Heterogeneous Ensemble Model For Software Effort Estimation

Submitted in partial fulfilment of the requirements of the degree of

Bachelor of Technology (B.Tech)

by

Gaddala Sai Kamal (21CSB0A17)

Elkapelly Pranay Suhas (21CSB0A16)

Vadithya Sanjay (21CSB0A64)

Supervisors:

Dr. Manjubala Bisi (Supervisor)

Department of Computer Science and Engineering

NATIONAL INSTITUTE OF TECHNOLOGY WARANGAL

2025

Acknowledgement

We express our hearty gratitude towards Dr. Manjubala (Supervisor) Ma'am for her valuable guidance, supervision, suggestions, encouragement, and the help throughout the semester for the completion of my project work. She kept me going when I was down and gave me the courage to keep moving forward.

We want to take this opportunity once again to thank Dr. R Padmavathy, Head of the Department, Computer Science and Engineering, NIT Warangal, for giving us this opportunity and resources to work on this project and supporting us throughout.

We also want to thank the evaluation committee for their valuable suggestions on my proposals and research and for conducting a smooth presentation of the project.

Gaddala Sai Kamal	Elkapelly Pranay Suhas	Vadithya Sanjay
21CSB0A17	21CSB0A16	21CSB0A64

Date :

Declaration

We declare that this written submission represents our ideas, our supervisor's ideas in our own words and where others' ideas or words have been included, we have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in our submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Gaddala Sai Kamal 21CSB0A17	Elkapelly Pranay Suhas 21CSB0A16	Vadithya Sanjay 21CSB0A64

Date :

APPROVAL SHEET

This Dissertation Work entitled **"Heterogeneous Ensemble Model For Software Effort Estimation"** by **Gaddala Sai Kamal(21CSB0A17),Elkapelly Pranay Suhas(21CSB0A16) and Vadithya Sanjay(21CSB0A64)** is approved for the degree of Bachelor of Technology (B.Tech) in Computer Science and Engineering.

Examiners

_____

_____

_____

Supervisors

_____

Dr.Manjubala Bisi (Supervisor)

Chairman

_____

Dr. R.Padmavathy (HOD)

Date: _____

NIT Warangal

Certificate

This is to certify that the Dissertation work entitled **"Heterogeneous Ensemble Model For Software Effort Estimation"** is a bonafide record of work carried out by **"Gaddala Sai Kamal (21CSB0A17), Elkapelly Pranay Suhas (21CSB0A16) and Vadithya Sanjay (21CSB0A64)"**, submitted to the Assistant Professor, **Dr. Manjubala Bisi**, Department of Computer Science and Engineering, in partial fulfilment of the requirements for the award of the degree of B.Tech at "National Institute of Technology, Warangal" during the 2021–2025.

(Signature)
Dr. R. Padmavathy
Head of the Department
Department of CSE
NIT Warangal

(Signature)
Dr. Manjubala Bisi
Assistant Professor
Department of CSE
NIT Warangal

Abstract

Software projects are crucial part of a company. To manage software projects, companies need to know the effort required to complete software projects. By estimating the effort of a software project, a company can manage resources accordingly.

Various standalone models are used before to estimate the effort of software projects. Traditional standalone models like COCOMO II, Case-Based Reasoning (CBR) and machine learning techniques (e.g., XGBoost) have been widely used, they often struggle with data variability, noise, and project complexity. This study takes a step forward from the standalone models and combines different standalone models to create a single ensemble model.

This work highlights the potential of hybrid estimation techniques to enhance decision-making in software project management.To test our model, we compared the results of standalone models to the ensemble model for benchmark datasets.Additionally, we implemented an optimized Analogy-Based Model for effort estimation as a replacement for traditional Case-Based Reasoning, further improving the accuracy and adaptability of our approach.

Keywords —- Software effort estimation, COCOMO II, Case-Based Reasoning (CBR), XGBoost, Heterogeneous ensemble model, Analogy Based Model.

Contents

List of Figures

1

Introduction

1.1 The Critical Role of Software Effort Estimation

Software effort is the amount of work required to complete a software project. Software effort is usually measured in person-months. Software effort basically represent time and resources invested throughout the software project by various members involved in the project.

Software effort estimation is used for predicting the effort for a project beforehand so the resources required for a project can be planned accordingly. It is a crucial part of a software project because it can be useful for optimizing resources, budget planning, and reducing project failures.

1.2 Limitations Of Current Estimation Approaches

Traditional software effort estimation techniques, such as expert judgment, analogy-based methods, and algorithmic models like COCOMO, often struggle with accuracy due to their reliance on predefined formulas or subjective inputs. These methods may not adapt well to changes in project characteristics or handle non-linear relationships between variables effectively. Machine learning models, while more flexible, can still suffer from issues like overfitting, sensitivity to noisy data, and inconsistent performance across different datasets. Moreover, many single-model approaches fail to generalize well in real-world scenarios, especially when project data

is limited, imbalanced, or varies widely in scale and complexity.

1.3 Hybrid Solution Approach

This work uses a hybrid approach that combines different types of effort estimation models to get more accurate and reliable results. Instead of relying on just one method, it brings together Case-Based Reasoning (CBR), COCOMO II, and a machine learning model. Each model has its own strengths—COCOMO II uses formulas based on project parameters, CBR looks at past similar projects, and machine learning learns patterns directly from the data. By combining their predictions using the median combination rule, the final estimate is less affected by any one model's errors or outliers. This kind of diverse model setup, known as a heterogeneous ensemble, helps balance different perspectives and makes the estimation process more stable and effective across various types of projects.

1.4 Objectives

- To develop a heterogeneous ensemble model using different stand-alone models .
- To evaluate the performance of the heterogeneous ensemble model on benchmark software effort datasets using standard metrics.
- To compare the results of the heterogeneous ensemble model with stand-alone models using different metrics and different datasets.

2

Related Work

2.1 Traditional Software Estimation Methods

Early approaches to effort estimation relied heavily on expert judgment and algorithmic models. The Constructive Cost Model (COCOMO), introduced by Boehm in the 1980s [2], became a cornerstone for parametric estimation, using project size and cost drivers to predict effort. While effective for waterfall-style projects, these methods struggled to adapt to agile development and modern software complexities. Study of Heemstra (1992) [9] highlighted how rigid formulas often failed to account for team dynamics or emerging technologies, prompting the need for more flexible solutions.

2.2 Machine Learning in Effort Estimation

With the rise of data-driven techniques, researchers began applying machine learning (ML) to estimation. Wen et al. (2012) [16] systematically reviewed ML approaches, showing that regression models and neural networks could capture nonlinear patterns missed by traditional methods. XGBoost, in particular, gained traction for its handling of missing data and feature interactions (Chen et al., 2016 [5]). However, challenges persisted, including overfitting on small datasets and the "black-box" nature of predictions, limiting trust among practitioners.

2.3 Case-Based Reasoning and Analogies

Case-Based Reasoning (CBR) emerged as a knowledge-based alternative, leveraging similarities to past projects. Mukhopadhyay et al. (1992) [11] demonstrated CBR's adaptability for software estimation, especially when historical data was sparse. Later work by Wu et al. (2018) [17] enhanced CBR with optimization techniques, improving similarity measurements.Later work by Mustyala and Bisi (2025) [12] enhanced CBR with optimization techniques, specifically by integrating the Harmony Search algorithm. Recent studies have focused on improving analogy-based effort estimation by integrating regression techniques to enhance prediction accuracy [7]. Despite its strengths, CBR's accuracy depended heavily on case library quality and the choice of similarity metrics—a limitation our ensemble approach aims to address.

2.4 Ensemble Models and Hybrid Approaches

Recent studies have explored combining methods to mitigate individual weaknesses. Ali et al. (2023) [1] proposed heterogeneous ensembles that blended ML with algorithmic models, reporting improved accuracy over standalone techniques. Idri et al. (2016) [10] systematically reviewed ensemble methods, identifying weighted averaging and median selection as effective combination rules. These findings informed our design, particularly in integrating COCOMO II's interpretability with XGBoost's predictive power.

2.5 Research Gaps and Our Contribution

While existing work advanced estimation accuracy, key gaps remained: (1) limited integration of parametric, knowledge-based, and ML models; (2) inadequate handling of small or noisy datasets; and (3) lack of frameworks balancing accuracy with practicality. Our research bridges these gaps by proposing a unified ensemble that harmonizes COCOMO II, CBR, and XGBoost, validated across diverse benchmarks (NASA, COCOMO81).

3

Methodology

This chapter provides the methodology used for developing heterogeneous ensemble model. We used standalone algorithmic and machine learning models to predict the effort. Then, we are going to discuss on our Heterogeneous approach.

3.1 Standalone Models

These are the individual models used in the ensembling process.

3.1.1 Case-Based Reasoning (CBR)

Case-Based Reasoning (CBR) is a reasoning technique that solves new problems based on historical data, i.e., by referencing similar past cases [11]. CBR assumes that similar projects have similar software development efforts. This method is widely used in Software Effort Estimation as it uses historical project data for predicting the effort of new projects. **Process of Effort Estimation:** CBR involves four major phases to predict effort. Each of these phases are explained below:

1. **Case Retrieval:** This phase involves retrieving the most relevant past cases. The new project attributes (e.g., size, complexity, development time) are compared with past projects stored in a case base. The similarity between two cases is measured using distance metrics such as Euclidean distance, cosine similarity, or weighted similarity functions.

2. **Case Reuse:** The retrieved past cases are analyzed, and their effort estimates are adjusted to better fit the attributes of the new project. These modifications consider variations in the project attributes.
3. **Case Revision:** If needed, specialists refine the estimate by incorporating project-specific factors.
4. **Case Retention:** After the project's completion, the actual effort is recorded in the case base for future use.

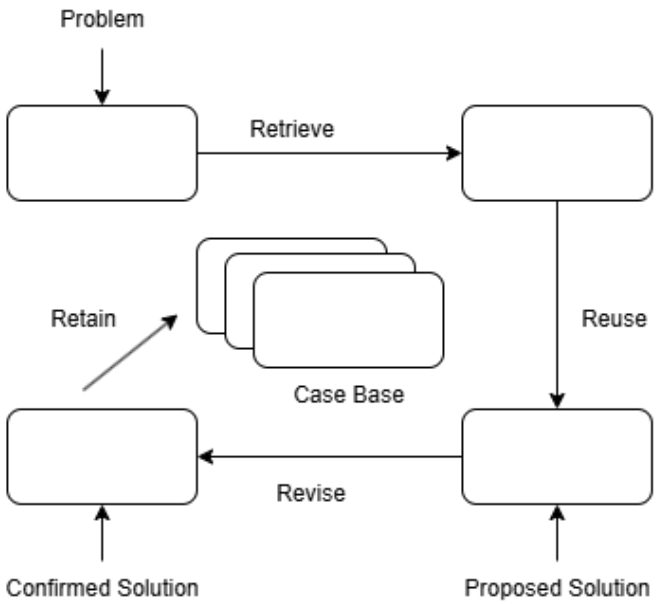

Figure 3.1: Case Based Reasoning(CBR)

The major advantages of CBR is its ability to handle uncertainties in software effort estimation and its adaptability to new data. However, the effectiveness of CBR depends on the quality of stored cases and the choice of similarity measurement.

3.1.2 COCOMO II (Constructive Cost Model)

COCOMO II is an algorithmic cost estimation model that predicts the effort required for software development based on project size and various cost

drivers [3]. It adjusts effort estimates using 17 cost drivers, each having a predefined rating (e.g., Very Low to Extra High), influencing the final effort estimate.

1. **Estimation Formula:** The equation for estimating effort in person-months is:

$$\text{Effort} = A \times (\text{Size})^B \times \prod E_i \tag{3.1}$$

Where:

- A, B - Model-specific constants.
- Size - Estimated code size in KLOC (Kilo Lines of Code).
- E_i - Effort multipliers based on cost drivers (e.g., complexity, team experience).

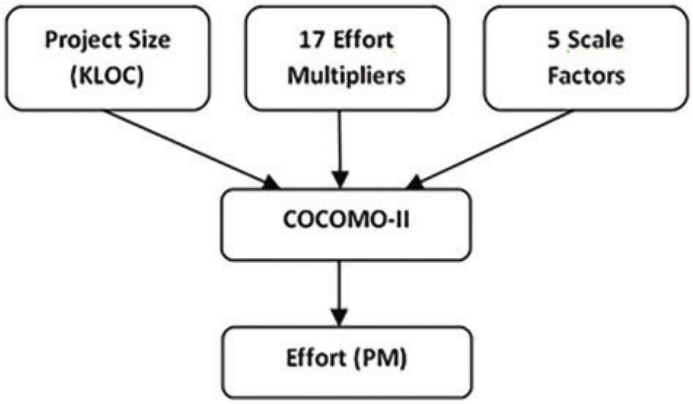

Figure 3.2: COCOMO II

COCOMO-II have some major advantages like this is applicable to different software development stages and supports modern software development paradigms like object-oriented and agile approaches.

3.1.3 XGBoost (Extreme Gradient Boosting)

XGBoost is a powerful and efficient machine learning approach [5]. Basically,it is a gradient boosting algorithm that builds an ensemble of decision trees sequentially. Each tree learns from the errors of the previous trees to

minimize prediction loss. XGBoost has a Boosting process where it starts with a weak model(a small decision tree) and then it gradually improves by reducing the residual errors of previous decision trees.To optimize the performance gradient descent approach is used in boosting process.

XGBoost works initially by making a prediction of taking the average of the target values, and then compute the difference (i.e., the residual error) between actual values and predictions. In order to improve accuracy, new decision trees are built with each one focused on correcting errors made by previous ones. This continues iteratively, further improving predictions. The major advantages of XGBoost is that it capture complex relationships providing high accuracy and also handles the missing data very well compared to other machine learning models.Also, its fast computing made possible by parallel processing greatly accelerates training,which makes it perfect for managing large amounts of data.

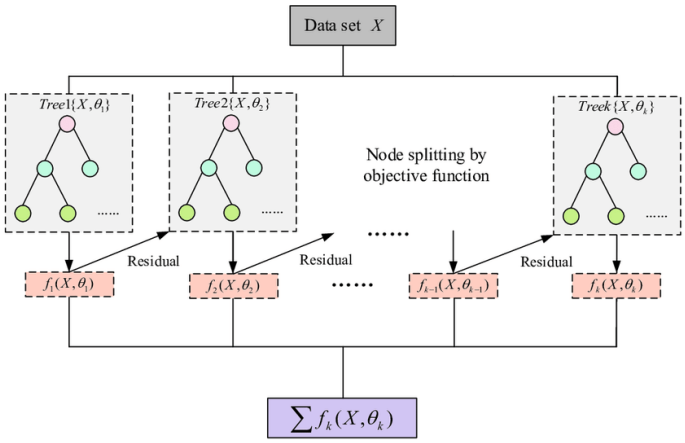

Figure 3.3: XGBoost (Extreme Gradient Boosting)

Furthermore, XGboost has different important hyperparameters,affecting the performance of the model. Some include learning rate, max-depth, Number of Trees (n-estimators). These hyperparameters should be properly tuned to get the best performance.

3.1.4 Artificial Neural Networks(ANN)

Artificial Neural Networks [4] represent a powerful class of machine learning models that have shown significant promise in software effort estimation tasks. These models mimic the information processing of biological neurons through interconnected nodes arranged in layered architectures. In typical implementations, an ANN consists of an input layer that receives project metrics, one or more hidden layers that transform these inputs through weighted connections and nonlinear activation functions, and an output layer that produces the effort prediction.

The true strength of ANNs lies in their ability to model complex, nonlinear relationships between various project characteristics and the resulting development effort. However, several practical challenges emerge when applying ANNs to effort estimation. The models typically demand large volumes of high-quality training data to achieve reliable performance, as insufficient data often leads to poor generalization on new projects. The training process itself can be computationally demanding, especially when working with deep network architectures containing numerous hidden layers.

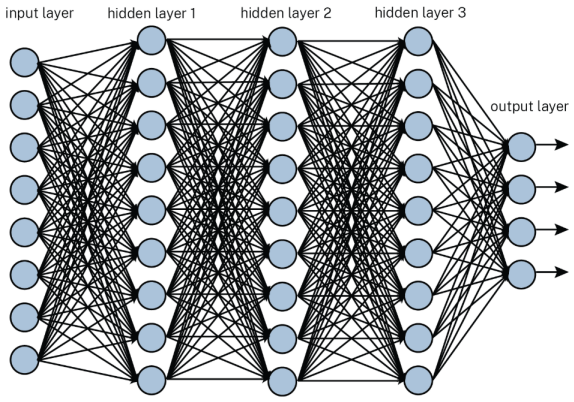

Figure 3.4: Artificial Neural Networks

3.1.5 Linear Regression

Multiple Linear Regression (MLR) [14] serves as one of the most fundamental yet valuable statistical approaches for software effort prediction.It extends simple linear regression by allowing multiple predictors, enabling more accurate and realistic modeling of complex systems.This method establishes a direct mathematical relationship between project characteristics (independent variables) and development effort (dependent variable) through a linear equation of the form:

$$\text{Effort} = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_n X_n + \varepsilon \tag{3.2}$$

In the context of software effort estimation, the β coefficients represent the importance or weight assigned to each project metric (X), where β_0 is the intercept and ε denotes the residual error or the portion of variability not explained by the model. Common predictors used in such estimations may include code size (e.g., lines of code), team experience, requirement complexity, and other measurable project features. The goal is to identify a linear model that best fits the data by minimizing the difference between the predicted and actual effort values.

3.1.6 K-Nearest Neighbors(KNN)

K-Nearest Neighbors (KNN) is a data-driven algorithm that relies on similarity between instances to make predictions [13]. Rather than creating an explicit model during training, KNN retains the full dataset and uses it during prediction by calculating the distance between the new input and all stored examples. The algorithm then selects the top 'k' closest examples and predicts the output based on their values, often using an average for regression tasks.

The estimated effort is then calculated by averaging the actual efforts of these nearest neighbors. This can be mathematically represented as:

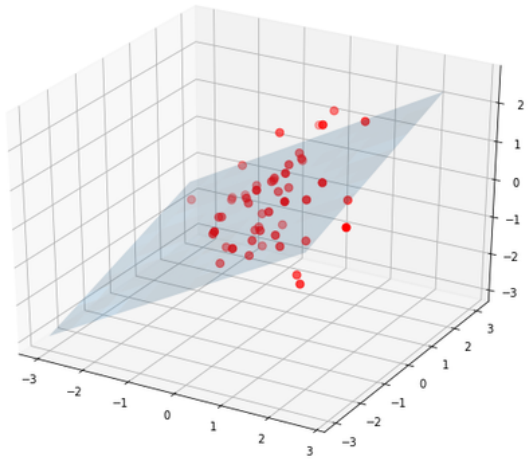

Figure 3.5: Multiple Linear Regression

$$\hat{y} = \frac{1}{k} \sum_{i=1}^{k} y_i \tag{3.3}$$

where $\hat{y}$ is the predicted effort, k is the number of nearest neighbors, and y_i represents the effort values of the i^{th} nearest neighbors. In the context of software effort estimation, KNN can be applied to forecast the development effort required for a new project by comparing it to previously completed projects. Factors such as code size, team skills, and project features are used to identify the most similar historical cases. The estimated effort is then derived from these comparable projects, allowing for flexible and intuitive predictions, especially when the underlying relationships are complex or non-linear.

3.1.7 Support Vector Regressor(SVR)

Support Vector Regressor (SVR) [6] is a regression technique based on Support Vector Machines (SVM). Unlike traditional regression methods that

aim to minimize prediction error, SVR tries to fit the best line (or curve) within a predefined margin of tolerance, known as epsilon (ε). The goal is to ensure that as many data points as possible fall within this margin, while still maintaining a flat and simple model.

In the context of software effort estimation, SVR is used to model the relationship between software project features—such as code size, number of developers, or system complexity—and the corresponding effort required. By learning from past projects, SVR can predict the effort for a new project, even in cases where the data is noisy or the relationship is non-linear.

3.1.8 Optimised Analogy Based Model

Analogy-Based Estimation (ABE) is a data-driven technique used to predict software development effort [15] .Fundamentally, Analogy-Based Estimation (ABE) is a variant of Case-Based Reasoning (CBR). It assumes that similar projects will have similar effort requirements. By comparing a new project to past, already-estimated projects (called analogies), the effort is inferred based on the most similar instances. The accuracy of this model is largely dependent on how similarity is measured and how analogies are weighted and aggregated.

A. Model Architecture and Parameter Optimization

ABE works by retrieving the k most similar historical projects to a target project using one or more similarity measures (e.g., Euclidean, Mahalanobis). The similarity is computed over project features, and each feature can be weighted differently to reflect its relevance. Once the nearest neighbors are identified, the model aggregates their effort values using strategies like mean, median, or inverse-distance weighting.However, defining optimal values for parameters such as:

- Feature weights
- Number of analogies (k)

- Similarity metrics
- Aggregation methods

is non-trivial. These parameters significantly influence model performance and are therefore subject to optimization.

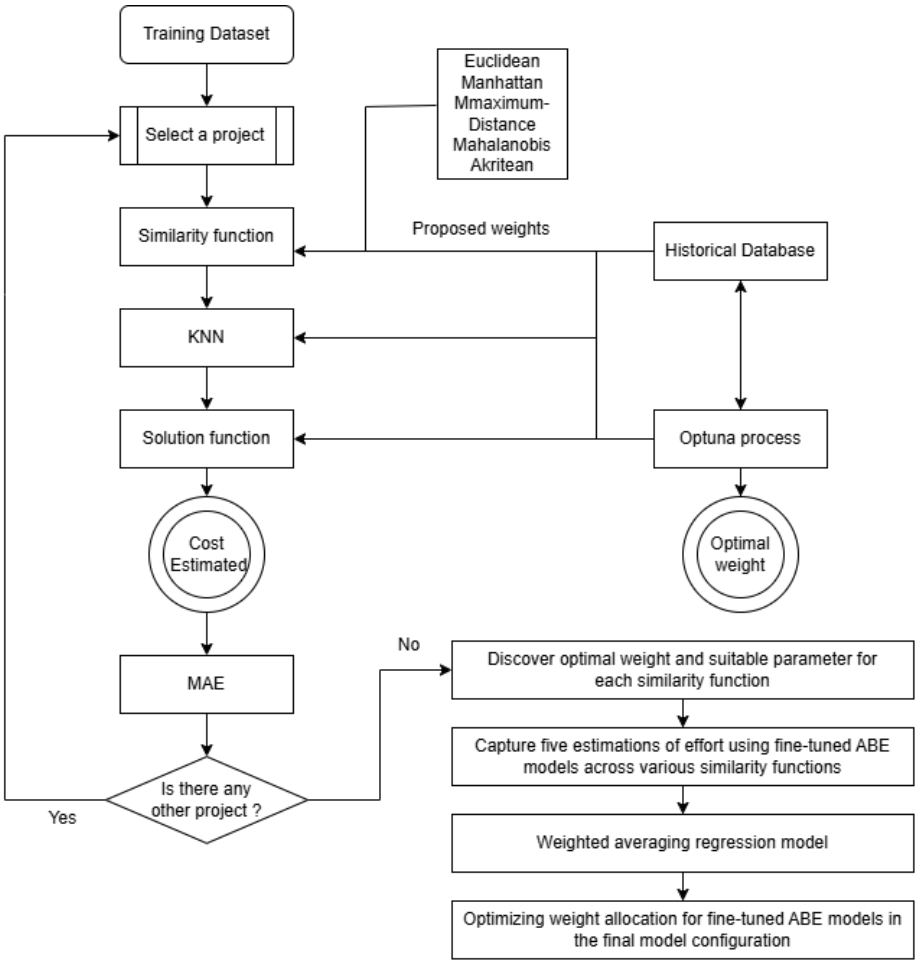

Figure 3.6: Working of Analogy Based Model

B. Optimisation with Optuna

Boosting algorithms, combined with automatic hyperparameter tuning using tools like Optuna, have also been employed to improve software effort estimation performance [8]. To intelligently select the best pa-

rameters, the model employs Optuna, an automatic hyperparameter optimisation framework. Optuna follows a define-by-run paradigm, where the search space is specified programmatically and can adapt dynamically based on prior results.

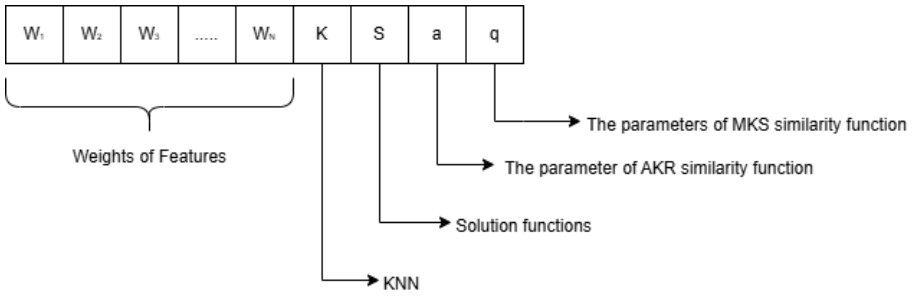

Figure 3.7: Parameters involved in ABE model

I. Trial Sampling and Search Strategy
At its core, Optuna uses a Tree-structured Parzen Estimator (TPE) for sampling. Unlike grid or random search, TPE models the distribution of successful and unsuccessful trials. For each parameter trial, it does the following steps:

- Optuna ranks previous trials based on performance and separates them into two groups using a threshold: good-performing and poor-performing trials.
- It builds two probability models using kernel density estimation — one from the good trials and one from the bad trials — to represent promising and less promising parameter regions.
- These models estimate the likelihood of various parameter values leading to good or poor outcomes.
- The algorithm computes the Expected Improvement (EI) by comparing the probability of success (good) to failure (bad) for candidate parameters.
- New parameter values are sampled from regions where the good-to-bad probability ratio is high, focusing on areas with the most potential for improvement.

- This targeted sampling strategy allows Optuna to explore the parameter space more efficiently than random or grid search.
- As more trials are completed, both models are updated dynamically, refining the search and accelerating convergence toward optimal solutions.

This allows for adaptive exploration, focusing more on areas of the parameter space likely to yield better performance.

II. Trial Execution and Objective Evaluation
Each trial in Optuna:

- Samples a unique set of parameter values,
- Applies these parameters to train an intermediate ABE model,
- Uses a validation subset (e.g., 30% of training data) to compute the Mean Absolute Error (MAE),
- Records the MAE as the trial's objective value.

Optuna then updates its probabilistic model to guide the search in future trials.

III. Early Stopping
To avoid wasting resources on unpromising trials, Optuna integrates a pruning mechanism. If a trial shows poor performance at intermediate stages (e.g., in early validation epochs), it is halted early. This ensures computational efficiency and allows more trials to be completed in the same time frame.

C. Final Model Configuration

After a predefined number of trials or when convergence is observed, the trial with the lowest validation MAE is selected as the optimal configuration. These best-found parameters are then used to run the final ABE model, which is evaluated on unseen test data. By tuning the model in this way, prediction accuracy and generalisability are significantly improved.

3.2 Heterogenous Ensemble Model

After the standalone models have been assessed, a heterogeneous ensemble model was modelled to combine the strengths of multiple different types of models. This ensemble model involves three individual models:

- **COCOMO-II**: A parametric model which estimate the effort based on project size and scale factors.
- **Case-Based Reasoning (CBR)**: A memory based method that estimates effort by retrieving and adapting from the similar past projects.
- **A Machine Learning Regressor**: One of SVR, Linear Regression, K-NN, ANN, or XGBoost is choosed.

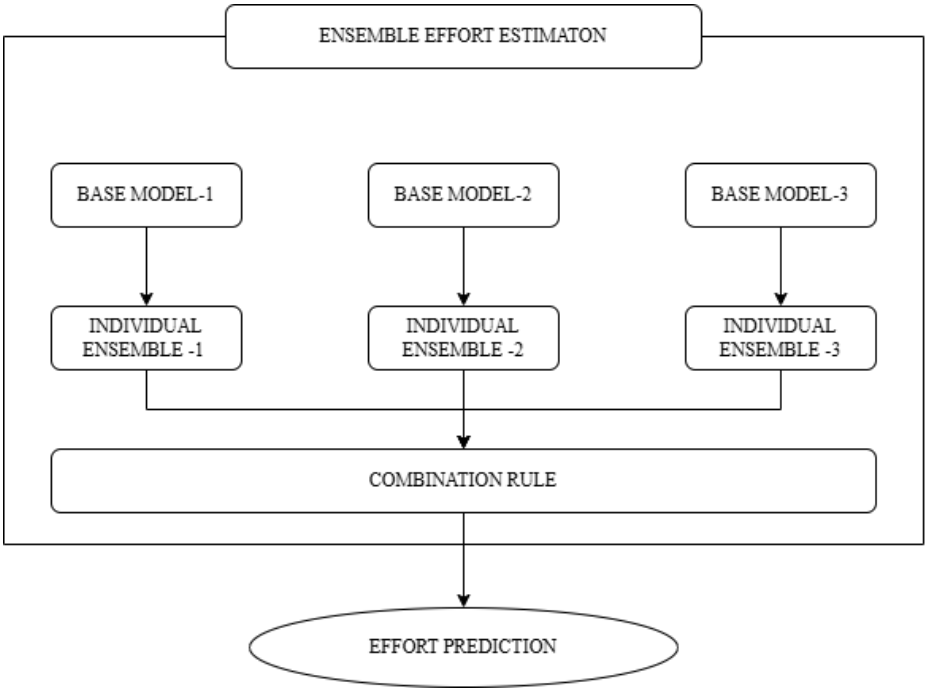

Figure 3.8: Heterogeneous Ensemble Model

The purpose of the ensemble method is to take advantage of the distinctive characteristics of each model, interpretability of algorithmic approaches (COCOMO-II) and the adaptability of machine learning models. They are

trained independently on the dataset, and their results (effort estimates) are subsequently combined with a combination rule to create a final effort estimation.

This technique is categorized as heterogeneous since it combines individually different type of models rather than using a multiple homogeneous models. Ensemble was tested with multiple data sets to verify its stability and generalization.

3.2.1 Combination Rule

To combine the predictions from the three models (COCOMO-II, CBR, and the choosed machine learning model) different combination rules were applied. These rules define how the individual predictions are merged to get a final effort estimation:

- **Linear Combination Rule:** In this approach, each output of a model is assigned with a weight, and the final prediction is predicted as the weighted sum of all three outputs of Individual Models. These weights can also be adjusted to increase the performance of model.

$$E_{\text{final}} = w_1 \cdot E_{\text{CBR}} + w_2 \cdot E_{\text{Cocomo}} + w_3 \cdot E_{\text{ML-approach}} \tag{3.4}$$

$$w_1 + w_2 + w_3 = 1 \tag{3.5}$$

- **Median Combination Rule:** In this approach, the median of the individual predictions is taken as the final prediction. This is useful when the model can produce outlier values, as the median is powerful for handling extreme values.

$$E_{\text{final}} = \text{median}(E_{\text{Cocomo}}, E_{\text{CBR}}, E_{\text{ML-approach}}) \tag{3.6}$$

Each combination rule was evaluated to determine the best combination rule. Each of these rules have advantages like the linear combination allows for controlled contribution from each model and the median rule offers robustness to handle inconsistent predictions.

4

Results

This chapter presents a comparative evaluation of the proposed ensemble model against various standalone models configured differently. To assess prediction accuracy, we employed several standard performance measures, including Mean Absolute Error (MAE), Mean Magnitude of Relative Error (MMRE), Median of MRE (MDMRE), and Prediction at 25% threshold (PRED(0.25)). The evaluation was performed using three well-known datasets commonly used in software effort estimation studies: NASA60, NASA93, and COCOMO81.

4.1 Dataset Description

In this project, we utilize three widely recognized datasets for software effort estimation: NASA60, NASA93, and COCOMO81. These datasets are extensively used in software engineering research and provide empirical data collected from real-world software projects. They are instrumental in evaluating and comparing the performance of various effort estimation techniques.

The NASA60 dataset contains 60 software projects compiled by NASA's Software Engineering Laboratory. It includes both numerical and categorical features that reflect project characteristics such as technical constraints, team experience, and development complexity. Each project record provides the actual development effort (in person-months) and project size mea-

sured in KLOC, making it valuable for traditional and data-driven estimation approaches.

The NASA93 dataset builds upon NASA60 and includes 93 software project entries. It offers a more comprehensive set of attributes that capture a wide range of factors influencing software development effort, such as team capabilities, application types, and platform characteristics. Due to its richer feature set and higher variability, NASA93 is often preferred for validating complex models like ensemble learning and optimization-driven estimators.

The COCOMO81 dataset originates from Barry Boehm's Constructive Cost Model (COCOMO), introduced in 1981. It consists of 63 project records, each including project size in KLOC, multiple cost drivers, and the actual effort. The dataset's effort values are derived using Boehm's parametric estimation formula, which incorporates scaling factors and multipliers. COCOMO81 serves as a benchmark for evaluating rule-based and parametric estimation models.

4.2 Evaluation Metrics

To evaluate the accuracy and robustness of the software effort estimation models used in this study, we adopted four widely recognized performance metrics: Mean Absolute Error (MAE), Mean Magnitude of Relative Error (MMRE), Median Magnitude of Relative Error (MdMRE), and Prediction Accuracy at 25% (Pred(0.25)). These metrics provide diverse insights into the model's performance across different datasets.

- **Mean Absolute Error (MAE)**: This metric represents the average of the absolute differences between predicted and actual effort values. It is calculated as:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|\hat{e}_i - e_i|$$

where $\hat{e}_i$ is the predicted effort, e_i is the actual effort, and n is the total

number of data points.

- **Mean Magnitude of Relative Error (MMRE)**: MMRE measures the average relative error across all predictions and is defined as:

$$MMRE = \frac{1}{n}\sum_{i=1}^{n}\frac{|\hat{e}_i - e_i|}{e_i}$$

- **Median Magnitude of Relative Error (MdMRE)**: MdMRE is the median of the relative errors and provides a robust measure against outliers:

$$MdMRE = \text{median}\left(\frac{|\hat{e}_i - e_i|}{e_i}\right)$$

- **Prediction Accuracy at 25% (Pred(0.25))**: This metric computes the percentage of predictions that fall within 25% of the actual effort:

$$Pred(0.25) = \frac{k}{n} \times 100$$

where k is the number of predictions satisfying $\frac{|\hat{e}_i - e_i|}{e_i} \leq 0.25$.

These evaluation criteria allow us to comprehensively compare the performance of different estimation approaches based on both accuracy and consistency.

4.3 Performance of Individual Standalone Models

Based on the experimental results shown in figure 4 and 5,XGBoost Regressor showed the optimal balance between accuracy and training times as shown in the figure.While ANN achieved the lowest MAE,but its significantly higher training times made it less practical with large datasets like Cocomo81.On the other hand, XGBoost was a more effective and scalable option because it consistently produced low MAE across all datasets while keeping training times manageable. While Linear Regression, even though

it was simple, did not do well, especially on the COCOMO81 dataset, showing that it is not good at capturing complex effort estimate patterns.Similarly KNN and SVR did not perform well when compared with XGBoost.

In conclusion, XGBooster emerged as the best choice for Ensembling with CBR and cocomoII due to ability to handle feature interactions,stop overfitting, and optimize computation through parallel processing. Thus, considering both accuracy and computational times, XGBoost was chosen as the final model for software effort estimation.

Table 4.1: Performance of Various Stand-alone models for NASA60

Models	Training Time (s)	MMRE	MAE	MDMRE	PRED(0.25)
SVR	0.1604	0.9163	265.5283	0.3960	0.3667
Linear Regression	0.1159	1.7822	249.3008	0.6457	0.2500
K-NN	0.1003	0.4803	159.8280	0.2873	0.4333
XGBoost Regressor	10.8870	0.3105	155.4475	0.1970	0.6000
ANN	72.0850	0.4069	144.6046	0.2546	0.4833
CBR	0.0031	0.4381	173.1817	0.3470	0.3833
COCOMO II	0.0002	0.2539	96.6304	0.1977	0.6500

Table 4.2: Performance of Various Stand-alone models for NASA93

Models	Training Time (s)	MMRE	MAE	MDMRE	PRED(0.25)
SVR	0.6864	0.9642	456.1414	0.5740	0.3011
Linear Regression	0.4145	2.6187	469.7880	0.6796	0.1505
K-NN	0.2912	1.5446	494.1114	0.7156	0.1720
XGBoost Regressor	15.9983	1.1594	395.8388	0.3580	0.3656
ANN	21.5327	1.5407	345.4534	0.7670	0.2258
CBR	0.0050	1.6340	571.9526	0.6962	0.2043
COCOMO II	0.0003	0.5937	470.0133	0.2709	0.4624

Table 4.3: Performance of Various Stand-alone models for Cocomo81

Models	Training Time (s)	MMRE	MAE	MDMRE	PRED(0.25)
SVR	0.1674	1.7206	612.2693	0.8245	0.1746
Linear Regression	0.1773	20.0269	958.4834	5.1893	0.0952
K-NN	0.0847	1.8015	580.0070	0.7481	0.0794
XGBoost Regressor	12.4247	1.3805	319.1951	0.7049	0.2063
ANN	85.9392	5.4754	687.9563	1.5318	0.1587
CBR	0.0032	1.8077	648.9275	0.7725	0.1111
COCOMO II	0.0002	0.3811	219.6751	0.2976	0.4286

4.4 Performance of Ensemble Model

In order to analyze how effectiveness of proposed model in estimating software effort, we developed and experimented with five combinations. Each of these combinations comprised COCOMO-II, Case-Based Reasoning (CBR), and a single machine learning approach - SVR, Linear Regression, K-NN, XGBoost Regressor, ANN. These are tested on three benchmark datasets—NASA93, COCOMO-II and NASA60 using performance metrics such as MAE (Mean Absolute Error), MMRE (Mean Magnitude of Relative Error), MDMRE (Median of MRE), and PRED(0.25). The goal was to determine which machine learning model improves the predictive power of the COCOMO-II and CBR models. The results are shown in the following Tables.

Table 4.4: Performance of Various Learning Models in CBR-COCOMOII-Based Ensemble for NASA60

Models	Training Time (s)	MMRE	MAE	MDMRE	PRED(0.25)
SVR	0.1637	0.3259	144.8552	0.2799	0.4667
Linear Regression	0.1193	0.3310	114.4340	0.2145	0.6167
K-NN	0.1036	0.3592	133.0375	0.2411	0.5167
XGBoost Regressor	10.8903	0.2853	129.8469	0.1937	0.5833
ANN	72.0883	0.2829	110.7127	0.2073	0.6000

Table 4.5: Performance of Various Learning Models in CBR-COCOMOII-Based Ensemble for NASA93

Models	Training Time (s)	MMRE	MAE	MDMRE	PRED(0.25)
SVR	0.6917	0.6844	365.7587	0.4120	0.3656
Linear Regression	0.4197	0.7252	312.4596	0.3939	0.3011
K-NN	0.2965	1.0735	428.1312	0.5643	0.2473
XGBoost Regressor	15.0036	0.7680	339.5386	0.3467	0.3763
ANN	21.5380	0.8504	321.7353	0.4929	0.3011

Table 4.6: Performance of Various Learning Models in CBR-COCOMOII-Based Ensemble for Cocomo81

Models	Training Time (s)	MMRE	MAE	MDMRE	PRED(0.25)
SVR	0.1708	1.1494	387.4852	0.6164	0.2063
Linear Regression	0.1807	0.7410	345.5308	0.3499	0.3651
K-NN	0.0881	1.4325	507.7392	0.6912	0.1587
XGBoost Regressor	12.4282	0.8143	284.4057	0.3853	0.2540
ANN	85.9426	0.9575	410.3751	0.3620	0.3333

The proposed CBR-COCOMOII-based ensemble process was reported

to have competitive performance when used with various learning models on NASA60, NASA93 and COCOMO81 datasets. The ANN and XGBoost Regressor consistently reported lower MMRE and MAE values, indicating higher estimation accuracy, especially on NASA60, but the training times of ANN is higher compared to XGBoost Regressor for most of the datasets. Linear Regression also reported good performance on MAE and PRED(0.25), while SVR and K-NN reported higher error rates on larger datasets like NASA93 and COCOMO81. Overall, the results indicate the versatility of the ensemble, and ANN and XGBoost were reported to be promising models to improve estimation precision in the CBR-COCOMOII framework.

4.5 Comparison Of Ensemble model with stand-alone models

To validate the heterogeneous ensemble model, we compared the results of the stand-alone models to the proposed heterogeneous ensemble model. Metrics such as MAE, MMRE, MDMRE, PRED(0.25) for stand-alone models of ML are compared to heterogeneous ensemble model which is a stand-alone model combined with CBR and COCOMO II.

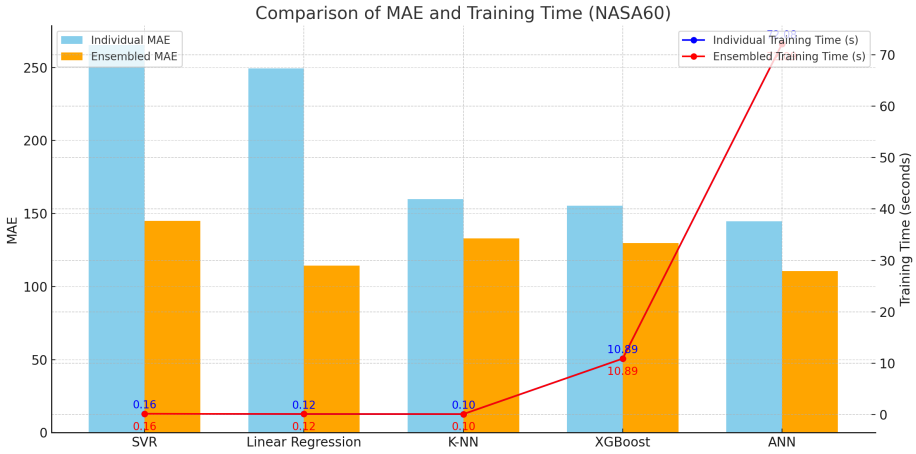

Figure 4.1: Comparison of MAE and Training Time (NASA60).

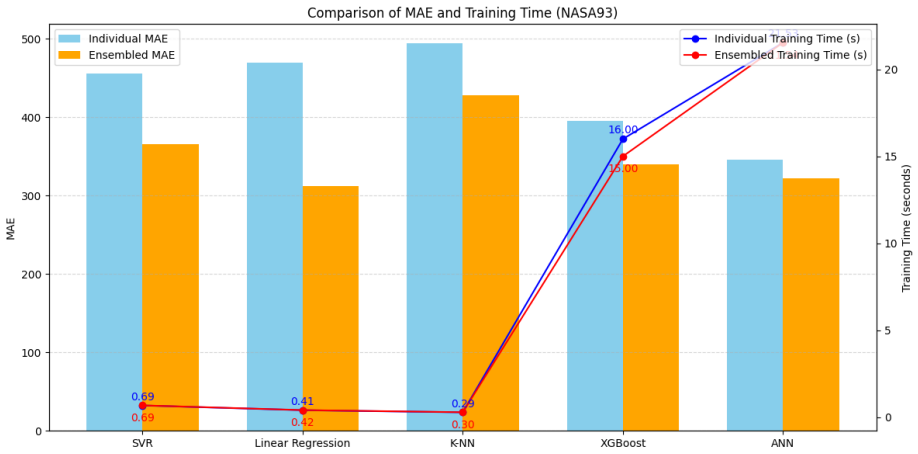

Figure 4.2: Comparison of MAE and Training Time (NASA93).

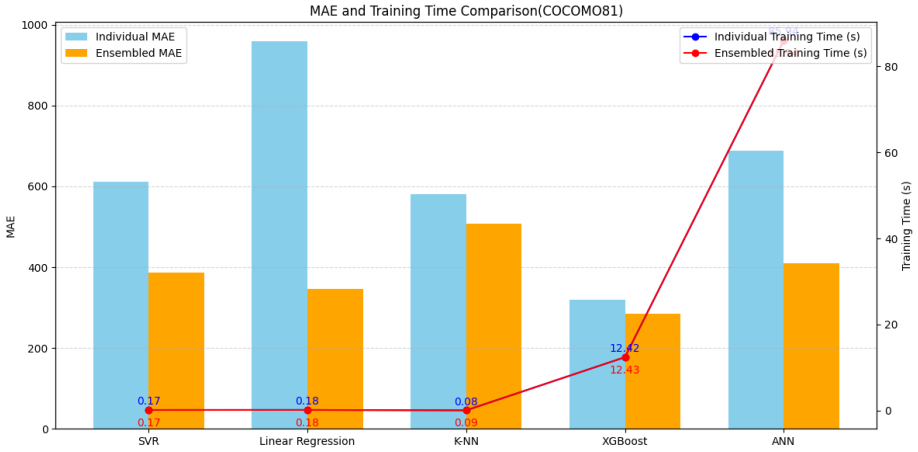

Figure 4.3: Comparison of MAE and Training Time (COCOMO81).

By the results we can say the heterogeneous ensemble model which contains a ML model performs better compared to that particular ML model used individually. The ensemble model produces better results than stand-alone model without any significant change in the training time. Results are compared for various datasets, the results remain same that is the ensemble model performs better than stand-alone model. By this we can say that the heterogeneous ensemble model outperforms stand-alone model.

4.6 Comparison between CBR and Optimised ABE model

Table 4.7: Results of Optimised ABE for NASA60

Model	MAE	MMRE	MdMRE	PRED(0.25)
CBR	173.1817	0.4381	0.3470	0.3833
Optimised ABE	166.8819	0.5199	0.3434	0.4167

Table 4.8: Results of Optimised ABE for NASA93

Model	MAE	MMRE	MdMRE	PRED(0.25)
CBR	571.9526	1.6340	0.6962	0.2043
Optimised ABE	456.1942	1.3146	0.4310	0.2903

Table 4.9: Results of Optimised ABE for COCOMO81

Model	MAE	MMRE	MdMRE	PRED(0.25)
CBR	648.9275	1.8077	0.7725	0.1111
Optimised ABE	620.7993	1.8775	0.9326	0.1111

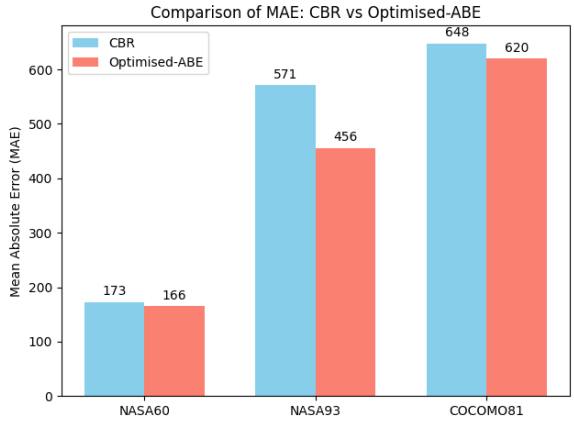

Figure 4.4: Comparison of MAE between CBR and Optimised ABE model).

The comparison between Case-Based Reasoning (CBR) and the Optimized Analogy-Based Estimation (Optimized-ABE) model reveals key differences in their performance across different datasets.

From the results, it is evident that Optimized-ABE consistently outperforms CBR in terms of prediction accuracy. For all the datasets considered,

Optimized-ABE shows a noticeable improvement in the Mean Absolute Error (MAE), indicating that the optimization process effectively enhances the model's predictive accuracy. This improvement highlights the strength of the Optimized-ABE model in refining the estimations when compared to the traditional CBR approach.

However, the trade-off for these improvements in accuracy is the increased training time required for Optimized-ABE. Unlike CBR, which is relatively quicker to train, Optimized-ABE necessitates additional computational resources due to the optimization procedures it undergoes.

5

Conclusion and Future Work

5.1 Conclusion

This study proposed a Heterogeneous Ensemble approach to estimate the effort required to complete a project by integrating COCOMO-II, Case-Based Reasoning (CBR), and a machine learning model such as Support Vector Regression (SVR), Linear Regression, K-Nearest Neighbors (KNN), Artificial Neural Networks (ANN), and XGBoost Regressor. After evaluating the standalone models, an ensemble method was developed to leverage the strengths of individual models by combining their properties, such as expert knowledge, historical similarity, and data-driven learning.

The ensemble model was tested with two types of combination rules: linear and median. The results revealed that the median combination rule performed better than the linear combination rule, as it was more sensitive to the distribution of weights. Among all the combinations tested, the COCOMO-II + CBR + XGBoost ensemble model, using the median combination rule, outperformed other configurations across all datasets.

Furthermore, the study compared Optimized Analogy-Based Estimation (Optimized-ABE) with the traditional CBR model. The results demonstrated that Optimized-ABE provided improved accuracy, delivering better Mean Absolute Error (MAE) scores across the datasets. However, this improvement in performance came at the cost of longer training times. While Optimized-ABE is more effective at estimating effort, its higher computa-

tional demand may limit its applicability in scenarios where training efficiency is a key concern.

These findings support the increasing use of Heterogeneous Ensemble models as a robust solution for software effort estimation in real-world project management. The integration of multiple models, particularly when combined with optimization techniques, shows promising potential for improving accuracy while balancing the trade-off between training time and model performance.

5.2 Future Work

In the continuation of the work,

- We wanted to improve the performance of case-based reasoning(CBR) by optimizing the hyperparameters.
- Also, integrate the optimized version in the ensembling process for better results.

Bibliography

[1] Syed Sarmad Ali, Jian Ren, Kui Zhang, Ji Wu, and Chao Liu. Heterogeneous ensemble model to optimize software effort estimation accuracy. *IEEE Access*, 11:27759–27792, 2023. 2.4

[2] Barry W Boehm. Software engineering economics. *IEEE transactions on Software Engineering*, (1):4–21, 1984. 2.1

[3] Barry W Boehm, Chris Abts, A Winsor Brown, Sunita Chulani, Bradford K Clark, Ellis Horowitz, Ray Madachy, Donald J Reifer, and Bert Steece. *Software cost estimation with COCOMO II*. Prentice Hall Press, 2009. 3.1.2

[4] Aggarwal Charu C. *Neural networks and deep learning: a textbook*. Springer, 2018. 3.1.4

[5] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016. 2.2, 3.1.3

[6] Harris Drucker, Christopher J Burges, Linda Kaufman, Alex Smola, and Vladimir Vapnik. Support vector regression machines. *Advances in neural information processing systems*, 9, 1996. 3.1.7

[7] Taghi Javdani Gandomani, Maedeh Dashti, Hazura Zulzalil, and Abu Bakar Md Sultan. Enhancing software effort estimation in the analogy-based approach through the combination of regression methods. *IEEE Access*, 2024. 2.3

[8] Maryam Hassanali, Mohammadreza Soltanaghaei, Taghi Javdani Gandomani, and Farsad Zamani Boroujeni. Software development effort estimation using boosting algorithms and automatic tuning of hyperparameters with optuna. *Journal of Software: Evolution and Process*, 36(9):e2665, 2024. **B.**

[9] Fred J Heemstra. Software cost estimation. *Information and software technology*, 34(10):627–639, 1992. 2.1

[10] Ali Idri, Mohamed Hosni, and Alain Abran. Systematic literature review of ensemble effort estimation. *Journal of Systems and Software*, 118:151–175, 2016. 2.4

[11] Tridas Mukhopadhyay, Steven S Vicinanza, and Michael J Prietula. Examining the feasibility of a case-based reasoning model for software effort estimation. *MIS quarterly*, pages 155–171, 1992. 2.3, 3.1.1

[12] Sarika Mustyala and Manjubala Bisi. Ensembling harmony search algorithm with case-based reasoning for software development effort estimation. *Cluster Computing*, 28(2):97, 2025. 2.3

[13] Leif E Peterson. K-nearest neighbor. *Scholarpedia*, 4(2):1883, 2009. 3.1.6

[14] Mark Tranmer and Mark Elliot. Multiple linear regression. *The Cathie Marsh Centre for Census and Survey Research (CCSR)*, 5(5):1–5, 2008. 3.1.5

[15] Fiona Walkerden and Ross Jeffery. An empirical study of analogy-based software effort estimation. *Empirical software engineering*, 4:135–158, 1999. 3.1.8

[16] Jianfeng Wen, Shixian Li, Zhiyong Lin, Yong Hu, and Changqin Huang. Systematic literature review of machine learning based software development effort estimation models. *Information and software technology*, 54(1):41–59, 2012. 2.2

[17] Dengsheng Wu, Jianping Li, and Chunbing Bao. Case-based reasoning with optimized weight derived by particle swarm optimization for software effort estimation. *Soft Computing*, 22:5299–5310, 2018. 2.3