Inclusion of special security features	1.0
T12	Provides direct access for third parties	1.0
T13	Special user training facilities are required	1.0

TABLE 9. Environmental complexity factors [48].

Factor	Description	Weight
E1	Familiar with the project model that is used	1.5
E2	Application experience	0.5
E3	Object-oriented experience	1.0
E4	Lead analyst capability	0.5
E5	Motivation	1.0
E6	Stable requirements	2
E7	Part-Time Staff	-1.0
E8	Difficult programming language	-1.0

A single neuron with adjustable weights and bias makes up the single-layer feedforward class. An example of a multilayer feedforward neuron consists of an input layer made up of a group of neurons, one or more hidden layers, and an output layer. The multilayer perceptron is another name for this type of neural network (MLP). Each neuron's model has a nonlinear activation function, and the network is very interconnected.

The intrinsic state of the cell influences how it reacts to outside stimuli. This function is known as an activation function. The fundamental responsibility of the activation function is to convert each potential internal state value into a desirable range of output values. The weights and bias values of a synthetic neuron are automatically modified to support learning. The most popular model for data processing and software estimation is the neural network. Because it can learn from any dataset, it is possible to obtain pertinent results from it. The general structure of ANNs is composed of the input layer, hidden layer, and output layer. To produce an output, it, therefore, comprises a collection of inputs that are weighted and integrated.

1) BASIC ARCHITECTURE OF PROPOSED BASE MODEL 2 (ANN)

We have used an Artificial Neural Network (ANN) with 100 nodes and Rectified Linear Unit (ReLU) as the activation function. Following is the basic architecture followed to make the Base Model 1.

- Input Layer: This layer receives the input data that the network is supposed to process. The number of input nodes in this layer would depend on the number of features or variables in the input data.
- Hidden Layer(s): This layer is where the actual computation takes place. In this case, the network has one hidden layer with 100 nodes. Each node in this layer takes in the output from the previous layer, performs a linear transformation on it, and applies the ReLU activation function to produce the output. The ReLU activation function is defined as $f(x) = \max(0,x)$, which means that if the input is positive, the output will be equal to the input, and if the input is negative, the output will be zero.
- Output Layer: This layer produces the final output of the network. The number of nodes in this layer would depend on the type of problem the network is trying to solve. For example, if the network is being used for

binary classification, there would be one output node that produces the probability of belonging to the positive class.

During training, the network adjusts its weights and biases to minimize the loss function, which measures how well the network is performing on the task at hand. This is typically done using back-propagation, which is an algorithm for computing the gradients of the loss function with respect to the weights and biases. The gradients are then used to update the weights and biases using an optimization algorithm such as stochastic gradient descent. This process is repeated for multiple epochs until the network's performance on a validation set stops improving.

2) RATIONALITY OF HYPER-PARAMETER SETTINGS IN DEEP LEARNING

The rationality of hyper-parameter settings in deep learning depends on several factors such as the problem at hand, the size of the dataset, the complexity of the model, and the computational resources available. Regarding the use of ReLU as an activation function and 100 nodes in an Artificial Neural Network (ANN), these are common choices that have been shown to work well in many applications. ReLU is a popular choice for an activation function because it has been shown to be effective in addressing the vanishing gradient problem and allows for faster training compared to other activation functions like sigmoid or tanh. Additionally, ReLU has a sparsity-inducing effect which can lead to better generalization performance. The choice of 100 nodes in an ANN depends on the complexity of the problem and the size of the dataset. If the problem is relatively simple or the dataset is small, then 100 nodes may be more than enough to capture the underlying patterns in the data. However, if the problem is more complex or the dataset is large, then a larger number of nodes may be necessary to capture the complexity of the problem.

3) MULTILAYER PERCEPTRON (MLP)

We have employed a feed-forward artificial neural network model called a multi-layered perceptron, which has a single input layer, at least one hidden layer, and a single output layer, as shown in Figure 4. An input vector is represented by each neuron in the input layer. A network is known as a perceptron if it just contains an input layer and an output layer (no hidden layers). In the neurons of the buried layer of an MLP network, a nonlinear activation function is frequently used. It is common practice to use a linear activation function to generate contrast in the output layer. The number of input neurons and training method employed determine, how many neurons are in the hidden layer. The back-propagation algorithm, a kind of gradient descent, is one of the most widely used training methods. The conjugate gradient approach is an alternative way of training an MLP network. One of the most popular neural network models in SDEE is the MLP network. Using performance evaluation criteria like the MMRE, these MLP models are compared to multiple linear regression models [46]. One of the most popular neural network models in SDEE is the MLP network. Using performance evaluation criteria like MMRE, these MLP models are compared to multiple linear regression models for analysis.

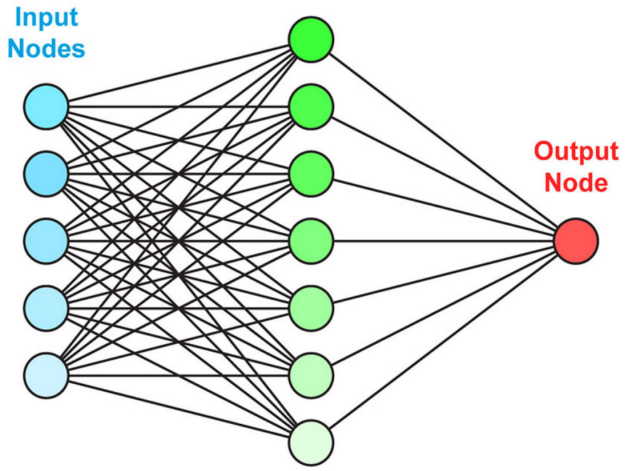


FIGURE 4. Multilayer Perceptron Model.

a: ACTIVATION FUNCTION

The activation function that will be applied to the neurons in the different layers of the neural network must be carefully chosen. The neural network model gains non-linearity from activation functions, enabling the network to accumulate more accurate feature representations over time. In the literature, a variety of activation mechanisms have been described. The most popular activation functions are linear, sigmoid, tanh, and ReLU, and they are typically selected empirically rather than using a strict data-driven methodology throughout the network building phase [51]. A neural network's output, such as yes or no, is decided by t. The values are changed from 0 to 1, −1 to 1, and so forth (depending upon the function). There are two groups that the activation functions fall under. An example of an activation function that is linear in Activation Functions That Aren't Linear is the Linear Activation Function. We have employed ReLU as our activation function.

b: ReLU

ReLU is a simple, non-linear, and efficient activation function that helps in addressing some of the common issues in neural networks. Its effectiveness in improving the learning process and accuracy of neural networks has made it a popular choice in many applications. Rectified Linear Units, or ReLU. A two-hidden layer feedforward neural network with ReLU is NP-hard to train, according to Agarap [52], [53]. ReLU is a common activation function that is used in a variety of contexts. Despite being widely used, the issue of how challenging it is to train a multi-layer fully-connected ReLU neural network has not been settled. The ReLU

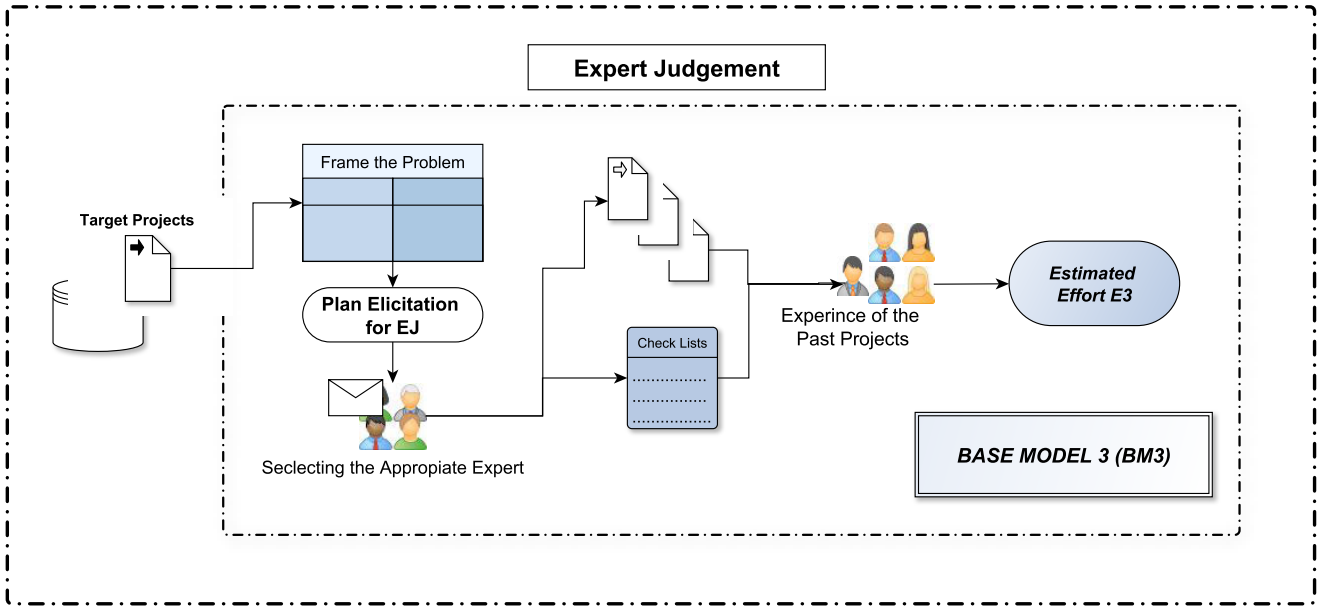


FIGURE 5. Expert Judgement Base Model 3 (BM3).

activation function was developed by Agarap [52], and it is founded on sound biological and mathematical concepts. It was shown to be helpful in deep neural network training in 2011. By setting the threshold to zero, f(x) = max, it operates (0, x). Succinctly summarized, when x is zero, it outputs 0; when x is more than zero, it outputs a linear function.

$$f\ (x) = \max\ (0, x)$$

Rectified Linear Unit (ReLU) is a popular activation function used in Artificial Neural Networks (ANN) and has gained popularity due to its effectiveness in improving the learning process and accuracy of neural networks. Here are some reasons why ReLU is preferred over other activation functions:

- Simplicity: ReLU is a simple function, and its implementation is straightforward. The function is linear for all positive input values, which means that its computation is fast and efficient.
- Non-linearity: ReLU is a non-linear function, and it can help to model complex non-linear relationships in the data. Non-linear activation functions are essential in neural networks because they allow the network to learn non-linear decision boundaries.
- Sparsity: ReLU has a sparsity property, which means that some of the neurons in the network may output zero values. This sparsity can help in reducing the number of parameters in the model and also help in preventing overfitting.
- Gradient vanishing: ReLU helps in addressing the gradient vanishing problem, which is a common issue in deep neural networks. The gradient vanishing problem occurs when the gradient of the cost function becomes very small as it is propagated through multiple layers. ReLU has a constant gradient of 1 for all positive input values, which helps to maintain the magnitude of the gradient during backpropagation.
- Large learning rates: ReLU allows for the use of large learning rates, which speeds up the learning process, as it has a constant gradient of 1 for all positive inputs.
- Simplicity: ReLU is a simple function that requires only one mathematical operation, which makes it easy to implement and optimize.
- Empirical performance: Empirical studies have shown that ReLU generally performs better than other activation functions in deep neural networks for various tasks, such as image recognition, speech recognition, and natural language processing.

The reasons why we use ReLU as an activation due to its popularity as an activation function used in ANN as compared to other activation functions are its ability to introduce sparsity, computational efficiency, non-linearity, ability to avoid vanishing gradient problem, ability to use large learning rates, simplicity, and empirical performance. ReLU has been shown to perform better than other activation functions such as sigmoid and tanh in many applications. This is because ReLU can help in faster convergence of the network.

L. EXPERT JUDGEMENT (EJ)

The most popular methods for predicting software development effort are those that rely on expert judgment and it is our Base Model(BM3). Experts engaged in expert estimating use their knowledge and prior experience to subjectively analyze a range of elements to estimate the development job. The fifth edition of A Guide to the Project Management Body of Knowledge (PMBOK® Guide) by far lists expert judgment as the tool or approach the most frequently. According to Jorgensen [54], professionals use

intuition as a ''non-explicit and non-recoverable reasoning process'' to reach this judgment. Jorgensen [13] suggests employing a variety of approaches when producing estimates for expert-based estimating. The process of expert estimating is straightforward and doesn't require a ton of documentation. COCOMO and other algorithmic estimate techniques pale in comparison. These qualities of expert estimation are aligned with agile processes, which give more importance to interactions between people than to technologies and processes. Without formalism, an expert estimation can have several problems; without a defined structure, an expert estimation can become haphazard. Experts may ignore crucial jobs and activities (such as testing efforts or non-functional needs), which leads to an overestimation of development time. We chose Expert Judgment(EJ) as the foundation model for our ensemble estimation model for a variety of reasons. The findings of empirical investigations contrasting estimates of expert and model-based software development efforts appear to be significant, according to Jorgensen [54]. It is generally impossible to determine whether estimate models or expert estimation is more accurate. Expert estimates, on the other hand, seem to be more accurate when key information is left out of the estimation models, when there is a high level of estimation uncertainty because of unaccounted-for environmental changes, or when straightforward estimation techniques result in reasonably accurate estimates. For this study, we have employed eight industry experts from different software organizations as shown in Figure 5. We are not disclosing the identity of personnel or organizations' names due to ethical reasons. To make sure that the right experts are available for the software company's effort calculation, the experts' demographic data is collected and examined using a checklist fill-in approach. It was difficult to approach the specialists because of many organizational laws and regulations. The software requirement specification (SRS) document, progress report (used for a real amount of effort), software design document, case selection, UCP size, and checklist were all used by the experts. Figure 6 presents our proposed ensemble model. We are going to predict the efforts of the target projects. As one can observe, each base model (BM1, BM2, and BM3) yields their respective efforts and then these efforts are combined using the rules to predict the combined results.

VII. HETEROGENEOUS ENSEMBLE MODEL

We have combined all three base models i.e. Base Model 1 (UCP) as in Figure 2, Base Model 2 (ANN) as in Figure 3 and our Base Model 3 (EJ) as in Figure 5. Each Base Model has yielded a separate effort. These efforts are combined using the combination rules as shown in Figure 6 inspired by Figure 1.

VIII. RESULTS AND FINDINGS

A. ML ALGORITHMS AND PERFORMANCE MEASURES

This section is comprised of three different sub-sections which exclusively present Machine Learning predictions on the primary sources and benchmark datasets i.e. ISBSG dataset and industrial case studies.

1) Results of Standalone Models.
2) Results of Ensemble Models (Heterogeneous Ensemble Model and Machine Learning Ensemble Model).
3) Comparative Analysis on Ensemble Approaches with rest of Standalone Algorithms

We have used five algorithms for our experiments i.e. SVR, Linear Regression, k-NN, XGBoost Regressor, and ANN. To estimate accuracy, we have employed accuracy metrics that include MMRE, MAE, MdMRE, MDAE, and PRED(25). Mean Magnitude of Relative Error (MMRE) (also known as mean absolute relative error) currently uses the most reliable and accepted measures, such as MMRE and PRED at power levels of 0.25, 0.50, and 0.75, respectively.

Results of Standalone Models - ISBSG Dataset: We carried out our experiments on ISBSG dataset. For experiment, we have incorporated famous machine leaning algorithms such as Support Vector Regressor (SVR), Linear Regression, K-Nearest Neighbor (k-NN), XGBoost Regressor and Artificial Neural Network (ANN). We are going to use different variation of datasets to avoid biasness in results. For instance we have used ISBSG in following manners:

- Applying all features on ISBSG dataset.
- Using selected features on recommendation of Nassif et al. [46]
- Applying Results of Statistical Feature Selection.

1) RESULTS OF STANDALONE MODELS USING ALL FEATURES SELECTION OF ISBSG DATASET

We have used all of the ISBSG dataset's features in our first experiment. All ML methods have been applied to all ISBSG Dataset features. In comparison to SVR, LR, k-NN, and XGBoost Regressor, ANN have demonstrated promising outcomes.

Models for estimating effort are evaluated using metrics (criteria metrics). Prediction at Level 1 (Pred (l)), Magnitude of Relative Error (MRE), and Mean Magnitude of Relative Error(MMRE) are most commonly used in the literature [10], [55], [56], [57].

The MMRE (Mean Magnitude Relative Error), Pred(), and MAE measurements have been used to assess performance and compare the proposed EEE model to conventional models that have previously been published in the literature [10], [55] [56]. Similar to this, Pred(25) is the main precision indicator used by the MMRE [10]. According to Kemerer [58], MMRE can be overestimated as well as underestimated.

The diagonal line, which indicates the real effort input, can be used to deduce the interpretation of Figure 10 of All Feature ISBSG. Since the real input value is placed on x and y, i.e., x=y, we obtain the red line in diagonal form.

Now, we can see that scatter plots have developed in Figure 10, of All Feature ISBSG. The values of the actual

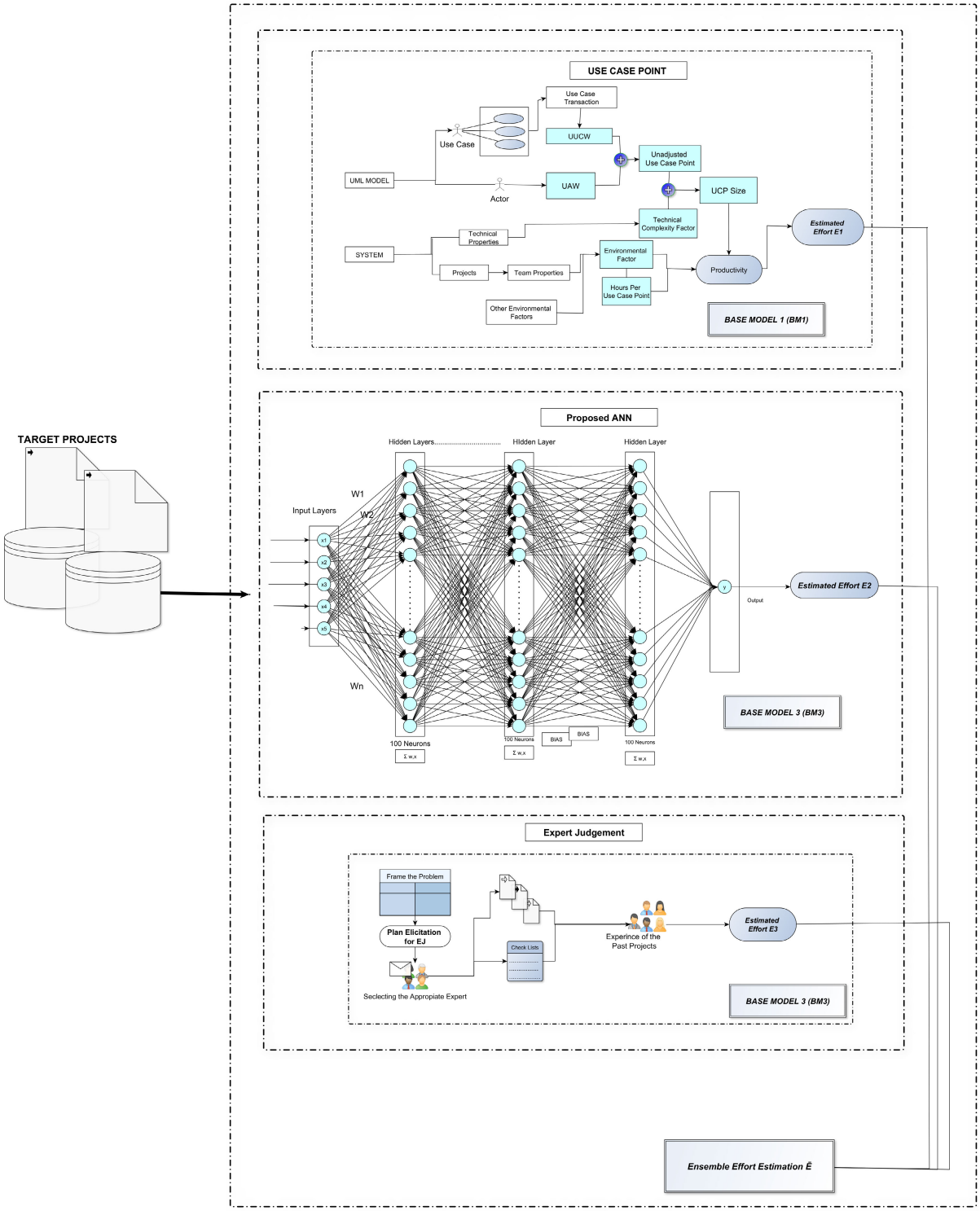


FIGURE 6. Hetrogeneous ensemble effort estimation model.

and expected effort are used to construct the data points on the scattered plot. For instance, the values on the x-axis correspond to the actual effort.

The anticipated effort datapoints are shown on the y-axis.

In contrast to the other ML models, ANN clearly lies on the line i.e. close to the actual effort, as seen in Figure 10, of All Feature ISBSG. According to the graph, ANN has produced excellent results and outperformed other ML algorithms. k-NN Regressor have shown good results as compared

TABLE 10. Standalone ML Model All Feature ISBSG.

	Training time(s)	MMRE	MAE	MdMRE	MdAE	PRED(25)
SVR	7387.36	0.35	476.01	0.080	155.45	0.8014
Linear Regression	145.89	1.28	969.66	0.217	342.62	0.54
k-NN	0.095	0.49	684.96	0.0915	144.33	0.786
XGBoost Regressor	10.73	0.745	985.57	0.102	189.86	0.782
ANN	436.409	0.06	302.78	0.0298	21.72	0.948

to Linear Regressor (LR) and XG-Boost. However, k-NN Regressor datasets fail to produce good results as compared to SVR and ANN.

In addition to the findings, the training time had a significant impact on the ML algorithm's effectiveness. SVR has produced positive results, but this ML model took the longest time to run the experiment. Compared to other ML algorithms, SVR has more than 7387.36 seconds (to be precisely), as shown in Figure 8. On the contrary ANN, which have shown the best results was nearly took half time i.e. 436.409 seconds, to produce the best results among rest of the ML Algorithms. k-NN produce the results with the least time taken i.e. 0.095 seconds and the results were the third best among the other ML Algorithm. Other model such as XG-Boost, Linear Regressor completed their experiments in 145.89 and 10.73 seconds respectively.

The precision of a software estimating model's predictions is a crucial consideration. The mean magnitude of relative error (MMRE), MdMRE, MAE, MdAE, and Pred(25) are used to calculate the effort prediction accuracy and compare the performance of multiple ML models. The estimation is more precise if the value of the MMRE is lower. Pred(25) displays the proportion of forecasts with errors that are smaller than 25% of the true value. The estimation would be more accurate the greater the Pred(25) [59]. The mean/median MRE (MMRE/MedMRE) and prediction at level p (Pred(p)), which counts the number of observations when an SDEE technique produced MREs that were fewer than p, are two of the most often used SDEE accuracy metrics. Although we have calculated MAE, MdMRE, and MdAE. However, when it comes to Software Development Effort Estimation (SDEE), MMRE and Pred(25) have been extensively used by the research community.

As we can see in Figure 9, the radar plot clearly shows the MMRE and Pred(25) of all ML algorithms. Typically, 25% is the ideal MMRE target number. This implies that the accuracy of the existing estimation models would normally be less than 25%. Better estimates are produced by software effort estimation models with lower MMRE values than models with larger values [60], [61]. During our experiments ANN have the lowest MMRE i.e. ANN(0.06) followed by SVR(0.35), XG-Boost(0.745), k-NN(0.49) and Linear Regression(1.28). The highest value of MMRE was generated by Linear Regression(1.28), which is not considered as optimal. Better estimates are produced by software effort estimation models with lower MMRE values than models with larger values [62]. On the other hand, the radar plot shows that ANN(0.948) outperformed all the other ML algorithms. The Pred(25) values of the rest of the algorithms are SVR(0.8014), Linear Regression (0.54), k-NN(0.786) and XG-Boost Regressor(0.782). The recommended estimate accuracy indicator is PRED (25). PRED determines the percentage of predictions that are 25% or less of the actual number (25).

The interpretation that a significant estimating error is a sign of poor estimation skills is not always correct. Alternative, competitive, or supplemental variables include things like poor project cost control, difficult development work, and more functionality provided than anticipated.

2) RESULTS OF LITERATURE-BASED FEATURES SELECTION ON ISBSG DATASET

To perform the comparison between different machine learning algorithms, we have done feature selection on the ISBSG dataset based on suggestions presented in the literature [46]. We used the ISBSG Release 11 industrial datasets to fairly compare machine learning models. There are more than 5000 cross-company efforts from all across the world in the ISBSG Release 11 database. The main characteristics and issues with ISBSG datasets are different platforms, programming languages, and software development life cycle models were used to construct ISBSG projects. Different measurements for program size are used. They consist of SLOC, IFPUG, and COSMIC. Each project is given an "A," "B," or "C" grade depending on its quality. According to the ISBSG guideline, projects with a rank other than "A" and "B" should be terminated. There are many rows (projects) with blank data. More than 100 columns (features) are present; some of them, such as the project number and project date, are not linked to the output (software effort). Statistical analysis revealed that although some of the traits are connected, they are statistically insignificant. Only project ranks "A" and "B" were considered for the experiments. The followings are the primary characteristics and problems of ISBSG datasets as identified by Nassif et al. [46], [63]:

1) ISBSG projects were built using a variety of platforms, programming languages, and software development life cycle models.

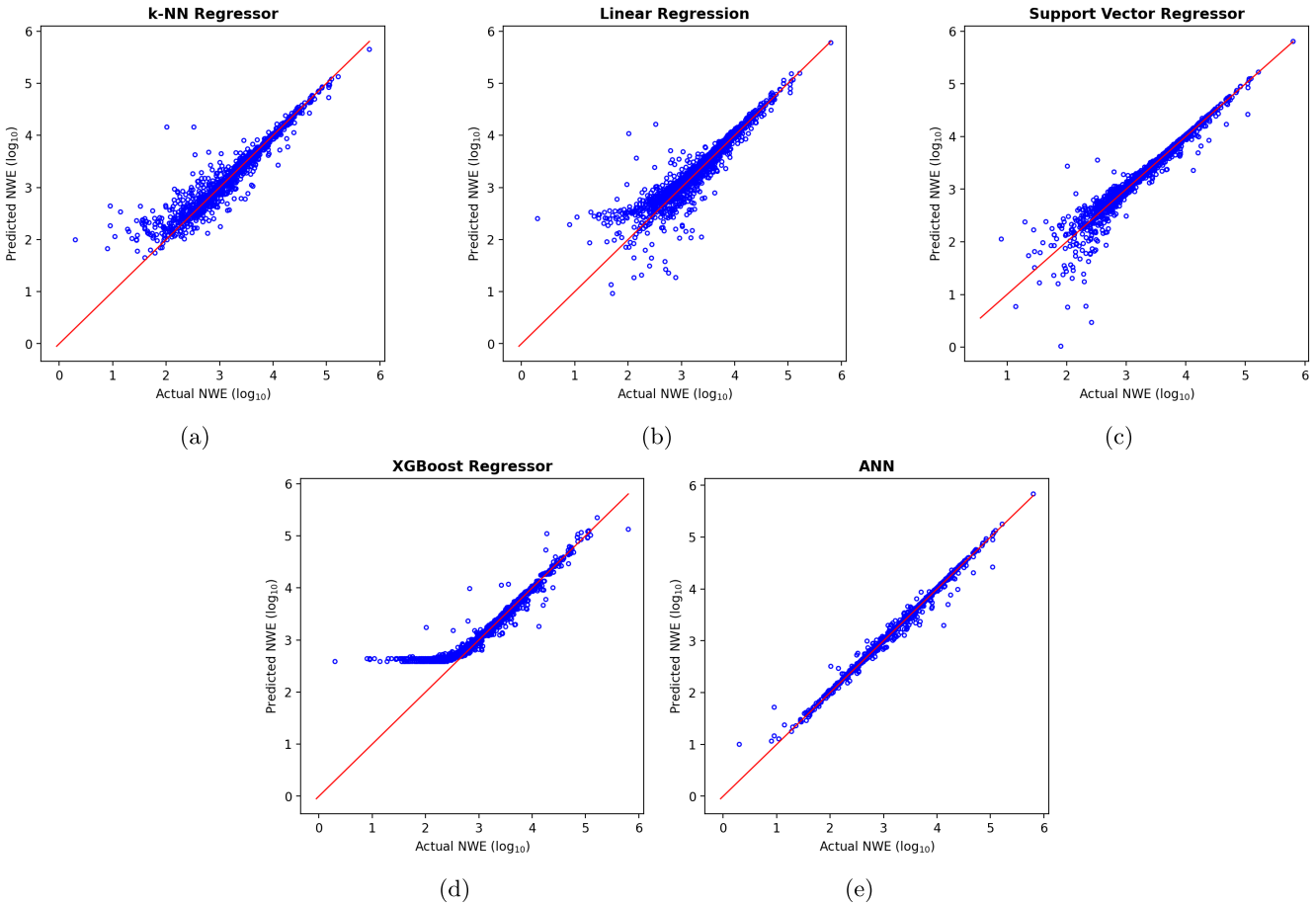


FIGURE 7. Results of all features ISBSG dataset.

TABLE 11. Results of Literature-Based Features Selection on ISBSG Dataset.

	Training time(s)	MMRE	MAE	MdMRE	MdAE	PRED(25)
SVR	0.47	1.01	3716.99	0.38	1123.19	0.37
Linear Regression	0.0069	1.1509	3503.81	0.408	1479.13	0.34
k-NN	0.0085	1.73	4462.60	0.46	1500.33	0.29
XGBoost Regressor	0.035	0.53	838.11	0.127	382.50	0.74
ANN	20.53	0.22	1950.16	0.051	159.21	0.74

2) There are several different metrics used to gauge program size. Some of these are COSMIC, SLOC, and IFPUG.
3) A, B, or C grades are assigned to each project based on its calibre. Projects with ranks other than "A" and "B" are advised to be cancelled, according to the ISBSG advice.
4) Numerous rows (projects) have empty values. Usually, such an occurrence can be replaced by different techniques as suggested by the literature.
5) More than 100 columns (features) are present; some of them, such as the project number and project date, are not linked to the output (software effort). Statistical analyses show that although some of the traits are connected, they are statistically insignificant.
6) In a new version of ISBSG 11, two types of development are new development and enhancement.
7) Even when utilising the same size metric, productivity (the ratio between software work and size) varies substantially because of the dataset's tremendous variety. The productivity, for projects of metric size IFPUG, for

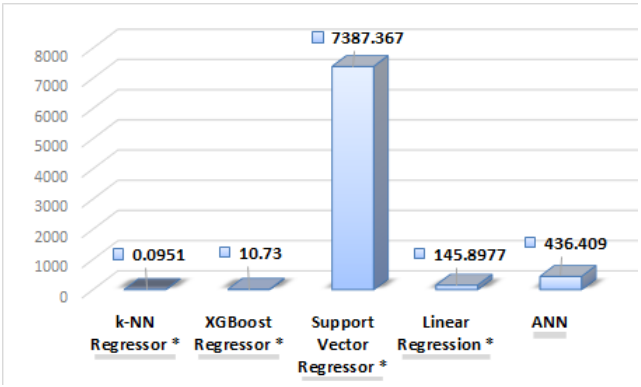


FIGURE 8. Training Time All Features ISBSG (Standalone ML-Model).

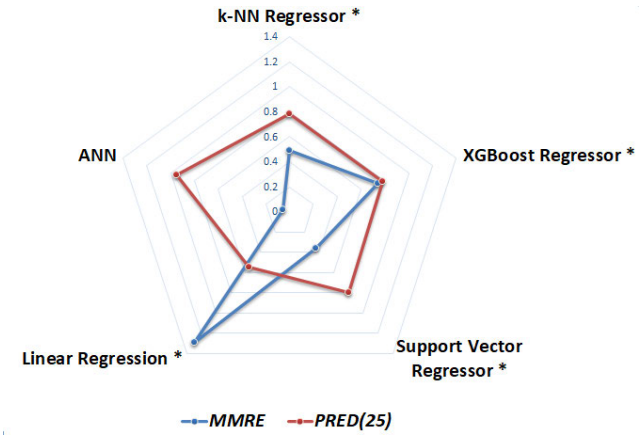


FIGURE 9. RADAR Graph: Results for all ISBSG Features Standalone Algorithm.

instance, ranges from 0.1 to 621. For instance, the time required to produce a project of 100 units can vary from 10 hours (assuming productivity of 0.1) to 6210 hours (if productivity is 621). This is a significant problem that requires attention.

We developed a scalable approach to filter the ISBSG Release 11 and generate five subsets based on the aforementioned ISBSG features (datasets). The ISBSG's suggestions were used to divide the general population into five subgroups. The first things we settled on were IFPUG adjusted function points (AFP), the development type ''new development,'' the development platform, the language type, the resource level, and the normalised work effort. The model's output is the latter (normalised work effort), while its inputs are the other attributes. It is important to note that only the software size (AFP) is a continuous variable among the model's inputs, whilst the others are categorical variables. Only projects with quality ratings of ''A'' or ''B'' and no missing data were taken into consideration, as advised by ISBSG. Based on the productivity value, five different subsets were chosen to address the problem raised in issue 7 above. When the range of the productivity values is from 0 to 4.9 inclusive, the first subset is selected. The second one occurs when productivity levels range from 5 to 9.9, and so forth. Each dataset is split into a training dataset and a testing dataset after five datasets have been prepared.

The projects are arranged in the datasets according to the dates on which they were completed, with the oldest 30% of the projects being utilised for testing and the oldest 70% for training. This strategy is comparable to how previous projects are utilised to prepare current ones and forecast their effort in the real world. This method is reproducible and the splitting is not random, so please be aware that it differs from the random 70/30 per cent stated above.

a: SELECTION ON THE BASIS OF DATA QUALITY RATING

As suggested by literature [46], we filter the dataset based on the data quality. A, B, or C grades are assigned to each project based on its caliber. Projects with ranks other than ''A'' and ''B'' are advised to be canceled, according to the ISBSG advice. We left with 7780 total results.

b: SELECTING FEATURES

As discussed in the literature we have selected features prescribed by the literature. Since these features are the ones that help predict effort. Therefore, we are cleansing our datasets of irrelevant features. We have selected attributes such as Adjusted Function Points (column G), Normalised Work Effort (Column J), Development Type (column AF), Development Platform (column BQ), Language Type (column BR), and Resource Level (Column CU). Please note that Normalised Work Effort is the output of the model (Dependent Variable) where the other attributes are the input of the model (independent variables). Please also note that Development Type will not be input because all projects have the same development type. After this selection, we have 6015 rows and 6 columns.

c: DROPING MISSING VALUES

Missing values always create problems in terms of analysing the dataset. These values has no meaning therefore can have a false impact on the results. After dropping the missing values we have 3432 records left in the dataset.

d: SELECTION OF PRODUCTIVITY ANALYSIS

After dropping values we tend to select the development type. We opted new development type for the productivity analysis.

e: SEPARATING INPUTS AND OUTPUTS

For implementation we need to select the input variables and output variable. For input data we have selected Adjusted Function Points, Development Platform, Language Type, Resource Level and Productivity. Similarly for the output we need only one variable i.e. Normalized Work Effort.

f: TRANSFORMATION OF NOMINAL DATA INTO NUMBERS

Our dataset is consisting of diverse input. For analysis of dataset we need to have similar datatype of all the input i.e. numbers so that our results are more understandable

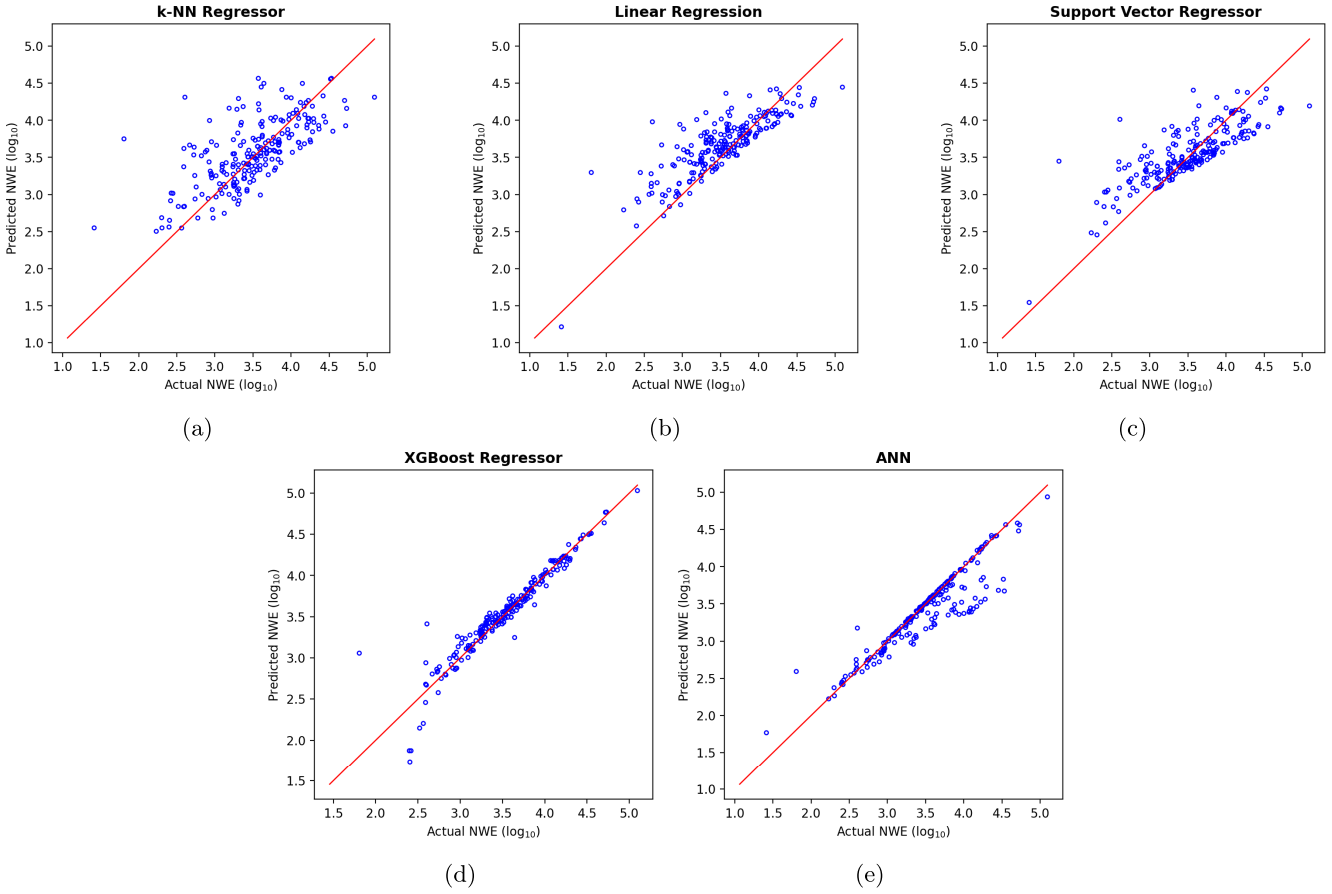


FIGURE 10. Results of ISBSG Dataset Literature-Based Feature selection.

and meaningful. Use the Nominal to Numerical operator to change non-numeric properties into a numeric kind. In addition to changing the type of the attributes, this operator changes all values for the selected attributes to numeric values. The values of a binary attribute are represented by the integers 0 and 1. The numerical input properties of the Example Set remain unchanged. Using this operator, there are three ways to go from nominal to numeric. This mode is managed by the coding type argument.

Filtering is the process in which we make our data more meaningful. For instance, we have eliminate the tuples which are meaningless. Similarly, the empty column were replaced by the mean of the column.

All ML techniques have been applied to every feature of the ISBSG Dataset. If ANN is compared against SVR, LR, *k*-NN, and XGBoost Regressor, it has showed good outcomes. The diagonal line, which represents the actual effort input, can be used to infer ISBSG. Due to the fact that the true input value is centred on x and y, i.e., x=y, we obtain the red line in diagonal form.

This particular feature selected was based on the literature-based as shown in Figure 10. The datapoints on the scattered plot are built using the values of the actual and predicted effort. The values on the x-axis, for instance,

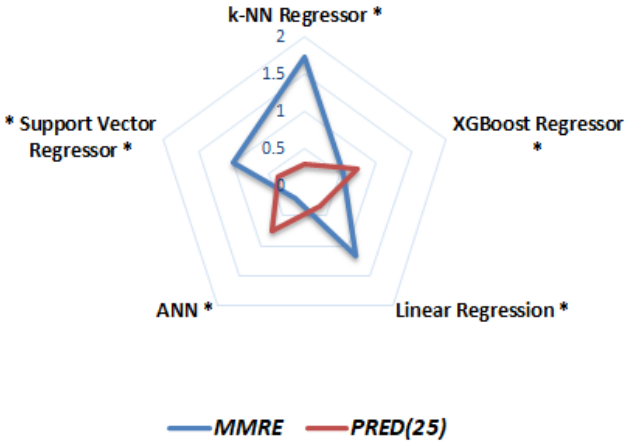


FIGURE 11. RADAR Graph: Results for All ISBSG Features Standalone Algorithm.

represent the real effort. The y-axis displays the datapoints representing the expected effort.

ANN produce better result in terms of scattered plaot in Figure 10. Majority of ANN datapoints lies on the red diagonal line. The x and y represent the values of Actual Effort (x) and Predicted Effort (y). However, other ML algorithm

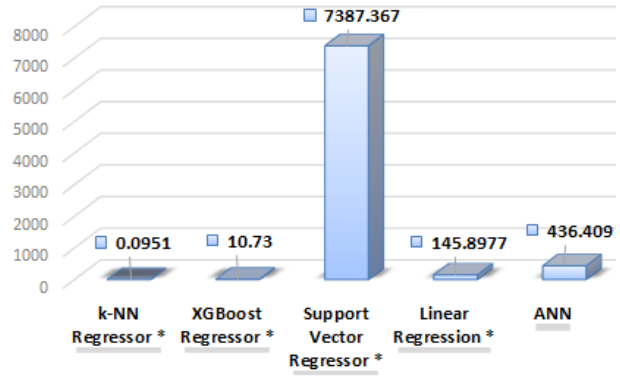


FIGURE 12. Training Time Literature-Based Features Selection on ISBSG (Standalone ML-Model).

were average in terms of lying on the diagonal line nearer to the actual effort. XG-Boost have shown good results even the datapoints were closer to the diagonal line. However, k-NN, Linear Regression and SVR datapoints/resulted scattered plot were not impressive as of ANN and XG-Boost. Therefore, in literature-based feature selection ANN and XG-Boost have performed well as compared to k-NN, Linear Regression and SVR

The fact that MMRE is frequently used in scenarios involving software effort estimation may be due to the fact that people believe it to be the mean absolute percentage error. Since MMRE is scale-free, estimation errors from all sizes of software development projects can be combined using it. The MMRE has no upper score limits for overestimation, but an underestimating of effort can never result in a score higher than one. Despite this scoring asymmetry restriction, MMRE (MAPE) may be the most widely used estimation error metric in research and industry [54].

The MMRE of ANN(0.22) was more better than the other ML algorithms as shown in Figure 11 XG-Boost (0.53) stands second best, SVR (1.01) third best, Linear Regression (1.15) fourth best and finally k-NN (1.73). These results show that ANN outperformed other ML algorithm when it comes to MMRE analysis. ANN also produce great results in terms of PRED(25). Comparison of ANN with other ML techniques with respect to MMRE and PRED(25) can be observed un RADAR plot in Figure 11 The higher the Pred value the better model is considered. ANN and XG-Boost were better in terms of PRED(25). While k-NN ranked second. Linear Regression produce better results as compared to SVR which has the accuracy of (0.37). Higher PRED(25) shows that its derived estimates are more accurate than other models.

Figure 12, contrasts the execution times of several models (in s). Compared to other algorithms, the ANN algorithms' training process takes a long time (SVR,LR, k-NN and XG-Boost). We can see that less complex algorithms take less time to train than simple ones like Linear Regression (LR) and k-NN (values can be interpreted from Table). When combining the execution times for training and testing, LR is one of the quickest algorithms.

3) RESULTS OF STATISTICAL BASED FEATURES SELECTION ON ISBSG DATASET

a: F- REGRESSION

Whether a characteristic is chosen manually or automatically depends on how important it is to prediction variable or desired result. Model may learn based on irrelevant features in the given data, which could result in a reduction in model accuracy.

Following is the equation of F- Regression Model. F-Regression ranks features based on its weight(importance).

Equation of F- Regression ranks features:

$$FR_{Feature} = \{\langle f_1, score\rangle, \langle f_2, score\rangle, \ldots., \langle f_n, score\rangle\} \quad (16)$$

Sort with respect to the score:

$$FR_{Sorting} = \{\langle f_1', score\rangle, \langle f_2', score\rangle, \ldots., \langle f_n', score\rangle\} \quad (17)$$

such that:

$$f_1' : score < f_2' : score, f_2' : score < f_2' : score, \ldots\ldots., f_n' : score < f_n' : score \quad (18)$$

Finally:

$$FR_{Top_25_Feature} = FR_{Sorted}[-25:] \quad (19)$$

The stages of feature extraction and selection are now required for "low loss dimension reduction." These fields—machine learning, data mining, and pattern recognition—all make use of this paradigm. In machine learning, a set of pertinent target features must be preprocessed, and to reduce dimensionality, the most suitable feature subset must be chosen for the classification task. As we have already covered the literature-based feature selection and all features selections. Now we are moving toward to our third feature selection method. This methods is being chosen on the basis of its results.

The choice of filter features typically depends on statistical assessments of correlation between input and output variables. Therefore, the types of variable data have a significant impact on the choice of statistical measures. The more details that are known about the data type of a variable, the simpler it is to choose an appropriate statistical measure for a filter-based feature selection strategy. Input and output are the two main categories of variables to take into account, as well as the two main types of variables: categorical and numerical. Input variables are the variables that are utilised as inputs in a model. When selecting features, we want to use as few of these variables as possible.

Response variables are typically used to refer to the variables that a model is meant to predict, also known as the output variables. The type of response variable typically determines the specific predictive modelling task under consideration. A problem with regression predictive modelling is indicated by a numerical output variable, for example, and a problem with classification predictive modelling by a categorical output variable. Typically, the target variable is utilised to construct the statistical measures used in

TABLE 12. Statistical based Feature Selection.

	Training time(s)	MMRE	MAE	MdMRE	MdAE	PRED(25)
SVR	3.55	0.15	445.65	0.059	62.18	0.80
Linear Regression	0.14	1.28	969.186	0.216	343.96	0.54
k-NN	0.0046	0.276	601.53	0.055	85.16	0.86
XGBoost Regressor	0.515	0.513	885.17	0.085	159.02	0.814
ANN	161.36	0.06	291.18	0.031	62.75	0.953

filter-based feature selection one input variable at a time. They are known as univariate statistical measures as a result. This could imply that the filtering procedure does not take into account any input variable interactions.

As mentioned in the previous two section the formation of the diagonal line in Figure 13, depend on the input of actual effort which is used to plot the diagonal line. ISBSG may be deduced from the diagonal line, which depicts the actual effort input. We get the red line in diagonal form because the true input value is centred on x and y, i.e., x=y. As we have previously observed in All features selection and Literature-based features selection, ANN has again outperformed other ML algorithm in statistical based features selection. The datapoints of ANN stick on the actual effort diagonal line which means that ANN predicted result were closer to the actual effort as compared to other ML algorithms. k-NN and SVR results were better after ANN results. Linear Regressor results were disperse across the actual effort diagonal line. XG-Boost few datapoints were scattered away from the actual effort. However, rest of the datapoints lie on the actual efforts diagonal line.

Training time consider to be an important aspect when we compare different Ml algorithm. Some models tends to have long training time. On the contrary, some models execute the experiment within no time. Training time is also plays a significant role in decisive of the optimal model. However, there are other factors which also constitute to make a model an optimal solution. As Figure 14 indicate that ANN has taken the most time to run its experiment. During our statistical based Feature Selection, ANN consume the maximum time to train its model on our dataset i.e. 161.36. However, ANN produced the best statistical based feature selection results as compared to other algorithm but it also consume the maximum training time among all of the models. Although the result on the scattered plot of ANN was far better than the other ML Algorithm. Other Algorithm which include SVR (3.55s), Linear Regression (0.14s), XG-Boost(0.515).k-NN (0.0046s) has taken the least time to execute statistical based feature selection experiment.

Radar plot in Figure 15 clearly shows that MMRE of ANN(0.06) was the least among all the other ML algorithms. Other algorithms such as SVR tends to have (0.15) MMRE and position second as the lowest MMRE. The model have the least MMRE are considered to be the best. Apart from that k-NN prove to the third best producing (0.276) MMRE. However, XG-Boost produce (0.513) MMRE forth best and Linear Regression has produced the worst among all, producing (1.28) MMRE.

B. RESULT OF STANDALONE ALGORITHMS ON CASE STUDIES

We have gathered eight case studies from the software industry. These initiatives are genuine industry projects. In this section, we will try to predict the effort of our collected case studies using our trained machine learning algorithms (SVR, LR, k-NN, XGB, and ANN). As was mentioned in the previous section, our ML models were created utilizing ISBSG datasets, and they have shown some positive results. We will now use the pickled model to anticipate case studies. The objective is to assess the error between the amount of effort used in the case studies and the amount of effort estimated by our trained ML models.

1) RESULTS: ML ALGORITHMS ON CASE STUDIES (ALL FEATURES)

Results from the Table 13, according to All Features, ML techniques were applied to the case studies, and the associated effort was calculated. The quantitative measurements we employed to quantify our experiment were MMRE, MAE, MdMRE, MdAE, and PRED (25). Pred(25) determines the proportion of an estimate that is within a 25% range of the true value. Pred(25) provides the closest and frequently most accurate result as a result. A good estimation model tries to increase PRED while attempting to decrease MMRE (25). A combination of MMRE and PRED, the evaluation function (EF) might be the third evaluating component (25).

The result in Table 13 shows that XGBoost and ANN yield to produce better results with respect to Pred(25) i.e. 1. However, SVR was not far behind with 0.875. k-NN and Linear Regression tend to produce 0.5. On the contrary, with respect to MMRE ANN lead all the other ML techniques by producing 0.0022. On the other hand, SVR(0.07) was the second, however LR(0.30) and k-NN failed to produce good results.

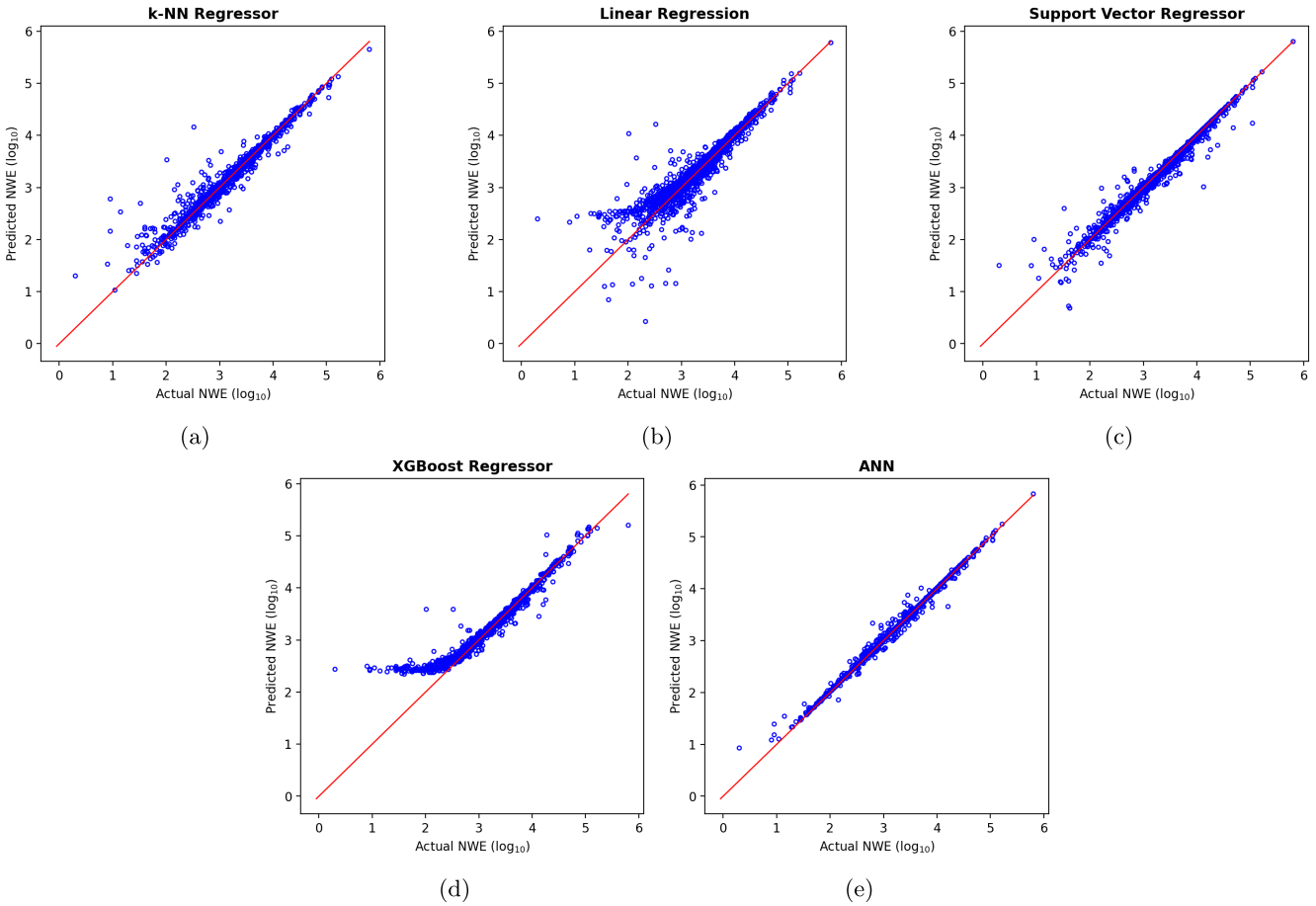


FIGURE 13. Results of ISBSG dataset statistical based features selection.

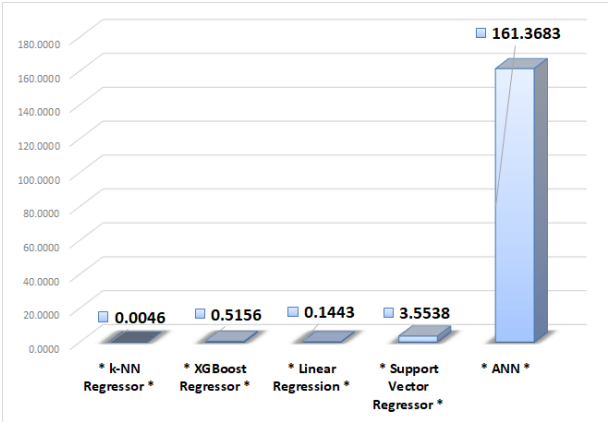


FIGURE 14. Training Time Statistical Based Features Selection on ISBSG (Standalone ML-Model).

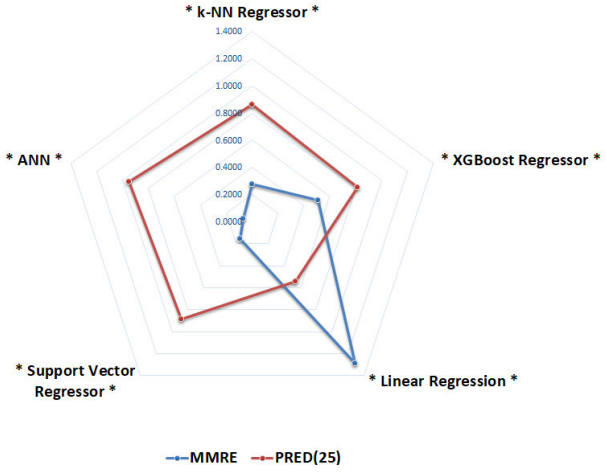


FIGURE 15. RADAR Graph: Results for Statistical Based Features Selection (Standalone Model) Algorithm.

2) RESULTS: ML ALGORITHMS ON CASE STUDIES LITERATURE-BASED FEATURES SELECTION

Similarly, we have run the ML trained algorithm(precisely on the literature based learning variation). As mentioned earlier, the literature based feature selection was done to avoid irrelevant data from the dataset to produce good results.

3) RESULTS: ML ALGORITHMS ON CASE STUDIES BASED ON F-REGRESSION FEATURES SELECTION

Following the procedure which we have obtained in the previous two subsections, we have applied our trained ML

TABLE 13. Results: ML Algorithms on Case Studies (All Features).

	ISBSG_ALL_k-NN	ISBSG_ALL_LR	ISBSG_ALL_SVR	ISBSG_ALL_XGB	ISBSG_ALL_ANN
CS1	2682.3333	3097.0452	3242.45	3080.30	2973.610
CS2	899	1642.069	1588.8743	1644.6914	1541.8499
CS3	2700.6666	515.9201	827.3412	1497.7968	1361.0531
CS4	1690	489.5411	1349.5646	1661.9857	1446.5652
CS5	899	1535.6108	1380.7577	1314.8732	1250.3455
CS6	1964	938.5012	1086.6939	1474.5128	1456.4803
CS7	3449	5179.1144	4610.9200	3963.8464	4217.909
CS8	3064.3333	2238.4011	2786.9095	3150.7168	3376.0957
MMRE	0.34049	0.3092	0.1503	0.0734	0.02208992
MAE	591.0	603.5421	304.46838	149.6523	36.236023
MdMRE	0.2313	0.2704	0.0991	0.06601	0.0023
MdAE	475.0	634.289	275.3794	125.05096	3.522
PRED(25)	0.5	0.5	0.875	1.0	1.0

TABLE 14. Results: ML Algorithms on Case Studies.

	ISBSG_NASIF_k-NN	ISBSG_NASIF_LR	ISBSG_NASIF_SVR	ISBSG_NASIF_XGB	ISBSG_NASIF_ANN
CS1	1571.666667	143911.6638	46763.43734	4309.826	54586.234
CS2	2472.666667	7475.628471	3936.636449	10121.08	8846.0205
CS3	5326	5404.340488	3412.235908	6524.895	6947.7163
CS4	6279.333333	36851.02618	12292.7161	6732.8193	12731.928
CS5	4695.666667	6711.33265	5239.724358	3155.4863	1606.744
CS6	4024.333333	19924.0852	6313.109243	9251.575	9391.158
CS7	13961.66667	46712.18928	15005.10133	15245.863	33336.645
CS8	8119.333333	73944.71011	24521.86921	10676.566	32851.258
MMRE	1.94728	15.8806	5.079647	3.11088	6.900151
MAE	3941.7916	40402.74702	12471.4787	6038.1389	17823.086
MdMRE	2.12991	11.83807	3.4083	3.1297	6.393572
MdAE	3706.83333	26980.0556	7868.60528	6281.69287109375	9654.043
PRED(25)	0.5	0.5	0.875	1.0	1.0

algorithms on our case studies using F-Regression feature selection.

Previously, in Table 12, ML algorithms tend to produce good results on simple ISBSG dataset. Lets compare these results with the results on Case Studies as mentioned above in Table 15. On both occasion, ANN and SVR produced good result, but on the basis of overall performance ANN have produced best results (PRED - ISBSG = 0.953 and Case Studies = 1) and (MMRE ISBSG = 0.06 and Case Studies = 0.03). Among ML, worst results was produced by Linear Regression (PRED - ISBSG = 0.5 and Case Studies = 0.5) and (MMRE - ISBSG = 1.28 and Case Studies = 1.28).

IX. RESULTS OF HETEROGENEOUS ENSEMBLE MODELS

The acquired information will be applied to the proposed model, and the measurement of the solution will be evaluated to ascertain the anticipated effort. To do a comparative analysis, four machine learning models—Support Vector Regressor (SVR), Linear Regression, K-Nearest Neighbor (k-NN), XGBoost Regressor, and Artificial Neural Network (ANN)—have been introduced.

The estimation of software work is a regression problem in the context of ML. Given a history of correlation between the two variables, the regression algorithms are an equation that seeks to estimate the value of a variable (y) based on one or more independent variables (x). The goal of the function is to create a linear relationship between X and Y so that X's value can be determined from Y's value [64].

A. RESULTS OF USE CASE POINTS FOR EFFORT ESTIMATION (UCP)

1) COMPUTE UNADJUSTED ACTOR WEIGHT

During this phase, the unadjusted actor weight is calculated over eight case studies. Unadjusted Actor Weight is the sum of all actor weights stated in Table 16. Three complex actors, one average actor, and two simple actors are among the 15 unadjusted actor weights in the case study (CS1). Similar results are obtained for CS2, which has 3 average and 2 complicated players, CS3, which has 1 simple and 1 difficult actor, and CS4, which has 2 complex actors, which, respectively, equal to 12, 4, and 6 UAW.

2) ESTIMATE THE UNADJUSTED USE CASE WEIGHT

The unadjusted use case weight at this level was calculated using data from four case studies. The total use case weights listed in Table 17 make up the unadjusted use case weight (UUCW). The calculated unadjusted use case weights for cases (CS1, CS2, CS3, CS5, CS6, CS7, and CS8) are 195, 70, 120, 130, 75, 90, 200, and 130, respectively.

3) ESTIMATION OF THE UNADJUSTED USE CASE POINT (UUCP)

The Unadjusted Use Case Point (UUCP), also known as the Unadjusted Use Case Points (UUCP) given in Table 18, is created by adding the Unadjusted Actor Weight (UAW) and Unadjusted Use-Case Weight (UUCW).

TABLE 15. Results: ML Algorithms on Case Studies - F-Regression.

	ISBSG_STATS_k-NN	ISBSG_STATS_LR	ISBSG_STATS_SVR	ISBSG_STATS_XGB	ISBSG_STATS_ANN
CS1	3113	3024.071459	3219.5	3084.6802	3031.651
CS2	1372.333333	1570.397974	1394.8	1556.179	1601.0916
CS3	773.6666667	443.6380079	980.23	1487.0033	1418.8856
CS4	1921.666667	416.152838	1627.3	1672.8125	1470.8054
CS5	1274.666667	1461.55463	1311.1	1436.6188	1291.2676
CS6	1537	866.2283226	1238.2	1529.8557	1467.7188
CS7	4014.666667	5111.351389	4253.8	4052.6697	4324.029
CS8	3131	2163.557731	2857.9	3324.4053	3425.283
MMRE	0.15897	0.32437	0.12212	0.0882	0.03563077
MAE	274.70833	622.25032	235.1922	150.5895	66.44356
MdMRE	0.1020	0.287639	0.1075	0.09194	0.0293
MdAE	233.16666	697.06153	213.39423	157.5756	61.26825
PRED (25)	0.75	0.5	0.875	1.0	1.0

TABLE 16. Unadjusted Actor Weight (UAW).

Case ID	Actor Type and Weight			Number of Actors			Product	UAW
	Simple	Average	Complex	SA	AA	CA		
CS1	1	2	3	2	1	3	1x2 + 2x1 + 3x3	13
CS2	1	2	3	1	0	1	1x1 + 2x0 + 3x1	4
CS3	1	2	3	0	3	2	1x0 + 2x3 + 3x2	12
CS4	1	2	3	1	0	1	1x1 + 2x0 + 3x1	4
CS5	1	2	3	2	0	1	1x2 + 2x0 + 3x1	5
CS6	1	2	3	1	1	0	1x1 + 2x1 + 3x0	3
CS7	1	2	3	3	2	2	1x3 + 2x2 + 3x2	13
CS8	1	2	3	0	0	2	1x0 + 2x0 + 3x2	6

TABLE 17. Use Case Weight (UUCW).

Case ID	Use Case Type and Weight			Number of Use Cases			Product	UUCW
	Simple	Average	Complex	SUC	AUC	CUC		
CS1	5	10	15	23	5		5x23 + 10x5 + 15x2	195
CS2	5	10	15	2	5	4	5x2 + 10x5 + 15x4	120
CS3	5	10	15	2	3	2	5x2 + 10x3 + 15x2	70
CS4	5	10	15	3	2	2	5x3 + 10x2 + 15x2	65
CS5	5	10	15	3	3	2	5x3 + 10x3 + 15x2	75
CS6	5	10	15	4	2	2	5x4 + 10x2 + 15x2	90
CS7	5	10	15	18	6	2	5x18 + 10x6 + 15x2	200
CS8	5	10	15	3	7	3	5x3 + 10x7 + 15x3	130

TABLE 18. Unadjusted Use Case Point (UUCP).

Case ID	UAW	UUCW	UUCP = UAW + UUCW
CS1	13	195	208
CS2	4	120	124
CS3	12	70	82
CS4	4	65	69
CS5	5	75	80
CS6	3	90	93
CS7	13	200	213
CS8	6	130	136

4) COMPUTING TECHNICAL COMPLEXITY FACTORS

The technical complexity factors, which have values ranging from 0 (completely inconsequential) to 5, characterize the non-functional project needs (very relevant). The author provided these components to the creators of the chosen instances in order for them to achieve the appropriate value (RV) against 13 technical characteristics. It is crucial to remember that separate teams built the target instances with various requirements and scopes, which results in a range of associated values. The values shown in Table and Figure were created by the developers of the selected instances using their prior development experience. The technical factors

(T1-T13) are multiplied by their corresponding weights to produce the technical factor (TFactor), which is then calculated by adding up all the resulting values. Equation 12 is used to compute the TCF. The TCF for (CS1, CS2, CS3, CS5, CS6, CS7, and CS8) is, respectively,.89,.93,.95, and.92.

5) COMPUTING ENVIRONMENTAL COMPLEXITY FACTORS

The effects of environmental circumstances on production are calculated using eight environmental parameters (E1-E8). According to these criteria, the initiatives are graded from value (completely unimportant) to 5. (very relevant). This study employed the expertise of experts from six software sectors to compute the related value (RV) against eight environmental elements in order to prevent bias in the outcomes of these aspects. In addition, four experts who each work in a different area of the software industry provided the values for CS1 and CS2 (expert 1 for CS1 and expert 2 for CS2, respectively). The data are then combined to improve the values' accuracy. However, for CS3 and CS4, the expertise of two professionals working in two separate software businesses is utilized. It is crucial to keep in mind that because different teams generated the target instances, the associated values are diverse. The values presented in Tables 27 and Table 26 were recommended by the experts based on their prior development experiences(Please refer to Appendix A for all the Tables[1]).

The computations for the environmental factor (EFactor) and environmental complexity factor (ECF).[2] Table 19 reveals that the average ECF for CS1, CS2, CS3, CS4, CS5, CS6, CS7, and CS8 is, respectively, 0.755, 0.5525, 0.62, 0.5825, 0.8675, 0.8, 0.83, and 0.8525.5.

a: COMPUTING ADJUSTED USE CASE POINT (AUCP) AND ESTIMATED EFFORT

The size figure for the chosen situations is provided by the AUCP. The productivity factor is a crucial component of estimating software effort because it helps translate size into human work. It is described as a ratio of effort to area. After the corrected UCP has been calculated, productivity is multiplied. The estimated and actual effort of the selected situations are shown in Table 20 and Figure 18.

Karner's first calculation of the number of man hours required for each AUCP was twenty. the notion that environmental factors should be taken into account when calculating the number of work hours needed for each use case point. The number of environmental factors with scores between one and three in categories E1 through E6 is tallied and added to the factors with scores between three and eight in categories E7 through E8, according to Schneider and Winters [65]. if more than two are united. If there are three or four UCPs, use 28 man hours per UCP instead of the typical advice of 20. If there are more than five use case points, it is frequently suggested to consider budget 36 man hours for each one. So, we applied the Schneider and Winters [65] approach to calculate the productivity factor. But using this method results in a productivity factor of 20 man-hours for each use case point for all of the selected examples.

B. EFFORT ESTIMATION USING EXPERT JUDGEMENT

One of the most widely used estimation strategies is effort estimation by expert judgment. Many sectors rely on the expert opinion when estimating a project's cost, the number of people needed, and how the tasks will be distributed among them. The level of effort in the selected situations is estimated using the experts' expertise. The experts' demographic information is obtained and analyzed using a checklist fill-in method to ensure that the appropriate specialists are accessible for the software company' effort calculation. Due to various organizational laws and regulations, approaching the specialists may be a challenging task. Although the expert may be extremely skilled and knowledgeable and able to complete the assignment in a field closely connected to it, accurate estimation may be difficult.[3]

The primary determinants of effort are the target instances' functional requirements. The analysts were able to identify the most likely attempt in each instance using the checklist's A and B. Before estimating the time needed to accomplish the functional needs of the projects, the specialists were aware of the technical and environmental factors[4] present in the target scenarios. For the four case studies, there are a total of 28, 11, 23, and 34 functional criteria, respectively. Different professionals have different levels of prejudice, which are a product of both conscious and unconscious processes. The average of each expert's unique estimations, which are added together to create a single estimate, is taken into consideration to lessen the amount of bias. The time experts estimate it will take to develop the target cases.

X. COMPARATIVE ANALYSIS, RESULTS AND DISCUSSION

A. RESULTS OF BASE MODELS BM1, BM2 AND BM3

We are going to show the outcomes of the UCP, ANN, and EJ in this part. Additionally, the actual outcome will be shown for comparison. Table 22 shows the efforts of BM1, BM2, and BM3 with regard to their case study.

a: RQ1: DOES A HETEROGENEOUS ENSEMBLE EFFORT ESTIMATION MODEL USING UCP, ANN AND EJ PRODUCED BETTER RESULTS AS COMPARED TO STANDALONE ML MODELS?

To answer RQ1, lets first have a brief review of the experiments. Machine learning models were trained using the benchmark datasets ISBSG and UCP. Eight case studies from the software industries were gathered, and these ML methods were then used on them in real-world scenarios. The

[1]Please refer Appendix for Table 25 consist of Technical Complexity factors of selected case studies.

[2]Please refer Appendix section for Table 27 consist of environmental values assigned by experts of case studies.

[3]Please refer Appendix for Table 28 consist of Feedback given by Experts.

[4]Please refer Appendix-A for Table 26 for Environment Complexity Factor.

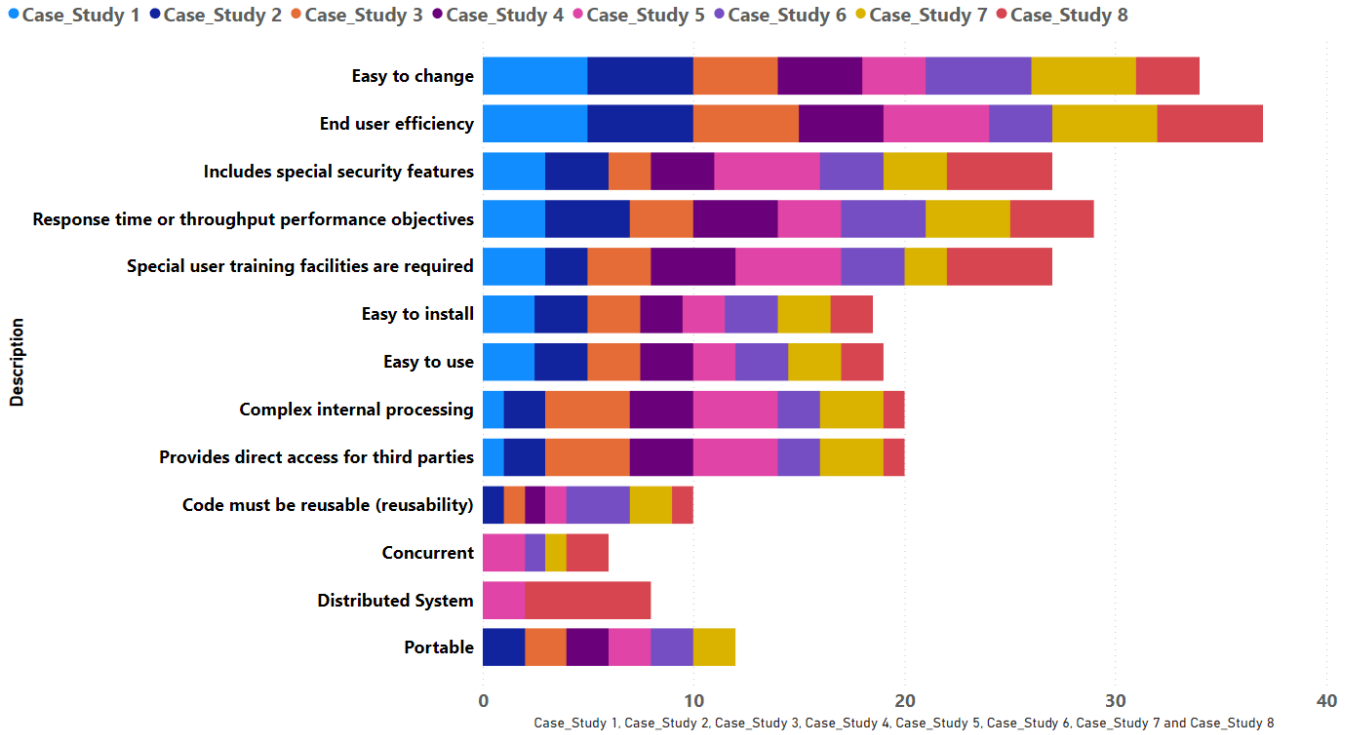


FIGURE 16. Graphical representation TCF.

TABLE 19. Average ECF of Case Studies.

Case ID	EFactor		ECF		Average ECF
	Expert 1	Expert 2	Expert 1	Expert 2	
CS1	21	22	0.77	0.74	0.755
CS2	29.5	27	0.515	0.59	0.5525
CS3	23.5	28.5	0.695	0.545	0.62
CS4	25.5	29	0.635	0.53	0.5825
CS5	17	18.5	0.89	0.845	0.8675
CS6	19.5	20.5	0.815	0.785	0.8
CS7	19	19	0.83	0.83	0.83
CS8	17.5	19	0.875	0.83	0.8525

TABLE 20. Estimated effort using UCP method.

ID	U_{UCP}	TCF	ECF	AUCP	Productivity	Effortest (man-hours) AUCP X Productivity	Actual Effort
CS1	208	0.87	0.755	136.62	20	2732.4	2970
CS2	124	0.93	0.5525	63.71	20	1274.2	1545
CS3	82	0.93	0.62	47.28	20	945.6	1358
CS4	69	0.925	0.5825	37.17	20	743.4	1450
CS5	80	1	0.8675	69.4	20	1388	1250
CS6	93	0.93	0.8	69.192	20	1383.84	1365
CS7	213	0.95	0.83	167.95	20	3359	3259
CS8	136	0.99	0.8525	114.71	20	2294.2	2640

assessment standards for our findings were carried out by PRED. The PRED value with the highest value was selected using the combination rule of the Ensemble model. All of the ML models were applied to the case studies, and the ML models were then integrated by averaging the highest PRED score.

Table 23 compares Heterogeneous Proposed Ensemble with stand-alone algorithms. The Heterogeneous Ensemble outperforms the Standalone Algorithm during our experimentation as one can observed in the cell with the darkest colour, as seen in Table 23. This would be seen as the ensemble's most noticeable improvement over the stand-alone method

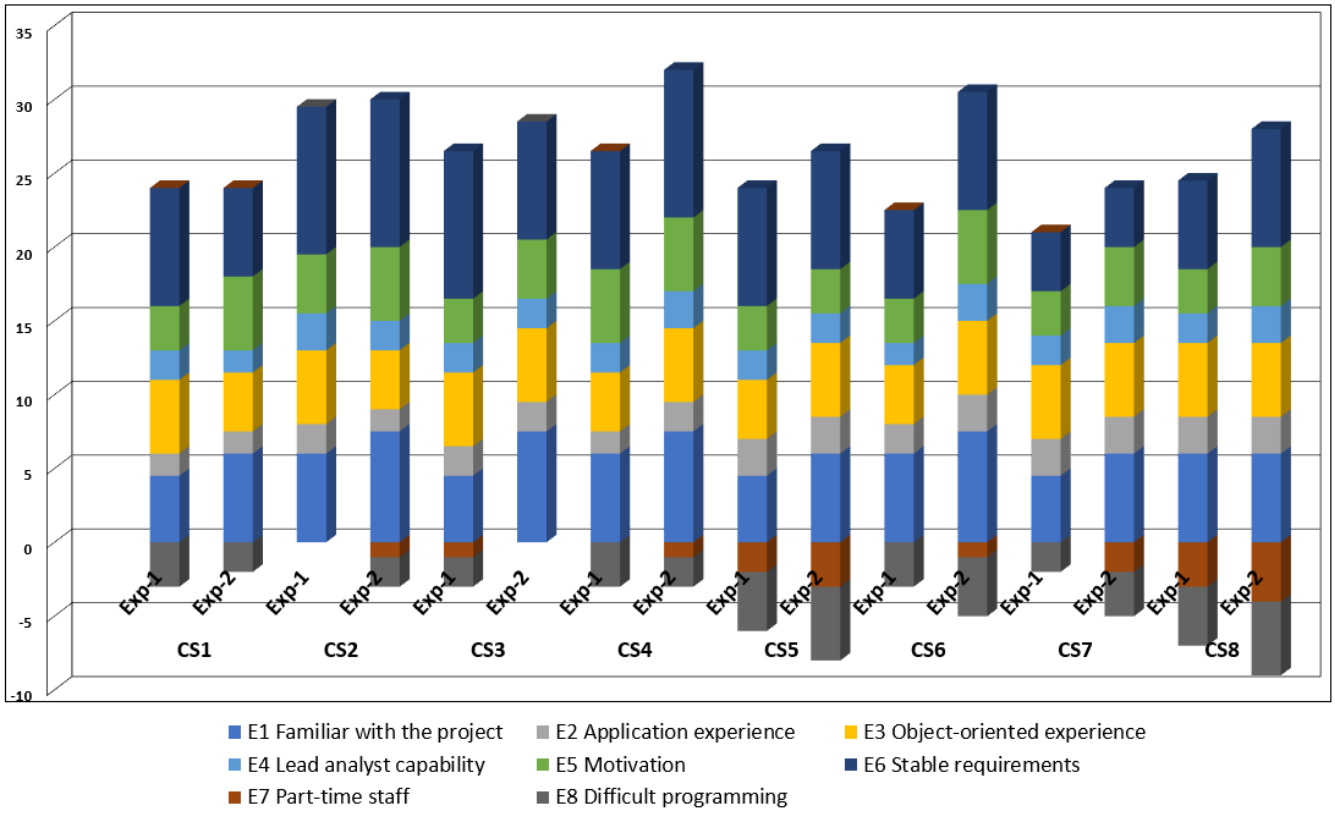


FIGURE 17. Graphical representation ECF.

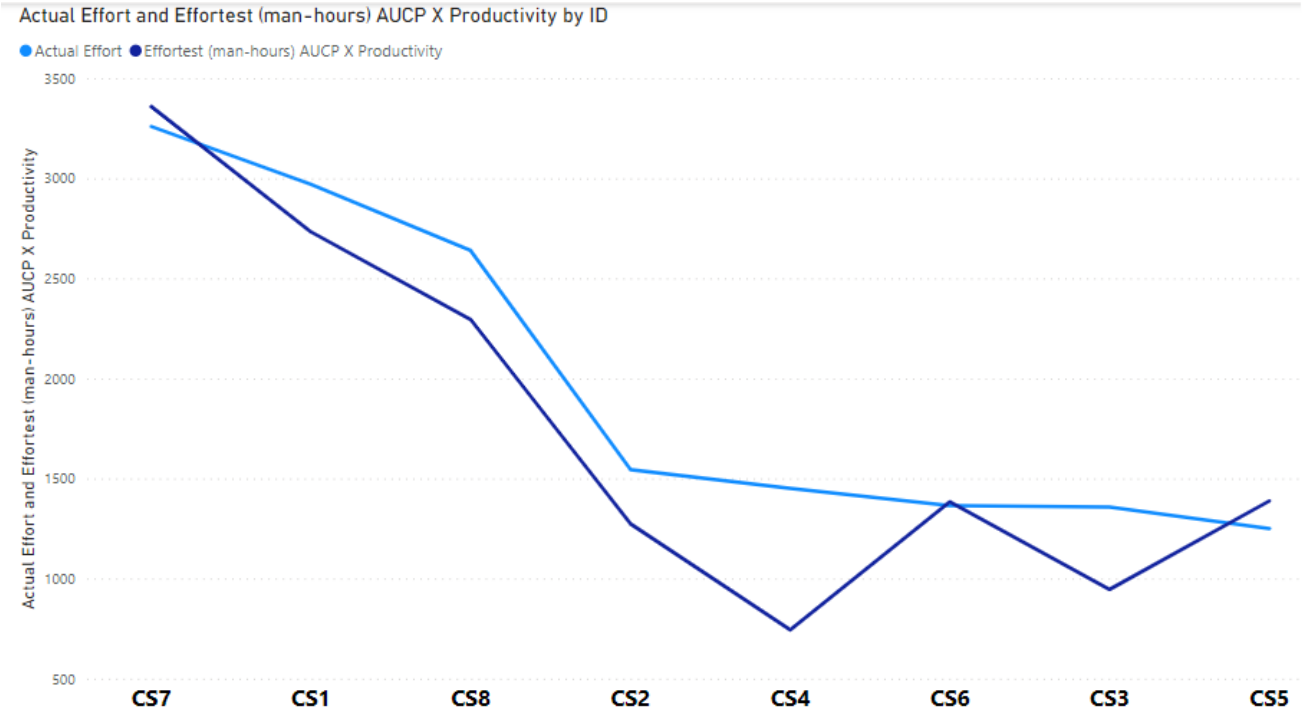


FIGURE 18. Estimated effort using UCP method.

TABLE 21. Results: Expert Judgement - Effort Estimation.

ID	Expert-1	Expert-2	Effort (man-hour)	Actual Effort (man-hour)
CS1	2815	3065	2940	2970
CS2	2105	2215	2160	1545
CS3	1415	1615	1515	1358
CS4	1855	1805	1830	1450
CS5	1120	1090	1105	1250
CS6	1075	1195	1135	1365
CS7	3635	3470	3580	3259
CS8	2895	3050	2972.5	2640

TABLE 22. Hetrogeneous Model Results.

	Actual Effort	UCP	EJ	ANN
CS1	2970	2732.4	2940	2973.6108
CS2	1545	1274.2	2160	1541.8499
CS3	1358	945.6	1515	1361.0531
CS4	1450	743.4	1830	1446.5652
CS5	1250	1388	1105	1250.3455
CS6	1365	1383.84	1135	1456.4803
CS7	3259	3359	3580	4217.909
CS8	2640	2294.2	2972.5	3376.0957

as Heterogeneous Ensemble outperformed XG-Boost Standalone in all areas of the ISBSG dataset by 78%. Second, the findings of the heterogeneous ensemble model exceeded those of the k-NN by a factor of 70.45%. (ISBSG Dataset on Literature-Based Feature Selection). Table 23 shows that the Heterogeneous Ensemble likewise performed well on SVR by 62.7% and LR by 65.45%(ISBSG Dataset on Literature-Based Feature Selection).

B. ML ENSEMBLE MODEL VS. STANDALONE ML MODEL

RQ2: Does Ensemble Model Produced Better Results as Compared to Machine Learning Model?: ML Ensemble models have improved Standalone Model results on k-NN by 55%. (ISBSG Dataset with Literature-Based Feature selection). The ML Ensemble has also delivered successful results on SVR by 47% and LR by 50% with light-darker shading (ISBSG Dataset with Literature-Based Feature selection).

The cell with the white shade, on the other hand, shows that there was no appreciable difference between the performance of the ML Ensemble model and that of Standalone Models, including the variants of Standalone models LR and XG-Boost. The negative numbers show that the Standalone models' values were bigger than those of the ML Ensemble, which skews the findings in that way.

C. RESULTS BASED ON FEATURE SELECTION

RQ3: Does the Variation in Feature Selection of the Dataset Have Any Impact on Result Accuracy: In our study we have used feature selection not only to avoid the irrelevant information. Nevertheless, it also help us to avoid the biasness of the results. We have used three variation in our experiment just to make sure that we have fully grasp the understanding of the implementation of the ML technique and our proposed model on our datasets.

The feature selection process is one of the crucial steps in a feature engineering process. This technique involves reducing the amount of input variables to produce a predictive model. Feature selection procedures are used to reduce the number of input variables by eliminating unused or duplicated characteristics. The most crucial features for the machine learning model are then eliminated from the list of features. In machine learning, a feature selection aim chooses the most advantageous set of characteristics that can be used to build powerful models of the phenomenon being studied.

Lets compare the results of ANN algorithm in different feature selection setups i.e. all features, selection based on literature and on F-Regression. The training time of ANN in All features was (436.409), in literature based selection ANN spend (20.53), while on F-Regression ANN yield (161.36). As we can observe that with all the irrelevant features, ANN tends to take enormous time. Simply including all features tends to waste time and eventually the results would not be optimal. Hence we can say that feature selection not only provide optimal results but also save a lot of precious time.

The MMRE of ANN in All features was (0.06), in literature based selection ANN spend (0.22), while on F-Regression ANN yield (0.06). However, PRED(25) of ANN in All features was (0.948), in literature based selection ANN spend (0.74), while on F-Regression ANN yield (0.953). Concluding the results based on precision metrics, we can say that ANN has produced better results on feature selection.

In our concluding remarks we can say that feature selection increase accuracy. It also enhances the algorithms' capacity to predict results by concentrating on the most crucial variables and eliminating the unnecessary ones. it minimizes the overfitting, improve accuracy and reduce training time. Decisions are less likely to be based on noise when there are fewer redundant data points. Less data allows algorithms to operate more rapidly.

TABLE 23. Results Comparation of (Hetrogen) Proposed Ensemble Model VS ML Standalone Models.

		(Hetrogen) Proposed Ensemble vs. ML Standalone Algorithms					
		k-NN	SVR	LR	XGB	ANN	(Hetrogen.)
Proposed Ensemble							
	ISBSG_ALL CS	0.2137	0.1985	0.4534	0.7821	0.0515	
	ISBSG_LR CS	0.7045	0.6273	0.6545	0.2545	0.2545	
on case studies	ISBSG_FS CS	0.1356	0.1174	0.4546	0.1858	0.0466	1

TABLE 24. Results Comparison of ML Ensemble Model VS ML Standalone Models.

		ML Ensemble Model VS ML Standalone Models					
		K-NN	SVR	LR	XGBoost	ANN	ML Ensemble
	ISBSG_ALL CS	0.0637	0.0485	0.3034	0.0679	-0.0985	
	ISBSG_LR CS	0.5545	0.4773	0.5045	0.1045	0.1045	
on case studies	ISBSG_FS CS	-0.0144	-0.0326	0.3046	0.0358	-0.1034	0.8500

TABLE 25. Technical complexity factors of selected cases T1 - T13.

Factor	Description	Weight (W)	Related Value (RV) 0-5								Impact (I = W × RV)							
			CS1	CS2	CS3	CS4	CS5	CS6	CS7	CS8	CS1	CS2	CS3	CS4	CS5	CS6	CS7	CS8
T1	Distributed System	2	0	0	0	0	1	0	0	3	0	0	0	0	2	0	0	6
T2	Response time or throughput performance objectives	1	3	4	3	4	3	4	4	4	3	4	3	4	3	4	4	4
T3	End user efficiency	1	5	5	5	4	5	3	5	5	5	5	5	4	5	3	5	5
T4	Complex internal processing	1	1	2	4	3	4	2	3	1	1	2	4	3	4	2	3	1
T5	Code must be reusable (reusability)	1	0	1	1	1	1	3	2	1	0	1	1	1	1	3	2	1
T6	Easy to install	0.5	5	5	5	4	4	5	5	4	2.5	2.5	2.5	2	2	2.5	2.5	2
T7	Easy to use	0.5	5	5	5	5	4	5	5	4	2.5	2.5	2.5	2.5	2	2.5	2.5	2
T8	Portable	2	0	1	1	1	2	1	1	0	0	2	2	2	2	2	2	0
T9	Easy to change	1	5	5	4	4	3	5	5	3	5	5	4	4	3	5	5	3
T10	Concurrent	1	0	0	0	0	2	1	1	2	0	0	0	0	2	1	1	2
T11	Includes special security features	1	3	3	2	3	5	3	3	5	3	3	2	3	5	3	3	5
T12	Provides direct access for third parties	1	1	2	4	3	4	2	3	1	1	2	4	3	4	2	3	1
T13	Special user training facilities	1	3	2	3	4	5	3	2	5	3	2	3	4	5	3	2	5
	Technical Factor (TFactor)										27	33	33	32.5	40	33	35	39
	Technical Complexity Factor (TCF)										0.87	0.93	0.93	0.925	1	0.93	0.95	0.99

XI. LIMITATION OF THE PROPOSED HETEROGENEOUS ENSEMBLE MODEL

Heterogeneous ensemble models combine different types of base models, such as decision trees, neural networks, and support vector machines, to improve the overall performance of the ensemble. However, these models also have several limitations, including:

- Complexity: Our proposed Heterogeneous ensemble model can be more complex than homogeneous ensemble models because they combine different types of models with different parameters, architectures, and hyper-parameters. This can make it difficult to interpret the results and diagnose problems with the model.
- Over-fitting: Our proposed Heterogeneous ensemble model can be prone to over-fitting, if the base models are not diverse enough or if the ensemble is too complex. This can lead to a decrease in performance on new data.
- Computational cost: Heterogeneous ensemble models can be more computationally expensive than homogeneous ensemble models because they require training multiple types of models with different parameters and architectures. In our case we tried different cleaned datasets but still a lot of training was served. It would be interesting to see how proposed model react when encounter a diverse, large and complex dataset.

TABLE 26. Environmental Complexity Factor (ECF).

Factor	Description	Weight (W)	Related Value (RV) 0-5								Impact (I = W × RV)							
			CS1		CS2		CS3		CS4		CS1		CS2		CS1		CS1	
			Exp-1	Exp-2	Exp-1	Exp-2	Exp-1	Exp-2	Exp-1	Exp-1	Exp-1	Exp-2	Exp-1	Exp-2	Exp-1	Exp-2	Exp-1	Exp-2
E1	Familiar with the project model that is used	1.5	3	4	4	5	3	5	4	5	4.5	6	6	7.5	4.5	7.5	6	7.5
E2	Application experience	0.5	3	3	4	3	4	4	3	4	1.5	1.5	2	1.5	2	2	1.5	2
E3	Object-oriented experience	1	5	4	5	4	5	5	4	5	5	4	5	4	5	5	4	5
E4	Lead analyst capability	0.5	4	3	5	4	4	4	4	5	2	1.5	2.5	2	2	2	2	2.5
E5	Motivation	1	3	5	4	5	3	4	5	5	3	5	4	5	3	4	5	5
E6	Stable requirements	2	4	3	5	5	5	4	4	5	8	6	10	10	10	8	8	10
E7	Part-time staff	-1	0	0	0	1	1	0	0	1	0	0	0	-1	-1	0	0	-1
E8	Difficult programming language	-1	3	2	0	2	2	0	3	2	-3	-2	0	-2	-2	0	-3	-2
Environmental Factor (EFactor)											21	22	29.5	27	23.5	28.5	25.5	29
Environmental Complexity Factor (ECF)											0.77	0.74	0.515	0.59	0.695	0.545	0.635	0.53

TABLE 27. Environmental Values Assigned by Expert.

	CS-1		CS-2		CS-3		CS-4		CS-5		CS-6		CS-7		CS-8	
EXPERTS	EXP-1	EXP-2	EXP-1	EXP-2	EXP-1	EXP-2	EXP-1	EXP-2	EXP-1	EXP-2	EXP-1	EXP-2	EXP-1	EXP-2	EXP-1	EXP-2
E1 : knowledgeable about the development methodology being used																
With a score of 0, the team has no understanding of the development process.																
1 points: The group has a conceptual grasp of how development occurs.																
Scores of 2-3 show that at least one team member has occasionally used the strategy.																
Scores of 3 to 4 indicate that at least half of the team members have applied the strategy to other projects.	✓	✓		✓	✓		✓	✓	✓	✓		✓	✓		✓	✓
Score of 5: All team members have applied the method to a range of projects.			✓			✓					✓			✓		
E2: Overall application experience																
Score 0: The team as a whole lacks experience.																
Score 1-2: Only a few team members—for example, those with one to eleven-and-a-half years of experience—have any prior knowledge.																
Score 3: Each member of the team has at least 1.5 years of experience.			✓			✓				✓	✓	✓	✓			
Score 4: The majority of the team has two years' worth of experience.		✓		✓	✓		✓		✓					✓	✓	
Each team member has relevant experience, earning a score of 5.	✓							✓								✓
E3: Experience with Objects																
Score 0: The squad has no prior knowledge of OOAD.																
Score 1: No one on the team has more than a year of experience.																
Score 2-3: All team members have between one and one and a half years of experience.				✓			✓	✓								
Score 4: The team's majority of members have more than two years of experience.		✓	✓			✓			✓	✓		✓			✓	✓
Score 5: Experienced developers are all present (more than 2 years).	✓				✓						✓		✓	✓		
E4: Capability of the lead analyst Score 0: The lead analyst is inexperienced Score 1-2: Knowledge from a few projects EXPERTS																
Score 3–4: Several projects over the course of at least two years of experience																
Score 5: Three years or more of experience working on various projects																
E5 - Inspiration																
0 points equals no motivation.																
Score 1-2: Insufficient motivation								✓								
Score 3–4: The group is driven to perform well.	✓		✓	✓			✓		✓		✓		✓			
Score 5: The group is highly inspired and motivated.		✓			✓	✓				✓	✓	✓		✓	✓	✓
E6: Stable conditions																
Score 0: Extremely erratic requirements, always changing.																
Score 1-2: Unstable conditions.																
The customer requests numerous adjustments at different times.																
Score 3–4: Stability all around. Minor adjustments are required.			✓	✓			✓	✓				✓	✓		✓	
Score 5: Consistent specifications throughout.	✓	✓			✓	✓			✓	✓	✓			✓		✓
E7: Part-time staff																
Zero points: There are none.	✓	✓				✓		✓		✓	✓			✓	✓	
Score 1-2: A small (20%) number of part-time workers			✓	✓	✓		✓		✓			✓	✓			✓
3–4: Half of the team consists of part-timers.																
Score of 5: Each team member has a part-time job.																
E8 is a difficult programming language.																
Score 0: All in the team are seasoned programmers.	✓	✓				✓	✓		✓	✓	✓			✓		✓
Score 1: The average tenure of the team as a whole is greater than two years.			✓		✓			✓					✓		✓	
Score 2: The group as a whole has more than 1.5 years of experience.				✓								✓				
Score 3: The majority of the team members have more than a year of combined experience.																
Score 4: The bulk of the team are novices, with only a tiny percentage having experience, such as one year.																
Score of 5: The team as a whole lacks experience.																

TABLE 28. Expert's Feedback.

2[0]*ID	CS1			CS2			CS3			CS4			CS5			CS6			CS7			CS8		
	Exp-1	Exp-2	Est.Effort (Man-Hour)	Exp-1	Exp-2	Est.Effort (Man-Hour)	Exp-1	Exp-2	Est.Effort (Man-Hour)	Exp-1	Exp-2	Est.Effort (Man-Hour)	Exp-1	Exp-2	Est.Effort (Man-Hour)	Exp-1	Exp-2	Est.Effort (Man-Hour)	Exp-1	Exp-2	Est.Effort (Man-Hour)	Exp-1	Exp-2	Est.Effort (Man-Hour)
FR_1	75	55	65	60	55	57.5	60	65	62.5	80	75	77.5	110	120	115	70	80	75	120	100	110	90	70	80
FR_2	80	70	75	70	90	80	70	50	60	60	75	67.5	140	130	135	90	105	97.5	130	140	135	105	120	112.5
FR_3	110	125	117.5	65	80	72.5	45	75	60	95	105	100	120	110	115	110	135	122.5	110	60	85	70	80	75
FR_4	90	105	97.5	50	60	55	80	95	87.5	110	95	102.5	65	75	70	85	105	95	110	120	115	150	130	140
FR_5	110	135	122.5	110	90	100	75	90	82.5	40	50	45	130	120	125	115	130	122.5	130	140	135	80	95	87.5
FR_6	85	105	95	90	60	75	40	45	42.5	60	60	60	90	80	85	125	140	132.5	110	120	115	110	100	105
FR_7	115	130	122.5	120	110	115	85	95	90	70	70	70	135	120	127.5	120	110	115	140	130	125	70	80	75
FR_8	125	140	132.5	40	50	45	50	45	47.5	45	60	52.5	110	115	112.5	75	80	77.5	120	110	125	160	145	152.5
FR_9	90	120	105	60	65	62.5	55	70	62.5	80	70	75	70	80	75	90	110	100	65	75	97.5	90	70	80
FR_10	110	120	115	70	50	60	40	55	47.5	75	70	72.5	60	70	65	120	110	115	130	120	125	40	50	45
FR_11	80	60	70	45	75	60	60	75	67.5	110	80	95	90	70	80	75	90	82.5	110	120	115	110	125	117.5
FR_12	70	50	60	80	95	87.5	40	50	45	80	100	90			0			0	135	110	122.5	90	105	97.5
FR_13	60	75	67.5	75	90	82.5	45	60	52.5	125	55	90			0			0	110	70	90	110	135	122.5
FR_14	90	70	80	110	90	100	80	75	77.5	90	80	85			0			0	130	145	137.5	85	105	95
FR_15	105	120	112.5	80	95	87.5	70	85	77.5	110	90	100			0			0	150	140	145	115	130	122.5
FR_16	65	75	70	50	60	55	40	50	45	80	75	77.5			0			0	170	145	157.5	125	140	132.5
FR_17	80	100	90	60	70	65	70	80	75	70	120	95			0			0	130	100	115	120	90	105
FR_18	50	65	57.5	50	60	55	60	70	65	60	70	65			0			0	65	70	67.5	60	70	65
FR_19	60	75	67.5	75	70	72.5	90	70	80	90	60	75			0			0	120	110	115	80	100	90
FR_20	85	95	90	60	70	65	50	70	60	105	90	97.5			0			0	130	115	122.5	120	130	125
FR_21	105	115	110	75	80	77.5	70	80	75	65	105	85			0			0	120	110	115	60	75	67.5
FR_22	110	135	122.5	95	100	97.5	80	100	90	80	70	75			0			0	110	100	105	85	95	90
FR_23	85	105	95	45	55	50	60	65	62.5	75	80	77.5			0			0	120	110	115	105	115	110
FR_24	115	130	122.5	90	80	85			0			0			0			0	140	130	135	90	75	82.5
FR_25	125	140	132.5	50	75	62.5			0			0			0			0	110	130	120	105	110	107.5
FR_26	90	120	105	65	80	72.5			0			0			0			0	160	140	150	120	130	125
FR_27	110	120	115	75	80	77.5			0			0			0			0	130	115	122.5	70	60	65
FR_28	140	120	130	80	60	70			0			0			0			0	125	145	135	80	95	87.5
FR_29	120	130	125	60	70	65			0			0			0			0	95	120	107.5	60	75	67.5
FR_30	80	60	70	50	50	50			0			0			0			0	110	130	120	140	150	145
TOTAL	2815	3065	2940	2105	2215	2160	1415	1615	1515	1855	1805	1830	1120	1090	1105	1075	1195	1135	3635	3470	3580	2895	3050	2973

- Training data requirements: Heterogeneous ensemble models require more training data than homogeneous ensemble models because they have more parameters and architectures to learn. This can be a problem if the dataset is small or imbalanced. We faced similar problem when applying to another benchmark dataset i.e. UCP Dataset.
- Integration challenges: Heterogeneous ensemble models can be challenging to integrate with existing systems or workflows because they require different types of models and may have different input and output formats.

XII. CONCLUSION AND FUTURE DIRECTION

The primary goal of this research is to employ the ensemble technique to improve the precision of performance estimates for software development effort. For estimating software effort, we provided a state-of-the-art, Heterogeneous Ensemble model. To increase the efficiency and precision of software development initiatives, a heterogeneous ensemble model that combines artificial neural networks (ANN), use case points (UCP), and expert judgement (EJ) is proposed and implemented on benchmark dataset i.e. ISBSG and Industrial case studies. We have used different variation in using the dataset not only to avoid the biasness in results but also to observe the difference in results. It was interesting to see how machine learning algorithms react to different experimental setups.

During our experiments, we have found that ensemble models have produced better results as compared to standalone models on the benchmark datasets. On the other hand, our proposed heterogeneous ensemble model have also shown exceptional results on industrial case studies. We have observed in the literature [10], [55], and [56] that ensemble model have produce better results as compared to standalone models. With our experiments, not only we have proved the above said observations, but with our proposed heterogeneous model, it has open new dimensions of research. In future, we are going to implement the same setup (or with different combination rule) on some other datasets to observe, does number of features has any impact on the observations? does change in algorithms setup like changing the activation function of Artificial Neural Network or combination rule, have any impact on the heterogeneous ensemble model's results?. Moreover, we are planning to compare our heterogeneous ensemble model with different ensemble model setup and try to observe, how we can improve the accuracy by tuning the machine learning algorithms. It would be interesting to see how heterogeneous ensemble models compete with powerful ensemble models with respect to training time and precision metrics.

Our proposed ensemble model can provide better performance than a single model. However, it is arguable that a heterogeneous ensemble model is the state of the art for all problems. The choice of modeling approach depends on the specific problem, the available data, and the desired outcome. This is because different methods can capture different aspects of the data, and combining them can lead to a more robust and accurate prediction.

In the specific case of software development effort estimation, UCP is a widely used method for predicting the size and complexity of a software project, while ANNs have been shown to perform well in predicting software development effort. Expert judgment can also provide valuable insights and context-specific knowledge. Therefore, combining UCP, ANN, and expert judgment in a heterogeneous ensemble model for software development effort estimation can leverage the strengths of each method and lead to more accurate and reliable predictions. However, the effectiveness of the ensemble model also depends on how the individual models are combined and how the weights are assigned to each model.

APPENDIX A
TECHNICAL COMPLEXITY FACTOR

See Table 25.

APPENDIX B
ENVIRONMENTAL COMPLEXITY FACTOR (ECF)

See Table 26.

APPENDIX C
ENVIRONMENTAL VALUES ASSIGNED BY EXPERT

See Table 27.

APPENDIX D
EXPERT's FEEDBACK

See Table 28.

ACKNOWLEDGMENT

The authors would like to thank Dr. Yasir Mahmood (Universiti Teknologi Malaysia), Dr. Yaseen Khan (Daraz-Alibaba Group Inc.), and Dr. Salman Ahmed Khan (The University of Lahore, Pakistan) for their sincere guidance. This research work is dedicated to Dr. Chao Liu, who unfortunately passed away before this study report was published. He was an exceptional teacher, a supervisor, and a mentor. They all grieve his passing deeply.

REFERENCES

[1] N. Rankovic, D. Rankovic, M. Ivanovic, and L. Lazic, "A new approach to software effort estimation using different artificial neural network architectures and Taguchi orthogonal arrays," *IEEE Access*, vol. 9, pp. 26926–26936, 2021.

[2] P. Rai, D. K. Verma, and S. Kumar, "A hybrid model for prediction of software effort based on team size," *IET Softw.*, vol. 15, no. 6, pp. 365–375, Dec. 2021.

[3] Z. Dan, "Improving the accuracy in software effort estimation: Using artificial neural network model based on particle swarm optimization," in *Proc. IEEE Int. Conf. Service Oper. Logistics, Informat. (SOLI)*, Jul. 2013, pp. 180–185.

[4] *CHAOS Report 2015*, Standish Group Int., Boston, MA, USA, 2015, p. 13.

[5] *The Standish Group CHAOS Report*, Standish Group Int., Boston, MA, USA, 2009.

[6] M. Jørgensen, "Forecasting of software development work effort: Evidence on expert judgement and formal models," *Int. J. Forecasting*, vol. 23, no. 3, pp. 449–462, Jul. 2007.

[7] M. Z. M. Hazil, M. N. Mahdi, M. S. M. Azmi, L. K. Cheng, A. Yusof, and A. R. Ahmad, "Software project management using machine learning technique—A review," in *Proc. 8th Int. Conf. Inf. Technol. Multimedia (ICIMU)*, Aug. 2020, pp. 363–370.

[8] S.-J. Huang and N.-H. Chiu, "Optimization of analogy weights by genetic algorithm for software effort estimation," *Inf. Softw. Technol.*, vol. 48, no. 11, pp. 1034–1045, Nov. 2006.

[9] M. Auer, A. Trendowicz, B. Graser, E. Haunschmid, and S. Biffl, "Optimal project feature weights in analogy-based cost estimation: Improvement and limitations," *IEEE Trans. Softw. Eng.*, vol. 32, no. 2, pp. 83–92, Feb. 2006.

[10] J. Wen, S. Li, Z. Lin, Y. Hu, and C. Huang, "Systematic literature review of machine learning based software development effort estimation models," *Inf. Softw. Technol.*, vol. 54, no. 1, pp. 41–59, Jan. 2012.

[11] M. A. Ahmed, I. Ahmad, and J. S. AlGhamdi, "Probabilistic size proxy for software effort prediction: A framework," *Inf. Softw. Technol.*, vol. 55, no. 2, pp. 241–251, Feb. 2013.

[12] H. Leung and Z. Fan, "Software cost estimation," in *Handbook of Software Engineering & Knowledge Engineering*. Hong Kong: The Hong Kong Polytechnic Univ., 2002.

[13] M. Jørgensen, "Communication of software cost estimates," in *Proc. 18th Int. Conf. Eval. Assessment Softw. Eng.*, Jul. 2014, pp. 1–5.

[14] Y. Mahmood, N. Kama, and A. Azmi, "A systematic review of studies on use case points and expert-based estimation of software development effort," *J. Softw., Evol. Process*, vol. 32, no. 7, Jul. 2020, Art. no. e2245.

[15] T. K. Abdel-Hamid, "On the utility of historical project statistics for cost and schedule estimation: Results from a simulation-based case study," *J. Syst. Softw.*, vol. 13, no. 1, pp. 71–82, Sep. 1990.

[16] B. W. Boehm and R. Valerdi, "Achievements and challenges in cocomo-based software resource estimation," *IEEE Softw.*, vol. 25, no. 5, pp. 74–83, Sep. 2008.

[17] B. W. Boehm, *Software Engineering Economics*. Upper Saddle River, NJ, USA: Prentice-Hall, 1981.

[18] L. H. Putnam, "A general empirical solution to the macro software sizing and estimating problem," *IEEE Trans. Softw. Eng.*, vol. SE-4, no. 4, pp. 345–361, Jul. 1978.

[19] A. J. Albrecht, "Measuring application development productivity," in *Proc. IBM Appl. Develop. Symp.*, Oct. 1979, pp. 83–92.

[20] A. J. Albrecht and J. E. Gaffney, "Software function, source lines of code, and development effort prediction: A software science validation," *IEEE Trans. Softw. Eng.*, vol. SE-8, no. 6, pp. 639–648, Nov. 1983.

[21] *Users Manual El Segundo*, SEER-SEM, Galorath, Los Angeles, CA, USA, 2001.

[22] H. A. Rubin, "Interactive macro-estimation of software life cycle parameters via personal computer: A technique for improving customer/developer communication," in *Proc. Symp. Appl. Assessment Automated Tools Softw. Develop.* San Francisco, CA, USA: IEEE, 1983, pp. 44–54.

[23] Y.-C. Ho and C. D. McDevitt, "Determination of optimal resource allocation for software development—An application of a software equation," *Inf. Manage.*, vol. 18, no. 2, pp. 79–85, Feb. 1990.

[24] F. J. Heemstra, "Software cost estimation," *Inf. Softw. Technol.*, vol. 34, no. 10, pp. 627–639, Oct. 1992.

[25] Y. Yang, B. Boehm, and B. Clark, "Assessing COTS integration risk using cost estimation inputs," in *Proc. 28th Int. Conf. Softw. Eng.*, May 2006, pp. 431–438.

[26] J. Grenning, "Planning poker or how to avoid analysis paralysis while release planning," *Hawthorn Woods, Renaissance Softw. Consulting*, vol. 3, 2002.

[27] C. L. Martín, J. L. Pasquier, C. M. Yáñez, and A. T. Gutiérrez, "Software development effort estimation using fuzzy logic: A case study," in *Proc. 6th Mex. Int. Conf. Comput. Sci. (ENC)*, 2005, pp. 113–120.

[28] C. E. Walston and C. P. Felix, "A method of programming measurement and estimation," *IBM Syst. J.*, vol. 16, no. 1, pp. 54–73, 1977.

[29] S. K. Rath, B. P. Acharya, and S. M. Satapathy, "Early stage software effort estimation using random forest technique based on use case points," *IET Softw.*, vol. 10, no. 1, pp. 10–17, Jan. 2016.

[30] D. Wu, J. Li, and C. Bao, "Case-based reasoning with optimized weight derived by particle swarm optimization for software effort estimation," *Soft Comput.*, vol. 22, no. 16, pp. 5299–5310, Aug. 2018.

[31] J.-S. Chou and C.-C. Wu, "Estimating software project effort for manufacturing firms," *Comput. Ind.*, vol. 64, no. 6, pp. 732–740, Aug. 2013.

[32] A. B. Nassif, D. Ho, and L. F. Capretz, "Towards an early software estimation using log-linear regression and a multilayer perceptron model," *J. Syst. Softw.*, vol. 86, no. 1, pp. 144–160, Jan. 2013.

[33] F. Sarro, A. Petrozziello, and M. Harman, "Multi-objective software effort estimation," in *Proc. 38th Int. Conf. Softw. Eng.*, May 2016, pp. 619–630.

[34] M. Shepperd and G. Kadoda, "Comparing software prediction techniques using simulation," *IEEE Trans. Softw. Eng.*, vol. 27, no. 11, pp. 1014–1022, Nov. 2001.

[35] E. Kocaguneli, A. Tosun, and A. Bener, "AI-based models for software effort estimation," in *Proc. 36th EUROMICRO Conf. Softw. Eng. Adv. Appl. (SEAA)*, Sep. 2010, pp. 323–326.

[36] L. L. Minku and X. Yao, "Software effort estimation as a multiobjective learning problem," *ACM Trans. Softw. Eng. Methodol.*, vol. 22, no. 4, pp. 1–32, Oct. 2013.

[37] M. Azzeh, A. B. Nassif, and L. L. Minku, "An empirical evaluation of ensemble adjustment methods for analogy-based effort estimation," *J. Syst. Softw.*, vol. 103, pp. 36–52, May 2015.

[38] A. Idri, M. Hosni, and A. Abran, "Improved estimation of software development effort using classical and fuzzy analogy ensembles," *Appl. Soft Comput.*, vol. 49, pp. 990–1019, Dec. 2016.

[39] M. O. Elish, "Assessment of voting ensemble for estimating software development effort," in *Proc. IEEE Symp. Comput. Intell. Data Mining (CIDM)*, Apr. 2013, pp. 316–321.

[40] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Softw. Eng.*, vol. 14, no. 2, pp. 131–164, Apr. 2009.

[41] L. Breiman, "Bagging predictors," *Mach. Learn.*, vol. 24, no. 2, pp. 123–140, 1996. [Online]. Available: https://link.springer.com/article/10.1007

[42] Y. Liu, X. Yao, and T. Higuchi, "Evolutionary ensembles with negative correlation learning," *IEEE Trans. Evol. Comput.*, vol. 4, no. 4, pp. 380–387, Nov. 2000.

[43] M. Hosni, A. Idri, and A. Abran, "On the value of filter feature selection techniques in homogeneous ensembles effort estimation," *J. Softw., Evol. Process*, vol. 33, no. 6, pp. 1–38, Jun. 2021.

[44] E. Kocaguneli, T. Menzies, and J. W. Keung, "On the value of ensemble effort estimation," *IEEE Trans. Softw. Eng.*, vol. 38, no. 6, pp. 1403–1416, Nov. 2012.

[45] M. Shepperd and C. Schofield, "Estimating software project effort using analogies," *IEEE Trans. Softw. Eng.*, vol. 23, no. 11, pp. 736–743, Nov. 1997.

[46] A. B. Nassif, M. Azzeh, L. F. Capretz, and D. Ho, "Neural network models for software development effort estimation: A comparative study," *Neural Comput. Appl.*, vol. 27, no. 8, pp. 2369–2381, Nov. 2016.

[47] K. Roy Clemmons, "Project estimation with use case points," *CrossTalk*, vol. 19, no. 2, pp. 18–22, 2006.

[48] G. Karner, "Resource estimation for objectory projects," Objective Syst. SF AB, 1993.

[49] C. López-Martín and A. Abran, "Neural networks for predicting the duration of new software projects," *J. Syst. Softw.*, vol. 101, pp. 127–135, Mar. 2015.

[50] D. R. Pai, K. S. McFall, and G. H. Subramanian, "Software effort estimation using a neural network ensemble," *J. Comput. Inf. Syst.*, vol. 53, no. 4, pp. 49–58, Jun. 2013.

[51] C. Banerjee, T. Mukherjee, and E. Pasiliao, "An empirical study on generalizations of the ReLU activation function," in *Proc. ACM Southeast Conf.*, Apr. 2019, pp. 164–167.

[52] A. F. Agarap, "Deep learning using rectified linear units (ReLU)," 2018, *arXiv:1803.08375*.

[53] D. Boob, S. S. Dey, and G. Lan, "Complexity of training ReLU neural network," *Discrete Optim.*, vol. 44, pp. 1–21, May 2020.

[54] M. Jørgensen, "Unit effects in software project effort estimation: Work-hours gives lower effort estimates than workdays," *J. Syst. Softw.*, vol. 117, pp. 274–281, Jul. 2016.

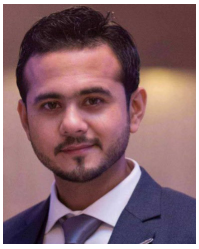
[55] A. Idri, M. Hosni, and A. Abran, "Systematic literature review of ensemble effort estimation," *J. Syst. Softw.*, vol. 118, pp. 151–175, Aug. 2016.

[56] Y. Mahmood, N. Kama, A. Azmi, A. S. Khan, and M. Ali, "Software effort estimation accuracy prediction of machine learning techniques: A systematic performance evaluation," *Softw., Pract. Exp.*, vol. 52, no. 1, pp. 39–65, Jan. 2022.

[57] S. S. Ali, M. S. Zafar, and M. T. Saeed, "Effort estimation problems in software maintenance—A survey," in *Proc. 3rd Int. Conf. Comput., Math. Eng. Technol. (iCoMET)*, Jan. 2020, pp. 1–9.

[58] C. F. Kemerer, "An empirical validation of software cost estimation models," *Commun. ACM*, vol. 30, no. 5, pp. 416–429, May 1987.

[59] Z. Li, "Intelligently predict project effort by reduced models based on multiple regressions and genetic algorithms with neural networks," in *Proc. Int. Conf. E-Bus. E-Government (ICEE)*, May 2010, pp. 1536–1542.
[60] A. Idri, I. Abnane, and A. Abran, "Evaluating Pred(*p*) and standardized accuracy criteria in software development effort estimation," *J. Softw., Evol. Process*, vol. 30, no. 4, pp. 1–15, Apr. 2018.
[61] S.-J. Huang and N.-H. Chiu, "Applying fuzzy neural network to estimate software development effort," *Int. J. Speech Technol.*, vol. 30, no. 2, pp. 73–83, Apr. 2009.
[62] S. H. S. Moosavi and V. K. Bardsiri, "Satin bowerbird optimizer: A new optimization algorithm to optimize ANFIS for software development effort estimation," *Eng. Appl. Artif. Intell.*, vol. 60, pp. 1–15, Apr. 2017.
[63] F. González-Ladrón-de-Guevara, M. Fernández-Diego, and C. Lokan, "The usage of ISBSG data fields in software effort estimation: A systematic mapping study," *J. Syst. Softw.*, vol. 113, pp. 188–215, Mar. 2016.
[64] W. D. O. Bussab and P. A. Morettin, *Estatística Básica*, 2010, pp. 16–540.
[65] G. Schneider and J. P. Winters, *Applying Use Cases: A Practical Guide*. India: Pearson, 2001.



SYED SARMAD ALI received the bachelor's degree in computer engineering from Sir Syed University of Engineering and Technology (SSUET), Pakistan, and the M.S. degree from Coventry University, Coventry, U.K., in 2012. He is currently pursuing the Ph.D. degree in software engineering with the Software Engineering Institute (SEI), Beihang University, Beijing, China.

Apart from working in the industry, he served as an Assistant Professor for ten years at different universities. His research interests include data science and machine learning, deep learning, ensemble models (heterogeneous and homogeneous), and MOEAs using co-evolution in search-based software engineering, under the supervision of Prof. Chao Liu, Prof. Ren Jian, and Prof. Ji Wu (Beihang Software Testing and Evolution Laboratory—BHSTEL).

JIAN REN received the dual M.Sc. degrees from the Queen Mary University of London and Kings College London and the Ph.D. degree in computer science from the University College London. He is currently an Assistant Professor with the School of Computer Science, Beihang University, Beijing. His research interests include search-based software engineering, software project planning and management, requirements engineering, and evolutionary computation.

KUI ZHANG received the master's degree in information management and information systems from the Beijing Institute of Technology, Beijing, China, in 2010, and the Ph.D. degree from the Software Engineering Institute (SEI), Beihang University. His research interests include model-driven engineering, model-based real-time analysis, airworthiness certification, model-based safety analysis, and general model-based software engineering.

JI WU received the M.S. degree from the Second Research Institute, China Aerospace Science and Industry Group, in 1999, and the Ph.D. degree from Beihang University, in 2003. He is currently an Associate Professor of software engineering with Beihang University. His research interests include embedded systems and software modeling and verification, software requirement and architecture modeling and verification, safety and reliability assessment, and software testing.

CHAO LIU received the M.S. degree in computer software and theory and the Ph.D. degree from Beihang University. He is currently a Professor of software engineering with Beihang University. In the last ten years, he mainly focuses on the modeling and verification of safety-critical software and systems, including safety requirement modeling and analysis, evidence-based software safety analysis and evaluation, software safety and reliability analysis based on the software development process, and model-driven software testing. His research interests include software quality engineering, software testing, model-driven software development, and software process improvement.