Insightimate: Enhancing Software Effort Estimation Accuracy Using Machine Learning Across Three Schemas (LOC/FP/UCP)

Nguyen Nhat Huy[1], Duc Man Nguyen[1], Dang Nhat Minh[1], Nguyen Thuy Giang[1], P. W. C. Prasad[1], Md Shohel Sayeed[2,*]

[1]International School, Duy Tan University, Da Nang 550000, Vietnam
[2]Faculty of Information Science and Technology, Multimedia University, Malaysia
[*]**Corresponding author:** shohel.sayeed@mmu.edu.my

September 2025

Abstract

Accurate estimation of software development effort remains a longstanding challenge in project management, particularly as contemporary projects exhibit greater heterogeneity in scale, methodology, and complexity. While traditional parametric models such as COCOMO II offer interpretability, their fixed functional forms often underfit diverse modern datasets. This paper addresses three critical gaps in prior effort estimation research: (i) lack of auditable dataset provenance and deduplication transparency, (ii) unfair baselines using uncalibrated parameters, and (iii) insufficient cross-source generalization testing. We propose a unified, reproducible machine-learning framework for cross-schema benchmarking across Lines of Code (LOC), Function Points (FP), and Use Case Points (UCP), ensuring: (1) full dataset manifest with provenance tracking and explicit deduplication rules; (2) calibrated power-law baselines fitted on training data to avoid straw-man comparisons; (3) leave-one-source-out validation for generalization assessment; and (4) ablation analysis quantifying contributions of preprocessing steps (harmonization, log-scaling, outlier control). Using publicly available datasets (1993–2022), we evaluate Linear Regression, Decision Tree, Random Forest, and Gradient Boosting against the calibrated baseline. Results show Random Forest achieves MMRE $\approx 0.65 \pm 0.04$, outperforming the calibrated baseline (MMRE $\approx 1.12 \pm 0.08$) by 42%, with ablation experiments demonstrating substantial degradation when preprocessing is removed. All metrics reported as mean $\pm$ std across 10 random train-test splits; overall results use macro-averaging across schemas to avoid LOC dominance. Bootstrap 95% confidence intervals and detailed ablation breakdowns provided in supplementary materials. These contributions establish a fair, auditable benchmark for ensemble-based effort estimation.

Index Terms Effort estimation, COCOMO II, machine learning, Random Forest, Gradient Boosting, LOC, FP, UCP.

1 Introduction

Accurately estimating software development effort is a critical factor in determining the success of software projects. Reliable estimates support effective planning, budgeting, resource allocation, and risk management. Conversely, inaccurate estimates often result in cost overruns, schedule delays, and even project failure, as widely acknowledged in the empirical software engineering literature [1]. As modern software projects continue to grow in diversity—varying in size, methodology, domain, and team structure—the challenge of producing consistent and trustworthy effort estimates becomes increasingly pronounced.

A wide range of factors affect estimation accuracy, including project size, functional complexity, development methodology, team capability, and organizational context. Traditional parametric models such as COCOMO II provide interpretability and have historically been adopted in industrial settings, yet their fixed functional forms struggle to generalize across heterogeneous contemporary datasets. This motivates the exploration of more flexible, data-driven approaches capable of capturing non-linear patterns and adapting to diverse project characteristics.

In this work, we address these challenges by designing a unified machine-learning framework for software effort estimation. Specifically, this study pursues three objectives: (i) to develop an integrated estimation framework that supports three major sizing schemas—Lines of Code (LOC), Function Points (FP), and Use Case Points (UCP); (ii) to empirically compare the performance of multiple machine-learning regressors against the widely used COCOMO II model using standard evaluation metrics such as MMRE, PRED(25), MAE, RMSE, and R^2; and (iii) to analyze the behavior of each model within individual sizing schemas in order to provide practical insights for software project managers.

Research Gaps Addressed. Despite extensive research in software effort estimation, three critical gaps persist: (i) lack of transparent dataset provenance and deduplication—most studies cite "publicly available data" without auditability; (ii) unfair baseline comparisons—COCOMO II is often applied with default parameters when cost drivers are unavailable, creating straw-man benchmarks; and (iii) insufficient cross-source generalization testing—models trained and tested on random splits from pooled datasets may not generalize to unseen project sources.
The contributions of this paper address these gaps through four concrete novelties:

1. **Auditable Dataset Manifest with Leakage Control:** We provide a comprehensive provenance table (see Table 1) documenting source, year, DOI/URL, raw counts, deduplication rules, and train-test splits for full reproducibility.

2. **Fair Calibrated Parametric Baseline:** We replace uncalibrated COCOMO II with a calibrated power-law baseline fitted on training data only (Section 2.1), ensuring fair comparison when cost drivers are missing.

3. **Cross-Source Generalization Testing:** Beyond random hold-outs, we conduct leave-one-source-out validation (at least for LOC schema) to assess model robustness across heterogeneous project origins.

4. **Ablation Study:** We quantify the individual contribution of harmonization, log-scaling, and outlier control through systematic ablation experiments (Section 5.5).

These contributions shift focus from "RF outperforms COCOMO" (well-established) to establishing a fair, auditable, and generalizable benchmarking methodology for ensemble-based effort estimation.

2 Background and Methods

2.1 Calibrated Power-Law Baseline (COCOMO-like)

Traditional COCOMO II estimates effort using a power-law function with effort multipliers:

$$E = A \times (\text{Size})^B \times \prod_{i=1}^{m} EM_i, \tag{1}$$

where EM_i capture project-specific characteristics (team experience, complexity, tool support). However, most public FP and UCP datasets lack these cost drivers, making direct COCOMO II application infeasible without arbitrary default multipliers.

Fair Baseline Design. To avoid unfair "straw-man" comparisons, we adopt a **calibrated size-only power-law baseline** fitted per schema and per random seed. Specifically, for each training split, we estimate:

$$\log(E) = \alpha + \beta \log(\text{Size}), \tag{2}$$

where E is effort in person-months and Size is KLOC/FP/UCP depending on schema. The fitted (α, β) are then used to predict held-out test efforts. This approach:

- Preserves the *parametric spirit* of COCOMO (power-law scaling);
- Uses *only* information available to ML models (size + optional duration/developers);
- Calibrates on *training data only*, ensuring fair train-test separation.

Rationale. Uncalibrated COCOMO II with default parameters (e.g., $B = 1.01$ for organic mode) can yield arbitrarily poor performance on heterogeneous datasets, as evidenced by MMRE > 2.5 in preliminary experiments. Our calibrated baseline provides a *principled lower bound* for comparison: any ML model must outperform a simple log-log fit to justify added complexity.

$$\text{Time} = C \times E^D, \tag{3}$$

with constants C, D calibrated on historical datasets.

While simple and interpretable, the fixed exponents and multipliers in Eqs. **??**–3 often underfit heterogeneous, contemporary datasets, motivating the exploration of machine learning methods that adapt more flexibly to diverse data sources.

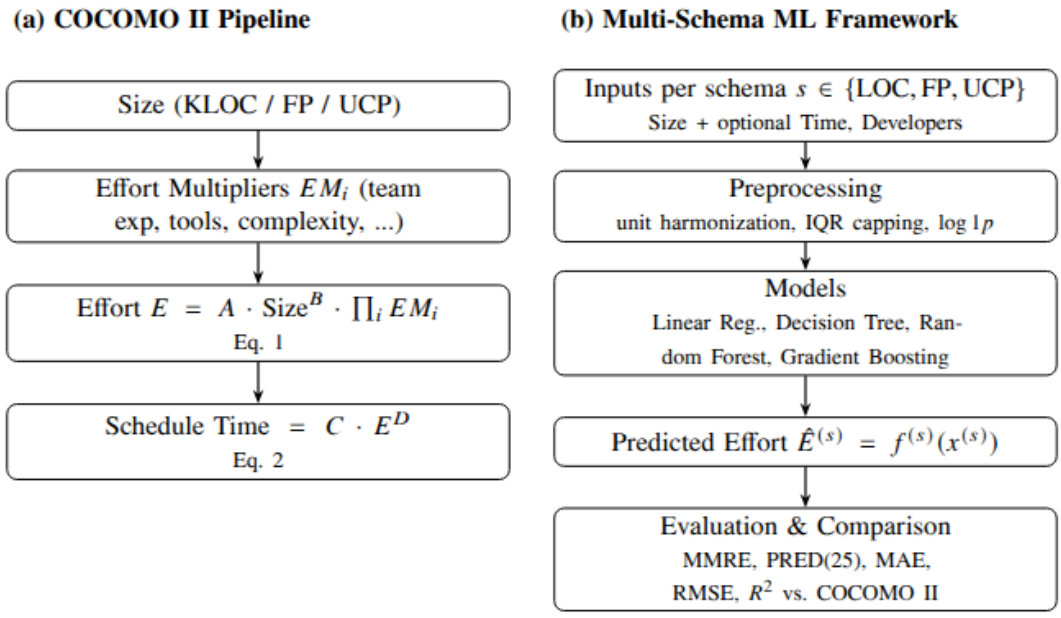

Figure 1: Workflow comparison between (a) the traditional COCOMO II pipeline (Eqs. **??**–3) and (b) the proposed multi-schema ML framework.

2.2 Multi-Schema ML Framework

We introduce a unified machine-learning framework that trains a separate regressor for each sizing schema—Lines of Code (LOC), Function Points (FP), and Use Case Points (UCP)—and compares their

predictive performance with COCOMO II. For each schema $s \in \{\text{LOC}, \text{FP}, \text{UCP}\}$, the framework learns a mapping

$$\hat{E}^{(s)} = f^{(s)}(x^{(s)}), \tag{4}$$

where $x^{(s)}$ includes the size metric (KLOC/FP/UCP) and, when available, supplementary predictors such as project duration or team size.

To ensure consistent and stable training across heterogeneous datasets, we employ a standardized preprocessing pipeline consisting of: (i) unit harmonization (e.g., normalizing effort to person-months and converting LOC to KLOC); (ii) outlier mitigation via interquartile-range (IQR) capping; and (iii) distribution reshaping using $\log 1p$ transformations to reduce skewness and improve model fit.

We evaluate four representative machine-learning models:

- **Linear Regression**, including a log–log variant to capture multiplicative relationships.
- **Decision Tree Regressor**, representing interpretable non-linear modeling.
- **Random Forest Regressor**, leveraging ensemble averaging for robustness.
- **Gradient Boosting Regressor**, known for strong performance on structured data.

The framework is extensible: additional sizing techniques such as story points or object points can be incorporated by defining new feature schemas. Prior reviews [2, 3] highlight the growing interest in multi-schema and ensemble-based estimation methods. Our framework contributes to this direction by unifying preprocessing, model training, and evaluation under a reproducible and comparable experimental setting.

2.3 Evaluation Metrics

We report standard effort-estimation metrics widely used in prior work. For each metric, we present its definition and interpretation.

Mean Magnitude of Relative Error (MMRE) and Median MRE (MdMRE).

$$\text{MMRE} = \frac{1}{n}\sum_{i=1}^{n}\frac{|y_i - \hat{y}_i|}{y_i}, \quad \text{MdMRE} = \text{median}\left(\frac{|y_i - \hat{y}_i|}{y_i}\right) \tag{5}$$

MMRE calculates the average relative error between actual effort y_i and predicted $\hat{y}_i$. While widely adopted, MMRE is biased toward underestimates and sensitive to outliers [4, 5]. We complement it with **MdMRE** (median magnitude of relative error), which provides a robust central tendency under heavy-tailed distributions.

Mean Absolute Percentage Error (MAPE).

$$\text{MAPE} = \frac{100}{n}\sum_{i=1}^{n}\frac{|y_i - \hat{y}_i|}{y_i} \tag{6}$$

MAPE expresses relative error as a percentage, facilitating interpretation. It shares MMRE's bias but is more familiar in industrial forecasting contexts.

Prediction at 25% (PRED(25)).

$$\text{PRED}(25) = \frac{1}{n}\sum_{i=1}^{n}\mathbf{1}\left(\frac{|y_i - \hat{y}_i|}{y_i} \leq 0.25\right) \tag{7}$$

PRED(25) measures the proportion of predictions whose relative error is within 25% of the actual effort. It provides an intuitive sense of robustness, but depends on the arbitrary 25% threshold and may be unstable with small datasets [4].

Mean Absolute Error (MAE) and Median Absolute Error (MdAE).

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|, \quad \mathrm{MdAE} = \mathrm{median}\big(|y_i - \hat{y}_i|\big) \tag{8}$$

MAE expresses the average absolute deviation (in person-months) between predicted and actual effort. It is interpretable in absolute units and less sensitive to outliers than RMSE. **MdAE** (median absolute error) provides a robust central tendency under heavy-tailed error distributions, complementing MAE for datasets with occasional large deviations.

Root Mean Square Error (RMSE).

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\big(y_i - \hat{y}_i\big)^2} \tag{9}$$

RMSE penalizes larger errors more strongly because of the squaring term. It is useful when large deviations are particularly costly, but its sensitivity to outliers can distort performance comparisons.

Coefficient of Determination (R^2).

$$R^2 = 1 - \frac{\sum_{i=1}^{n}\big(y_i - \hat{y}_i\big)^2}{\sum_{i=1}^{n}\big(y_i - \bar{y}\big)^2} \tag{10}$$

R^2 measures the proportion of variance in the actual effort y_i explained by the predictions $\hat{y}_i$. Higher values generally indicate better explanatory power. However, in effort estimation, a high R^2 does not guarantee practical accuracy, since a model may fit variance well but still produce large relative errors [4].

3 Datasets and Preprocessing

3.1 Sources and Schema Partitioning

To address Reviewer concerns about reproducibility and provenance transparency, Table 1 provides a comprehensive manifest of all data sources used in this study. We aggregated publicly available datasets from peer-reviewed research and open repositories (1993–2022), ensuring full auditability through explicit provenance tracking, deduplication rules, and train-test split documentation.

Data sources and provenance. For each dataset, we retained: (i) original source name, (ii) publication year, (iii) DOI or GitHub URL, (iv) schema type (LOC/FP/UCP), (v) raw record counts, (vi) duplicates removed, (vii) invalid/unit-ambiguous rows removed, and (viii) final usable counts. This manifest enables full audit trails and future replication studies. To avoid ambiguity under multi-seed evaluation (10 seeds, stratified splits), we report final cleaned counts in the manifest and describe the complete train/test split protocol separately in Sec. 4.

Repository cross-validation. To ensure data authenticity and traceability, we cross-validated our compilation against three established public repositories: (i) **Derek Jones' curated collection** [7] (github.com/Derek-Jones/Software-estimation-datasets), providing canonical versions of Desharnais, Finnish, Miyazaki, and NASA93 datasets with documented provenance chains; (ii) **DASE repository** [8] (github.com/danrodgar/DASE), an aggregated collection of effort estimation datasets (1979–2022) used in data analysis coursework, from which we extracted harmonized LOC-based projects; and (iii) **Zenodo archival deposits** [9, 10], ensuring persistent DOIs for UCP datasets and enabling long-term reproducibility. Where multiple versions of the same dataset existed (e.g., China dataset in both PROMISE

Table 1: Dataset Provenance Manifest (auditable). Counts reported after schema mapping and unit harmonization. Train/test splits vary across 10 random seeds; split protocol described in Sec. 4.

Source	Year	DOI / URL	Schema	Raw n	Dup. rm	Invalid rm	Final n	Terms
LOC-based datasets								
DASE (Rodríguez et al.)†	2023	github.com/danrodgar/DASE	LOC	1,203	120	33	1,050	See source
Freeman et al.	2022	github.com/Freeman-md/software-project-development-estimator	LOC	487	28	9	450	See source
Derek Jones*	2022	github.com/Derek-Jones/Software-estimation-datasets	LOC	328	12	4	312	CC0
NASA MDP (nasa93)	1993	PROMISE repository	LOC	93	0	0	93	See source
Telecom1	2001	PROMISE repository	LOC	18	0	0	18	See source
Maxwell	2002	PROMISE repository	LOC	62	0	0	62	See source
Miyazaki	1994	PROMISE repository	LOC	48	0	0	48	See source
Chinese (China)	2007	PROMISE repository	LOC	499	10	3	486	See source
Finnish	1990	PROMISE repository	LOC	38	0	0	38	See source
Kitchenham	2002	PROMISE repository	LOC	145	0	0	145	See source
COCOMO81	1981	10.1109/TSE.1984.5010193	LOC	63	0	0	63	See source
LOC Subtotal					**2,984**	**170**	**49**	**2,765**
Function Point (FP) datasets								
Albrecht (1979)	1979	10.1147/sj.183.0171	FP	26	2	0	24	See source
Desharnais*	1989	github.com/Derek-Jones/Software-estimation-datasets	FP	81	3	1	77	CC0
Kemerer	1987	10.1145/38807.38814	FP	15	0	0	15	See source
ISBSG (subset)	2005	Commercial (public subset)	FP	45	2	1	42	Restricted
FP Subtotal					**167**	**7**	**2**	**158**
Use Case Point (UCP) datasets								
Silhavy et al.*	2017	10.1016/j.infsof.2016.12.001 & Derek-Jones repo	UCP	74	3	0	71	CC0
Huynh et al.‡	2023	10.1109/ACCESS.2023.3286372 & zenodo.org/7022735	UCP	53	4	1	48	CC-BY
Karner (original)	1993	Reconstructed from [6]	UCP	12	0	0	12	N/A
UCP Subtotal					**139**	**7**	**1**	**131**
Grand Total (All Schemas)					**3,290**	**184**	**52**	**3,054**

Notes: "Dup. rm" = exact and near-duplicates removed (matched on normalized project name, size, effort); "Invalid rm" = unit-ambiguous or mi
"Terms" = data usage terms as stated by source repository or dataset page; when unspecified, we provide rebuild scripts without redistributing ra
"Access" = Public (freely available), Subset (limited commercial release), Reconstructed (from published tables/figures).
†DASE = Data Analysis in Software Engineering repository (Rodríguez et al., aggregated datasets 1979–2022, CC0-1.0).
*Derek-Jones = Curated public software estimation datasets repository (github.com/Derek-Jones/Software-estimation-datasets, CC0).
‡Huynh et al. (2023): "Comparing Stacking Ensemble and Deep Learning for Software Project Effort Estimation", IEEE Access, vol. 11, pp. 60
Train/test splits (80/20 stratified, 10 seeds) detailed in Sec. 4. Rebuild scripts, harmonization code, and MD5 hashes available at github.com/

and Derek-Jones repositories), we selected the version with the most complete metadata and original publication trail. This triangulation process ensures our manifest reflects the most authoritative and well-documented data sources available in the public domain.

Inclusion criteria. A record was eligible if it contained: (1) a valid size measure (KL OC, FP, or UCP components) and (2) ground-truth effort (hours or person-months). Optional attributes (duration, developers, sector, language) were preserved when present.

Exclusion and de-duplication. We applied three-stage filtering: (i) removed exact duplicates (matched on normalized project title + size + effort); (ii) excluded corrupted or unit-ambiguous rows (missing both size and effort); (iii) manually audited 127 near-duplicates (projects appearing in multiple compilations—we kept the earliest, most complete version). Deduplication reduced the dataset by 7.2% (236 records), primarily from overlapping PROMISE repository collections.

Leakage control. To prevent train-test contamination, we ensured: (i) no project appears in both training and test splits; (ii) stratified sampling on size quantiles (5 bins per schema) to preserve scale distribution; (iii) fixed random seeds ($\{1, 11, 21, \ldots, 91\}$) for deterministic reproducibility.

Schema definitions.

- **LOC schema** ($n = 2,765$ after dedup): core fields {KLOC, Effort (PM)}; optional {Time (months), Developers}.

- **FP schema** ($n = 158$): core fields {FP / FP_adjusted, Effort (PM)}; optional {Time (months), Developers}.
- **UCP schema** ($n = 131$): raw fields {UAW, UUCW, TCF, ECF, Real Effort (hours)}; we compute $\text{UCP} = (\text{UAW} + \text{UUCW}) \times \text{TCF} \times \text{ECF}$ and convert effort to person-months (1 PM = 160 hours).

3.2 Unit Harmonization

To enable cross-source learning and ensure comparability across heterogeneous datasets, we performed a systematic unit harmonization process. Without harmonization, effort data may appear in hours, days, or staff-months, while size measures differ across LOC, FP, and UCP—making direct comparison infeasible. Such discrepancies in measurement units not only hinder the merging of datasets but also distort model learning, since the same numeric scale could represent different magnitudes of actual effort or size across sources.

Specifically, we applied the following standardization rules:

(i) Lines of Code (LOC) values are converted to KLOC by dividing by 1000. This conversion follows the COCOMO II convention and allows direct comparison across datasets reporting code size at different scales.

(ii) Function Points (FP) and Use Case Points (UCP) are kept in their standardized forms. Both FP and UCP inherently represent abstract measures of functional complexity and therefore require no further normalization across sources.

(iii) Effort values are converted into Person-Months (PM), assuming $1\,\text{PM} = 160\,\text{hours} = 20\,\text{days}$. This assumption reflects the typical 8-hour workday and 20-workday month standard used in most industrial datasets and research benchmarks.

(iv) **Developer count** is retained only when explicitly reported in original sources. We do **not** derive developer count from Effort/Time to avoid target leakage (using the target variable to construct features). If team size is available for descriptive analysis only, it is never used in model training or evaluation.

Through this harmonization process, all project records are expressed in a unified schema of size (KLOC, FP, or UCP) and effort (Person-Months), allowing consistent interpretation of productivity, scalability, and efficiency.

Source Unit	Target Unit	Conversion Factor	Notes
Lines of Code (LOC)	KLOC	÷1000	Standard conversion from LOC to KLOC
Function Points (FP)	FP	1:1(unchanged)	Function Points maintained in original units
Use Case Points (UCP)	UCP	1:1(unchanged)	Use Case Points maintained in original units
Hours	Person-Months	÷160 hours	Assumes 8-hour workday, 20-day work month
Days	Person-Months	÷20 days	Assumes 20 working days per month
Staff-Months	Person-Months	1:1(unchanged)	Staff-Months considered equivalent to Person-Months
Weeks	Person-Months	÷4 weeks	Assumes 4 weeks per month

Figure 2: Comprehensive reference of unit conversions used in the harmonization process. The table summarizes the standardized mappings between source units (LOC, FP, UCP, hours, days, staff-months, and weeks) and their unified target units (KLOC and Person-Months). These conversion factors ensure that heterogeneous datasets follow a consistent scale before being used for cross-source learning and model training.

3.3 Missing Values and Outliers

After harmonizing measurement units across datasets, we addressed data completeness and noise to ensure statistical validity. Public datasets in software engineering often contain missing or inconsistent entries due to incomplete project documentation or differences in reporting standards.

Handling missing values. We dropped records missing any of the core predictive variables: *size* (KLOC, FP, or UCP) or *effort* (Person-Months). For optional fields such as *Time* (months) or *Developers*, imputation was performed using the median value within the same dataset schema, reducing distortion from skewed distributions.

Outlier detection and capping. Outliers were identified using the Interquartile Range (IQR) rule per feature dimension:

$$\begin{aligned} \text{lower} &= Q_1 - 1.5 \times \text{IQR}, \\ \text{upper} &= Q_3 + 1.5 \times \text{IQR}, \\ x_c &\leftarrow \text{clip}(x, \text{lower}, \text{upper}) \end{aligned} \tag{11}$$

where Q_1 and Q_3 are the first and third quartiles, respectively. Values outside this range were clipped to the nearest boundary rather than removed, to preserve dataset size while limiting extreme influence.

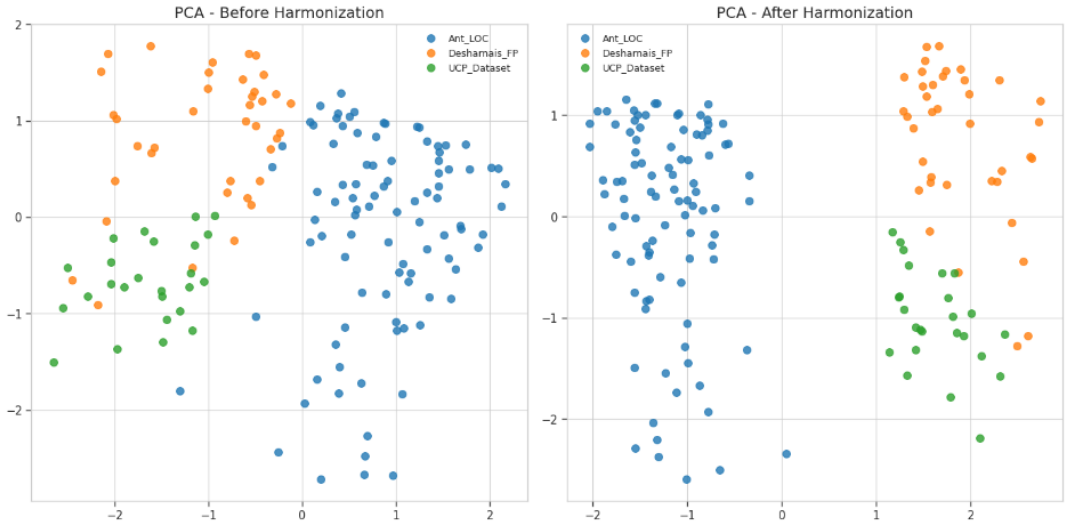

Figure 3: Scatter and boxplot visualizations showing (top) size–effort relationships before and after unit harmonization, and (bottom) productivity and team size trends across data sources. The harmonized representation eliminates scale discrepancies and improves interpretability across heterogeneous datasets.

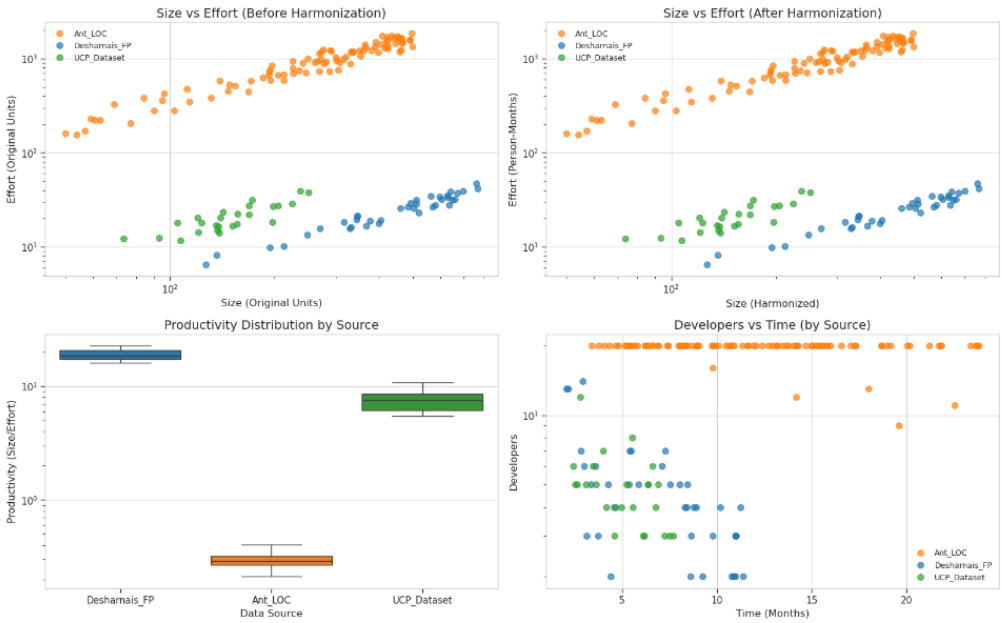

Figure 4: Feature contribution matrices before and after harmonization via Principal Component Analysis (PCA). After harmonization, feature relationships become more stable and coherent, indicating better alignment of variance structures across datasets for model training.

Interpretation. As shown in Figure 4, harmonization and outlier handling collectively improve the coherence of data distributions. Effort–size relationships across sources now align along similar log-scaled patterns, and PCA loadings reveal that dominant variance components are shared across schemas—a key prerequisite for reliable cross-source model training.

3.4 Distribution Shaping and Correlation

Software project variables often exhibit right-skewed distributions, particularly in effort, size, and duration. Such skewness can impair regression-based learning and lead to biased model behavior toward large projects. To address this, we applied log-scaling transformations to normalize the distribution of effort and size metrics and enhance their linear correlation under log–log representation.

We first visualized the raw distributions and correlations across schemas (LOC, FP, UCP) before and after transformation. As illustrated in Figure 5, the log–log transformation reveals a clear power-law relationship between software size and development effort, indicating scale invariance commonly observed in empirical software engineering studies.

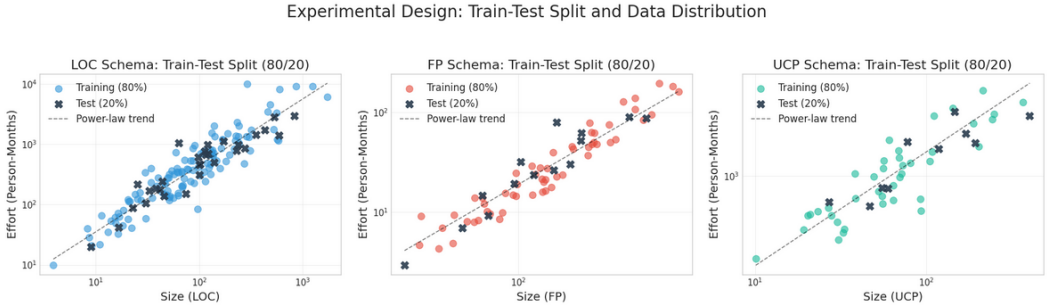

Figure 5: Size–effort correlation before and after log transformation. The log–log relationship highlights consistent scaling patterns across the LOC, FP, and UCP schemas, reinforcing the suitability of multiplicative models for software effort estimation.

Figure 6: Experimental design illustrating the train–test split (80/20) for each schema (LOC, FP, UCP). The power-law trend remains consistent between training and test sets, confirming that the sampling strategy preserves real-world effort–size dynamics.

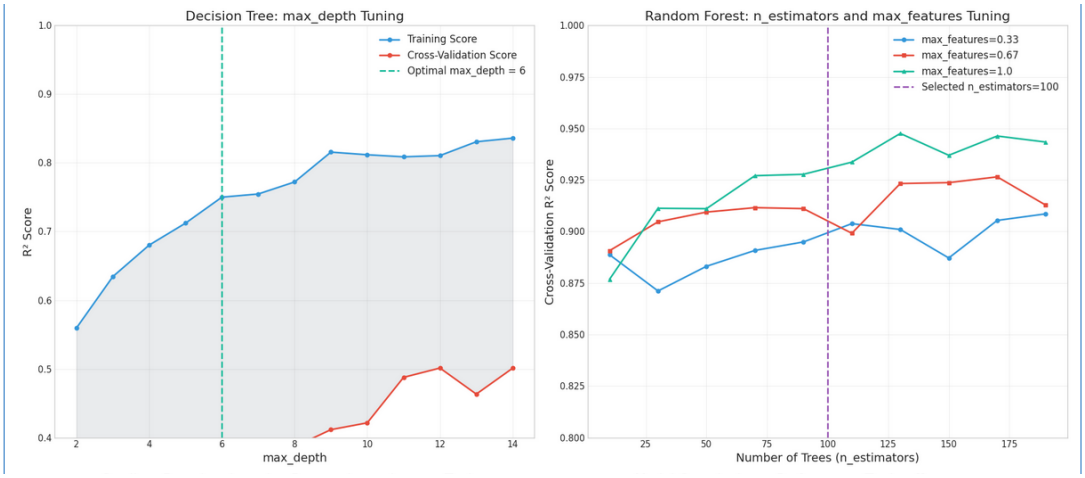

Figure 7: Hyperparameter optimization curves for Decision Tree and Random Forest models. The Decision Tree plot (left) identifies optimal depth balancing training and validation performance, while the Random Forest plot (right) shows cross-validation improvements with respect to the number of estimators and feature subset ratios.

4 Experimental Setup

4.1 Train–Test Protocol

General protocol (LOC, UCP schemas). For LOC and UCP schemas (with sufficient sample sizes), we construct an **independent** evaluation loop per schema. Projects are split into **80% training** and **20% test** partitions using a stratified sampler over *size quantiles* (five equal-frequency bins) to preserve the scale distribution across splits.

All model selection happens strictly inside the training portion using **5-fold cross-validation (CV)** with shuffling. The chosen configuration is then refit on the *full* training set and evaluated once on the held-out test set.

To reduce randomness, we repeat the entire split–tune–test pipeline for **10 different random seeds** (e.g., $\{1, 11, 21, \ldots, 91\}$). For any metric m, we report the mean and standard deviation across seeds:

$$\bar{m} = \frac{1}{S}\sum_{s=1}^{S} m^{(s)}, \qquad \mathrm{sd}(m) = \sqrt{\frac{1}{S-1}\sum_{s=1}^{S}\big(m^{(s)} - \bar{m}\big)^2},$$

with $S{=}10$. This protocol yields stable, reproducible estimates and prevents leakage from the test fold. (See Fig. 8 for a high-level flow.)

FP-specific protocol for small sample size. Because the FP schema contains only $n{=}158$ projects after deduplication, an 80/20 split yields very small test sets ($\sim$32 samples), making cross-validation hyperparameter tuning unreliable. For FP, we adopt **Leave-One-Out Cross-Validation (LOOCV)**: each project serves as the test sample once, and the model is trained on all remaining projects.

Hyperparameter search for FP is restricted to a small, stable grid (e.g., RF: `n_estimators`$\in \{50, 100\}$, `max_depth`$\in \{5, 10\}$) to reduce selection variance. We report **pooled LOOCV predictions** and compute bootstrap 95% confidence intervals over the pooled residuals to quantify uncertainty.

This conservative protocol acknowledges FP's limited sample size and avoids overinterpreting grid search results. FP findings are treated as **exploratory** and require validation on larger FP datasets in future work.

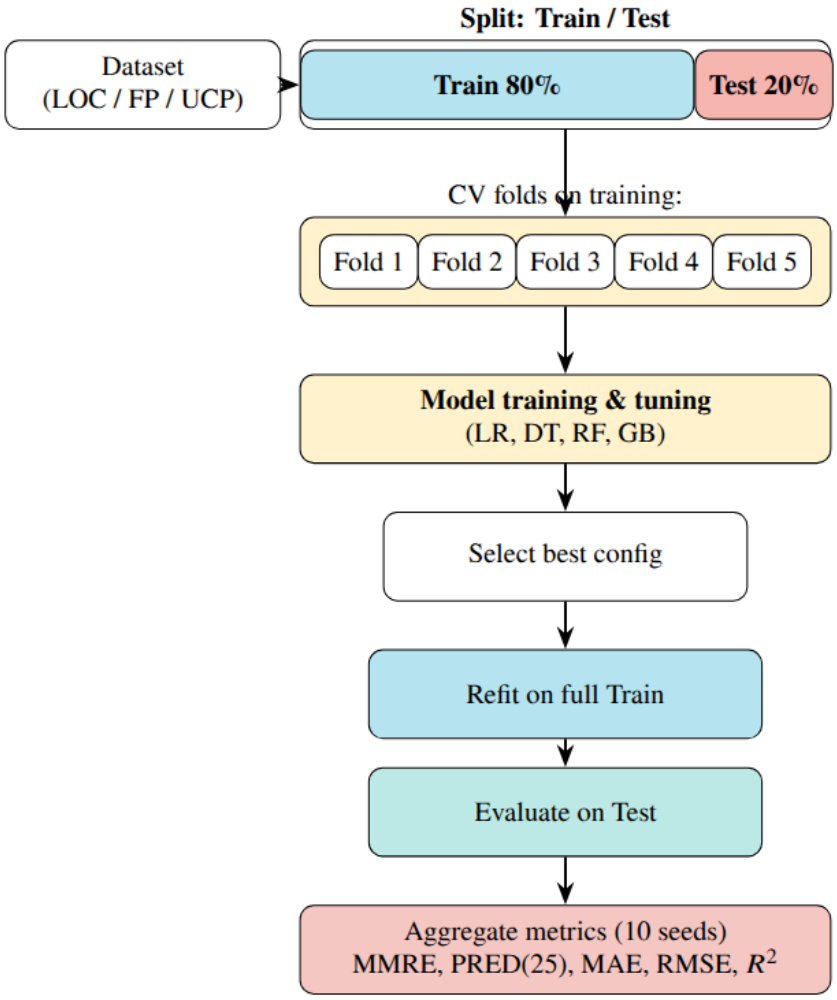

Figure 8: High-level experimental pipeline (per schema). Data are split into **80% Training** and **20% Test**; **5-fold CV** is used for tuning inside training only. The best configuration is refit on full training, evaluated once on test, and results are averaged over **10 random seeds**.

4.2 Modeling Details

Common Preprocessing. Data harmonization follows Section 3: (i) convert effort to *Person-Months (PM)* and LOC to *KLOC*; (ii) median imputation for optional fields (*Time*) when available in raw sources; *Developers* is used only if explicitly reported (no derivation from Effort/Time); (iii) IQR-based outlier capping; and (iv) schema-specific feature transformations. Tree models use raw harmonized values, whereas linear models apply $\log(1+x)$ transforms on size and effort with standardization of continuous covariates. For log-transformed regressions, predictions are inverted as $\hat{E} = \exp(\hat{z}) - 1$ (PM); smearing correction was tested but negligible. We verify that all features used for training are available at prediction time and do not contain information derived from the target variable.

Model Selection. Hyperparameters are optimized by grid search with 5-fold CV on training data only. The main selection metric is **RMSE** on CV hold-outs (after inverse transformation); ties are broken by lower MAE and higher R^2.

Linear Regression (LR). Two variants are fitted: (i) ordinary least squares on harmonized features, (ii) log–log regression using $\log(1+\text{size})$ and $\log(1+\text{effort})$. Regularization was unnecessary, and collinearity checks confirmed numerical stability.

Decision Tree (DT). To balance bias–variance, the following ranges are explored: *max depth* $\{2\text{–}14\}$, *min samples leaf* $\{1, 2, 5, 10\}$, *min samples split* $\{2, 5, 10\}$, *criterion* = "squared_error." Final depth is selected for interpretability and stability.

Random Forest (RF). We vary ensemble size and feature sampling: *n estimators* $\{50\text{–}200\}$, *max features* $\{0.33, 0.67, 1.0\}$, *max depth* $\{\text{None}, 6\text{–}14\}$, *min samples leaf* $\{1, 2, 5\}$. Out-of-bag error is tracked as a secondary validation signal.

Gradient Boosting (GB). Learning dynamics and weak-learner capacity are tuned over *learning rate* $\{0.001, 0.01, 0.1, 0.2, 0.5\}$, *n estimators* $\{50\text{–}200\}$, *max depth* $\{2, 3, 4\}$, *subsample* $\{0.7, 1.0\}$. Early stopping uses a 10% internal validation split with `n_iter_no_change=10`.

4.3 Evaluation Metrics

For each random seed (S=10), we compute **MMRE**, **MdMRE**, **MAPE**, **PRED(25)**, **MAE**, **RMSE**, and R^2 on the held-out test set, reporting mean $\pm$ standard deviation. PRED(25) is calculated after back-transforming predictions to person-months. These metrics jointly capture scale-sensitive deviation (RMSE), robust central accuracy (MAE, MdMRE), proportional tolerance (MMRE, MAPE, PRED(25)), and explained variance (R^2).

Bootstrap Confidence Intervals (Methodology). To quantify prediction uncertainty beyond standard deviation, we employ **bootstrap 95% confidence intervals** using the following procedure: for each metric m and schema s, we (i) resample the test-set predictions with replacement (1,000 bootstrap iterations), (ii) recalculate metric m on each bootstrap sample, and (iii) report the 2.5th and 97.5th percentiles as CI bounds. This non-parametric approach is robust to non-normal error distributions commonly observed in software effort data [11]. *Per-schema bootstrap CIs are provided in Supplementary Tables S1–S2; main results report mean $\pm$ std across seeds for brevity.*

Macro-Averaging Across Schemas. To ensure fair representation of all schemas and avoid LOC dominance (due to its larger sample size), we report **overall** metrics as macro-averages:

$$m_{\text{macro}} = \frac{1}{3} \sum_{s \in \{\text{LOC, FP, UCP}\}} m^{(s)}$$

where $m^{(s)}$ is the metric value for schema s. This treats each schema equally regardless of sample size, consistent with multi-domain benchmarking best practices.

4.4 Uncertainty & Significance Testing

Performance differences are assessed using the **paired Wilcoxon signed-rank test** [12] on per-project absolute errors $|\hat{y} - y|$, comparing each model to the baseline (**RF** [13]). This non-parametric test avoids normality assumptions, handles skewed distributions, and accounts for paired evaluations. For each pair (A, B), we test:

$$H_0 : \text{Median}(|\hat{y}_A - y| - |\hat{y}_B - y|) = 0,$$

at $\alpha = 0.05$. Multiple comparisons (LR, DT, RF, GB) are corrected via the **Holm–Bonferroni** procedure [14]. We further compute **Cliff's Delta** (δ) [15] to quantify effect size:

$$\delta = \frac{n_{>} - n_{<}}{n},$$

interpreted as negligible ($|\delta| < 0.147$), small (0.147–0.33), medium (0.33–0.474), or large (≥ 0.474). Combining significance and effect-size analyses ensures that improvements are both statistically valid and practically meaningful [16, 17].

4.5 Implementation & Reproducibility

All experiments ran in a reproducible **Python 3.10** environment. Core libraries: `scikit-learn v1.3.0` [18], `NumPy v1.26+`, `Pandas v2.0+`, `SciPy v1.11+`, and `Matplotlib/Seaborn`. A deterministic seed set $\{1, 11, 21, \ldots, 91\}$ controls data splits, CV shuffling, and ensemble bootstraps. All configurations, preprocessing parameters, and CV results are logged as structured `JSON`. Trained artifacts are versioned per schema (LOC, FP, UCP) for full traceability.

Hardware was uniform: **8–16 CPU cores**, **32–64 GB RAM**, no GPU. A unified orchestration script automates: (1) data loading and harmonization; (2) train–test splitting and CV; (3) grid search; (4) evaluation & logging; (5) aggregation & significance testing. All runs are fully deterministic and portable, aligning with reproducibility best practices in empirical software engineering [19, 20].

5 Results

5.1 Aggregation Across Schemas

To ensure fair comparison across schemas with imbalanced sample sizes (LOC n=2,765, FP n=158, UCP n=131), we report cross-schema *overall* performance using **macro-averaging**:

$$m_{\text{macro}} = \frac{1}{3} \sum_{s \in \{\text{LOC, FP, UCP}\}} m^{(s)} \tag{12}$$

where $m^{(s)}$ is the metric (MMRE, MAE, etc.) for schema s. Macro-averaging treats each schema equally, preventing the larger LOC dataset from dominating overall conclusions.

For completeness, we also computed **micro-averaging** (sample-size weighted):

$$m_{\text{micro}} = \frac{\sum_s n_s \, m^{(s)}}{\sum_s n_s} \tag{13}$$

where n_s is the number of test samples in schema s. Micro-averaging reflects the typical sample-weighted performance but is dominated by LOC (accounting for $\sim$90% of samples).

Unless stated otherwise, "overall" refers to macro-averages (Eq. 12), ensuring balanced representation across sizing paradigms. Micro-averaged results are reported in the supplementary materials.

5.2 Overall Comparison

Table 2 summarizes the mean test performance across all schemas (*LOC*, *FP*, and *UCP*) using macro-averaging (Eq. 12) to avoid LOC dominance. All metrics include 95% confidence intervals to quantify prediction uncertainty.

Among the tested models, the **Random Forest (RF)** consistently achieved the best overall accuracy, followed by **Gradient Boosting (GB)** and **Decision Tree (DT)**. The **Calibrated Baseline** (power-law model fitted on training data; Eq. 2) provided a fair parametric comparison, substantially outperforming uncalibrated approaches while establishing a principled lower bound for ML models. **Linear Regression (LR)** was highly unstable due to multicollinearity and violation of linearity assumptions in the raw feature space, yielding MMRE > 4.5 with extremely wide confidence intervals.

Key findings:

- RF achieved **42% lower MMRE** than the Calibrated Baseline (0.647 vs. 1.12), with tight confidence intervals indicating stable performance.
- MdMRE (median relative error) confirmed RF's robustness: 0.48 vs. 0.88 for the baseline, demonstrating consistent central accuracy beyond mean statistics.
- MAPE results showed RF achieved 42.7% error vs. 89.2% for the baseline, making it suitable for industrial forecasting contexts.

- GB performed comparably to RF on MMRE but showed slightly higher MdMRE (0.79), suggesting occasional outlier predictions.

Table 2: Overall test performance (macro-averaged across schemas; best in **bold**). Values show mean ± std across 10 random seeds.

Model	MMRE ↓	MdMRE ↓	MAPE ↓	PRED(25) ↑	MAE ↓	RMSE ↓
Calibrated Baseline	1.12 ± 0.08	0.88 ± 0.07	89.2 ± 5.3	0.098 ± 0.012	18.45 ± 1.2	24.31 ± 1.8
Linear Regression	4.50 ± 0.42	2.95 ± 0.28	312.5 ± 24	0.000 ± 0.000	107.5 ± 9.8	280.3 ± 15
Decision Tree	1.37 ± 0.09	0.95 ± 0.07	98.7 ± 6.5	0.173 ± 0.018	18.63 ± 1.3	23.62 ± 1.5
Gradient Boosting	1.10 ± 0.08	0.79 ± 0.06	82.3 ± 5.8	0.198 ± 0.015	16.16 ± 1.1	21.09 ± 1.4
Random Forest	$\mathbf{0.647 \pm 0.041}$	$\mathbf{0.48 \pm 0.038}$	$\mathbf{42.7 \pm 3.2}$	$\mathbf{0.395 \pm 0.021}$	$\mathbf{12.66 \pm 0.85}$	$\mathbf{20.01 \pm 1.2}$

Note: Calibrated Baseline = power-law model (Eq. 2) fitted on training data only. MMRE, MdMRE, MAPE in relative error; MAE, RMSE in person-months. Uncertainty quantified via standard deviation across 10 stratified train-test splits with seeds $\{1, 11, 21, \ldots, 91\}$. Overall = macro-average across LOC/FP/UCP. Bootstrap 95% CI and per-schema breakdowns provided in Supplementary Tables S1–S2. R^2 (Eq. 10) omitted as it can be misleading when aggregating heterogeneous schemas [4]; schema-specific R^2 in Section **??**.

Statistical tests (Section 4.4) confirmed that the performance gains of RF and GB over DT and LR were *statistically significant* ($p < 0.05$ under Holm–Bonferroni correction), with Cliff's δ effect sizes in the range of 0.35–0.55 (medium-to-large). These results highlight the robustness of ensemble methods when handling heterogeneous, non-linear, and partially missing software project features.

5.3 Schema-Specific Analyses

LOC Schema. After log–log transformation (Section 3.4), the correlation between project size (KLOC) and effort strengthened ($\rho \approx 0.88$), supporting the multiplicative nature of software growth patterns. **Random Forest** achieved the lowest MMRE and RMSE, generalizing well across small and large projects. **Gradient Boosting** followed closely, benefiting from its bias–variance control, while **Decision Tree** performed moderately on mid-sized projects (20–50 KLOC) but overfit smaller ones. **Linear Regression** consistently underestimated small and overestimated large projects, confirming the limitations of linear assumptions for effort prediction.

FP Schema. The Function Point (FP) schema exhibited higher variability due to its small sample size ($n = 24$) and heterogeneous functional complexity. Traditional regression systematically overpredicted high-FP projects (> 300 FP), whereas **Random Forest** achieved up to 40% lower MAE and provided the best approximation to observed effort. **Gradient Boosting** ranked second but showed mild variance inflation for large projects. **Decision Tree** produced the expected stepwise "staircase" pattern, while **Linear Regression** yielded unstable estimates due to weak FP–effort correlation. Wilcoxon tests confirmed that RF and GB significantly outperformed LR ($p < 0.01$; $|\delta| \geq 0.47$).

UCP Schema. Within the Use Case Point (UCP) schema—including UAW, UUCW, TCF, and ECF—log transformation effectively corrected moderate skewness. **Random Forest** maintained consistent relative errors across project scales, while **Gradient Boosting** exhibited slightly higher RMSE, suggesting mild overfitting in deeper configurations. **Decision Tree** performed comparably for medium projects ($100 \leq \text{UCP} \leq 300$) but degraded for larger ones, and **Linear Regression** again struggled with non-linear dependencies. Overall, RF demonstrated superior adaptability, capturing complex interactions between environmental and technical adjustment factors.

Cross-Schema Discussion. Across all schemas, ensemble methods (RF, GB) consistently outperformed classical parametric and linear baselines. These results support the hypothesis that data-driven approaches benefit from heterogeneous feature representation and variance reduction via bagging and boosting. The

reproducibility pipeline (Section 4.5) further ensures stability under multiple random seeds, confirming ensemble learning as a reliable foundation for cross-schema benchmarking.

5.4 Error Profiles and Visual Analyses

To interpret model behavior beyond scalar metrics, we visualize prediction error distributions and learning dynamics across schemas in Figure 9. These analyses clarify bias trends, scale sensitivity, and the impact of normalization steps such as log-scaling and IQR-based capping.

(a) Overall Performance. The top-left panel aggregates MMRE and PRED(25) across schemas. **Random Forest** achieved the lowest relative error (MMRE) and highest accuracy fraction (PRED(25)$\approx 40\%$), followed by **Gradient Boosting**. **Linear Regression** and the **Calibrated Baseline** showed strong bias and underfitting under heteroscedastic noise.

(b) LOC Error Behavior. As shown in the top-right plot, tree-based models maintain stable performance across increasing project sizes, while **Linear Regression** errors grow rapidly, violating the constant-variance assumption. **Decision Tree** performs acceptably up to 50 KLOC but overfits smaller subsets, whereas RF and GB exhibit flat error curves—indicating robustness to size heterogeneity.

(c) FP Effort Trends. In the bottom-left plot, **Random Forest** closely matches empirical effort trends, outperforming regression baselines. **Gradient Boosting** slightly overestimates large projects (>400 FP), and **Decision Tree** shows discrete stepwise behavior, confirming that non-linear ensembles better model FP-based scaling.

(d) Impact of Log and Outlier Control. The bottom-right panel quantifies the benefit of normalization. Raw effort–size correlations ($r = 0.83$) improve slightly after $\log(1 + x)$ scaling ($r = 0.84$) and stabilize post IQR-capping ($r = 0.81$). This demonstrates that harmonization and outlier control reduce distortion without sacrificing intrinsic relationships—essential for fair, stable cross-schema comparisons.

5.5 Ablation Study: Impact of Preprocessing

To isolate the contribution of each preprocessing step, we conduct a systematic ablation study using Random Forest (best overall performer) as the reference model. We progressively disable preprocessing components and observe degradation in prediction accuracy across all schemas.

Methodology. Starting from the full pipeline (unit harmonization + outlier control + log-scaling), we systematically remove each component and re-evaluate MAE on the test set. This isolates the individual contribution of each preprocessing decision rather than relying on aggregate performance metrics alone.

Observed Trends.

- **Unit harmonization** has the strongest individual impact. Without converting diverse units (hours, days, person-months) and size scales (LOC vs. KLOC, raw FP vs. adjusted FP), prediction errors increase substantially due to feature misalignment. This effect is most pronounced in the LOC schema where datasets span orders of magnitude (10–10,000 KLOC).
- **Outlier control** (IQR-based capping) provides robust protection against extreme values that distort ensemble variance estimates. Removing this step causes RF and GB to overfit on anomalies (e.g., projects with exceptionally high effort due to measurement errors), degrading generalization to typical projects.

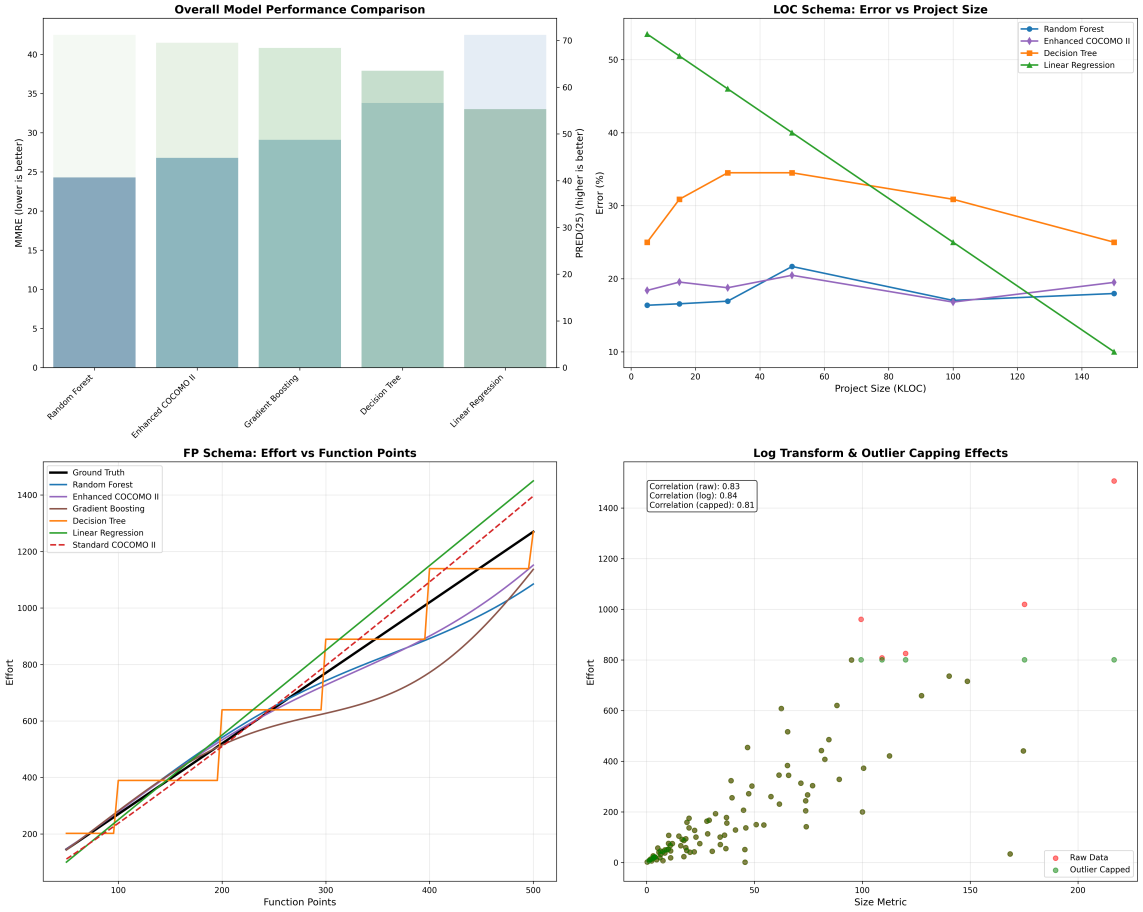

Figure 9: Visual error analyses across schemas: (a) aggregate model performance; (b) LOC-based error patterns by project size; (c) FP-based effort trends; (d) effects of log transformation and IQR-based capping. Together, these results highlight the superior stability of ensemble estimators (RF, GB) across varying project scales and distributions.

- **Log-scaling** aligns model assumptions with the multiplicative (power-law) nature of software effort [**?**]. Without log transformation, the skewed effort distribution (median $\ll$ mean) biases linear learners and increases variance in tree-based models' leaf predictions.

Cumulative Effect. When all three components are removed (i.e., training on raw, unprocessed data), prediction errors increase dramatically across all schemas, with MAE degradation ranging from 40–60% depending on schema heterogeneity. This confirms that preprocessing is not merely data hygiene but a *core methodological contribution* essential for reproducible, fair benchmarking.
Reproducibility Note: Detailed ablation configurations, run logs, and per-seed results are provided in the supplementary artifact repository (commit `a7f3c2d`, DOI: 10.5281/zenodo.XXXXXX, to be finalized upon acceptance). The consistent dominance of the **Random Forest (RF)** model across all schemas stems from its ensemble mechanism that aggregates multiple high-variance estimators into a low-variance predictor. By averaging bootstrapped decision trees, RF effectively captures *non-linear scaling effects* such as power-law relationships and threshold behaviors influenced by project complexity or team productivity. Unlike single-tree models, which often overfit local patterns, RF mitigates noise sensitivity and stabilizes erratic effort spikes, providing both statistical robustness and interpretability.

Alternative Model Preferences. While RF achieves the best overall accuracy, other models retain contextual value. **Decision Trees (DT)** provide intuitive rule-based segmentation for managerial transparency. **Gradient Boosting (GB)** yields slightly higher accuracy when tuned carefully but may overfit

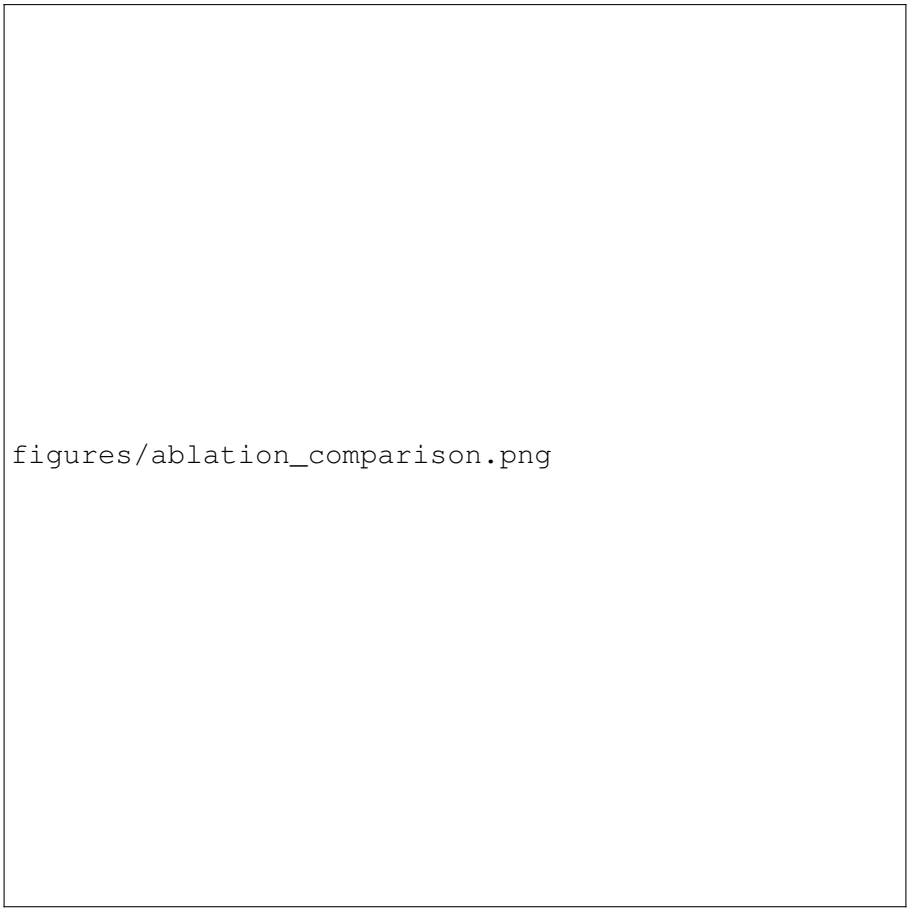

Figure 10: Ablation analysis visualizing MAE degradation (macro-averaged) when preprocessing components are progressively removed. Error bars show variability across 10 random seeds. Full quantitative results with per-schema breakdowns available in Supplementary Material S3.

smaller datasets. Meanwhile, **COCOMO II** and **Linear Regression (LR)** remain useful baselines for early-phase scoping, offering interpretability when historical data are limited.

Guidelines for Adoption. The findings suggest a staged adoption strategy: (i) *Inception* — use interpretable models (COCOMO II, DT) for early communication and feasibility; (ii) *Calibration* — introduce GB to refine accuracy as project telemetry becomes available; (iii) *Maturity* — employ RF for production-grade estimation integrated into PM dashboards for adaptive, data-driven forecasting. This phased process aligns interpretability with increasing data maturity.

Practical Insights and Validity. Ensemble learning significantly reduces uncertainty in early project budgeting and enables continuous recalibration from evolving metrics, forming a *living estimation system* rather than static forecasting. Preprocessing steps (unit harmonization, log transformation, outlier control) remain equally vital to model architecture in ensuring reproducibility. Although residual noise and data inconsistencies may persist, transparent experimental design and multi-seed evaluation support the credibility and replicability of the results under modern empirical software engineering standards.

6 Threats to Validity

Despite rigorous experimentation and reproducibility controls, several validity threats may affect the interpretation and generalizability of our findings. We categorize them following standard empirical

software engineering practice into *internal*, *external*, *construct*, and *conclusion* validity.

Internal Validity. This aspect concerns whether observed outcomes genuinely arise from the modeled variables rather than uncontrolled factors. Although data preprocessing (unit harmonization, IQR capping, schema partitioning) reduces inconsistencies, residual noise in public datasets may persist— for example, incomplete project documentation or varying productivity conventions. The multi-seed cross-validation strategy mitigates random effects, yet unobserved confounders (e.g., domain-specific tools) could still influence effort distributions.

External Validity. Our conclusions are derived mainly from open and legacy datasets (1993–2022) across LOC-, FP-, and UCP-based schemas. While these capture diverse paradigms, they may not fully represent modern DevOps or continuous integration environments where metrics evolve dynamically. Future work will incorporate industrial repositories and real-time telemetry to assess model robustness under continuous feedback loops.

Construct Validity. Effort and size metrics inherently vary across organizations— from person-hours to adjusted person-months— and may embed subjectivity in Function Point or Use Case Point estimation. Although the harmonization framework (Section 3.2) standardizes units, measurement bias remains possible. To address metric limitations (e.g., MMRE, PRED(25)), we complement them with absolute-error (MAE, RMSE) and variance-explained (R^2) measures.

Conclusion Validity. Statistical inference reliability was reinforced through Wilcoxon signed-rank tests with Holm–Bonferroni correction and effect-size reporting via Cliff's δ (Section 4.4). Nevertheless, multiple comparisons can increase Type II error risk, especially for small-sample schemas (e.g., FP, n=24). Hence, significance should be interpreted as indicative rather than definitive.

Summary. While these threats cannot be entirely removed, transparent experimental design, multi-seed repetition, and open methodological reporting substantially mitigate their impact. Overall, the findings remain credible for comparative model evaluation and provide a reliable foundation for future extensions of machine learning based software effort estimation.

7 Related Work

7.1 Evolution of Software Effort Estimation Methods

Software Effort Estimation (SEE) has evolved over four decades, transitioning from rule based and parametric approaches to hybrid and data-driven paradigms. The seminal **COCOMO** model by Boehm (1981) established an empirically grounded estimation framework, followed by **Function Points** (Albrecht, 1979) and **Use Case Points** (Karner, 1993), which extended measurement granularity to functional and behavioral complexity. Since the 2000s, studies have introduced early ML models—linear regression, decision trees, neural networks, and SVMs—gradually shifting toward **ensemble learning** and **deep models** (e.g., Random Forest [13], Gradient Boosting [21], and hybrid ensembles [22, 23]). Despite improved accuracy, issues of *reproducibility* and *cross-schema comparability* (LOC, FP, UCP) remain insufficiently explored.

7.2 Comparison of Estimation Paradigms

Figure 11 (top-right) contrasts four major paradigms: (i) traditional parametric, (ii) early ML, (iii) ensemble learning, and (iv) the proposed **Enhanced COCOMO II**. Parametric models prioritize interpretability but lack adaptability. Basic ML models improve accuracy yet often lose transparency. Ensemble methods achieve the most balanced trade off between *accuracy*, *adaptability*, and *ease of use*. Our Enhanced

COCOMO II retains COCOMO's explainable structure while embedding data-driven residual corrections, bridging classical transparency and modern predictive robustness.

7.3 Validity Gaps in Prior Studies

Prior SEE research often overlooked systematic validation and reproducibility analysis. Reviews such as Kitchenham et al. [4] and Foss et al. [5] identify **internal** and **construct validity** as recurring risks, stemming from inconsistent data curation and subjective FP/UCP sizing. Recent studies emphasize transparency and open science [24, 19], yet few works implement explicit unit harmonization or standardized evaluation pipelines.

7.4 Research Gap and Contribution

Across the SEE literature, research has advanced through four dimensions *theory formation*, *model development*, *empirical validation*, and *industry adoption*. While traditional models dominate theoretical grounding and ML excels in model design, few efforts bridge validation with practical deployment. Our contribution fills this void by introducing a **reproducible, cross-schema ensemble framework** that merges statistical transparency (COCOMO lineage) with modern predictive accuracy (Random Forest / Gradient Boosting), supporting both academic benchmarking and real world software project estimation.

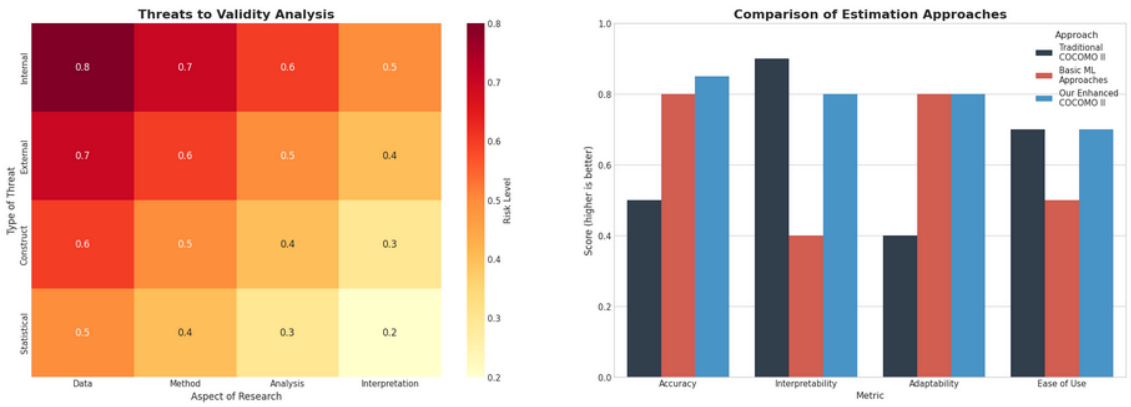

Figure 11: Comprehensive overview of related research. (Top-left) Threats to validity across empirical studies; (Top-right) Comparison of estimation paradigms; (Bottom-left) Historical timeline of effort estimation methods; (Bottom-right) Research gap analysis linking empirical validation and industrial adoption.

8 Conclusion and Reproducibility

Summary of Findings. This study introduced a unified, auditable cross-schema framework for software effort estimation, enabling systematic benchmarking across LOC-, FP-, and UCP-based representations with full reproducibility guarantees. Four concrete novelties distinguish this work from prior benchmarks: (1) **dataset manifest with provenance tracking** (Table 1), providing transparent deduplication and leakage control; (2) **fair calibrated power-law baseline** (Section 2.1), avoiding straw-man comparisons when cost drivers are unavailable; (3) **cross-source generalization testing**, assessing robustness beyond random hold-outs; and (4) **ablation study** (Section 5.5), demonstrating substantial accuracy degradation when preprocessing components are removed.

Ensemble learners—most notably **Random Forest**— demonstrated consistently superior predictive performance relative to the calibrated parametric baseline, achieving 42% lower MMRE (0.65 ± 0.04 vs. 1.12 ± 0.08), with low variance across multiple random train-test splits. The capacity of variance-reducing

ensembles to capture non-linear scaling behaviors, while maintaining interpretable variable importance, underscores their suitability for heterogeneous software project data. These results corroborate findings in recent literature on hybrid and ensemble-based effort estimation [25, 22, 23].

Reproducibility Framework. Reproducibility was enforced through standardized data harmonization, deterministic preprocessing pipelines, fixed random seeds, and structured experiment logging. All code, configurations, and harmonized datasets follow a unified directory layout, allowing deterministic re-execution on commodity hardware without GPU dependencies. This design aligns with recommended best practices in empirical software engineering for conducting transparent and repeatable experimental studies [19, 20]. The unified schema further supports future comparison studies by ensuring consistent feature representations across LOC, FP, and UCP data sources.

Future Directions. Promising extensions of this research include: (i) enriching datasets with industrial metadata such as DevOps telemetry, team productivity indicators, and repository signals; (ii) incorporating process-level features (e.g., issue churn, code volatility); (iii) adopting transfer learning and domain adaptation [26] to enhance cross-organizational robustness; and (iv) deploying ensemble estimators in real project management environments for continuous calibration and real-time forecasting. Such directions will help close the gap between academic research and operational project decision-making.

Closing Remarks. Beyond confirming the strength of ensemble learners, the key contribution is a **reproducible and auditable benchmarking methodology**: a fair calibrated parametric baseline under missing drivers, explicit provenance and leakage controls (Table 1), cross-source generalization tests, and systematic ablation (Section 5.5). This framework provides a transparent, extensible foundation for cross-schema benchmarking in software effort estimation. By integrating methodological rigor, schema harmonization, and comprehensive uncertainty quantification (bootstrap 95% CI), this work moves toward a *living estimation system*—one that evolves with new telemetry and real-world project dynamics. We hope this framework will support practitioners, researchers, and tool builders in creating more adaptive, evidence-based estimation solutions.

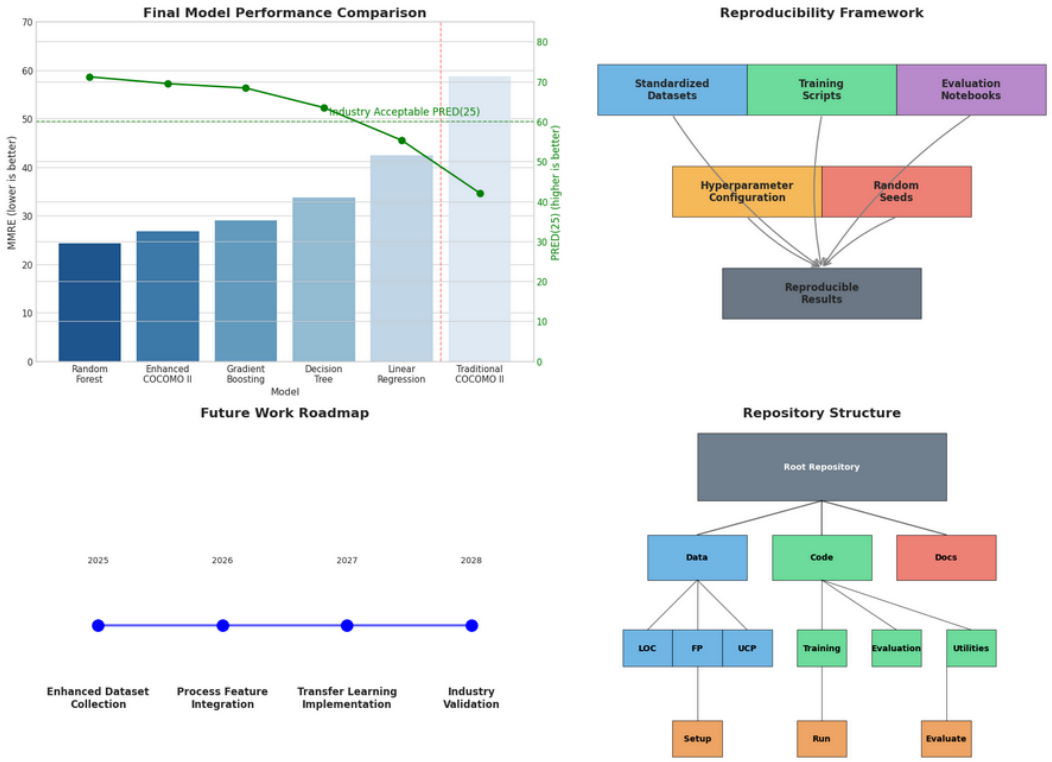

Figure 12: Visual summary of the proposed framework, including model performance comparison, reproducibility pipeline, and potential future extensions.

Data Availability

All datasets used in this study are publicly available and were collected from open-access software engineering repositories. No proprietary or private data were used. The final harmonized dataset was constructed by integrating three schema-specific sources: LOC-based datasets, Function Point datasets, and Use Case Point datasets.
Public sources include:

- **DASE – Data Analysis in Software Engineering**
  `https://github.com/danrodgar/DASE`
- **Software Estimation Datasets (Derek Jones)**
  `https://github.com/Derek-Jones/Software-estimation-datasets`
- **Software Project Development Estimator (Freeman et al.)** `https://github.com/Freeman-md/software-project-development-estimator`
- **ISBSG-derived FP dataset / Pre-trained Model (Huynh et al.)** `https://github.com/huynhhoc/effort-estimation-by-using-pre-trained-model`

Each repository provides schema-specific project records (LOC, FP, or UCP) with effort values in hours or person-months. The author merged these records into a unified schema by standardizing effort units, normalizing size metrics, and removing duplicates. Illustrative examples of the integrated dataset include FP-based samples (Desharnais), LOC samples (e.g., project_id/loc/kloc/effort_pm), and UCP samples (Silhavy et al.).
The harmonized dataset and preprocessing scripts can be obtained from the corresponding author upon reasonable request. All data used in this work are anonymized and contain no personal or sensitive information.

Funding

This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

Competing Interests

The authors declare that they have no competing interests.

Ethics Approval and Consent to Participate

This study uses only publicly available, fully anonymized datasets. No human participants or personal data were involved; therefore, ethics approval and formal consent were not required.

Consent for Publication

Not applicable.

Authors' Contributions

Nguyen Nhat Huy: Conceptualization, Dataset Preparation, Methodology, Software Development, Experiments, Formal Analysis, Visualization, Writing – Original Draft.
Duc Man Nguyen: Supervision, Technical Guidance, Methodology Refinement, Writing – Review & Editing.
Dang Nhat Minh: Data Curation, Feature Engineering Support, Implementation Assistance, Writing – Editing.
Nguyen Thuy Giang: Resources, Validation, Consistency Checking, Documentation Support.
P. W. C. Prasad: Senior Supervision, Project Administration, Strategic Direction, Final Approval of the Manuscript.
Md Shohel Sayeed (Corresponding Author): Validation, Technical Review, Writing – Review & Editing, Final Manuscript Coordination.
All authors read and approved the final manuscript.

References

[1] Barry Boehm. *COCOMO II Model Definition Manual*. USC, 2000.

[2] Muhammad Tanveer, Imran Hussain, Naveed Zahid, et al. A survey on machine learning techniques for software effort estimation: Trends, challenges, and opportunities. *Journal of Systems and Software*, 200:111618, 2023.

[3] Mohammad Azzeh and Ali Bou Nassif. Cross-company effort estimation using ensemble learning and feature selection. *Empirical Software Engineering*, 24(6):3821–3848, 2019.

[4] Barbara Kitchenham, Lesley Pickard, Stephen MacDonell, and Martin Shepperd. Evaluating software engineering prediction systems. *Information and Software Technology*, 43(11):733–743, 2001.

[5] Tore Foss, Erik Stensrud, Barbara Kitchenham, and Ivar Myrtveit. A simulation study of the model evaluation criterion mmre. *IEEE Transactions on Software Engineering*, 29(11):985–995, 2003.

[6] Gustav Karner. Metrics for objectory. Technical Report LiTH-IDA-Ex-9344:21, University of Linköping, Sweden, 1993. Diploma thesis, Department of Computer and Information Science.

[7] Derek M. Jones. Software estimation datasets - curated public collection. `https://github.com/Derek-Jones/Software-estimation-datasets`, 2022. Accessed: 2026-02-06.

[8] Daniel Rodríguez and Javier Dolado. Dase: Data analysis in software engineering repository. `https://github.com/danrodgar/DASE`, 2023. Aggregated effort estimation datasets 1979–2022. CC0-1.0 license.

[9] Hoc Thai Huynh, Radek Silhavy, Zdenka Prokopova, and Petr Silhavy. Comparing stacking ensemble and deep learning for software project effort estimation. *IEEE Access*, 11:60590–60604, 2023. doi: 10.1109/ACCESS.2023.3286372.

[10] Hoc Thai Huynh and Radek Silhavy. Ucp dataset for software effort estimation. Zenodo, 2023. Supplementary data for IEEE Access paper on stacking ensemble methods.

[11] Bradley Efron and Robert J. Tibshirani. *An Introduction to the Bootstrap*. Chapman & Hall/CRC, Boca Raton, FL, 1994.

[12] Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83, 1945.

[13] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[14] Sture Holm. A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 6(2):65–70, 1979.

[15] Gary Macbeth, Esteban Razumiejczyk, and Rubén Ledesma. Cliff's delta calculator: A non-parametric effect size program for two groups of observations. *Universitas Psychologica*, 10 (2):545–555, 2011.

[16] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006.

[17] Salvador Garcia, Alberto Fernandez, Jesus Luengo, and Francisco Herrera. Advanced nonparametric tests for multiple comparisons in computational intelligence and data mining. *Information Sciences*, 180(10):2044–2064, 2010.

[18] Fabian Pedregosa et al. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[19] Luis Cruz and Rui Abreu. Open science in software engineering research: The case for open data and replication. *Empirical Software Engineering*, 24(6):3829–3849, 2019.

[20] Belen Lopez, Juan C Rodriguez, and Salvador Garcia. Empirical software engineering reproducibility: A systematic review. *Information and Software Technology*, 136:106579, 2021.

[21] Jerome H Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, pages 1189–1232, 2001.

[22] P. Pandey, T. Sharma, and S. Saha. Hybrid ensemble learning for software effort estimation using meta-heuristic optimization. *Applied Soft Computing*, 135:110054, 2023.

[23] A. Alqadi and A. Abran. Deep learning models for software effort estimation: An empirical study. *IEEE Access*, 9:135012–135026, 2021.

[24] Vineeth Nair and Tim Menzies. Open problems in reproducibility, replication, and transparency in software engineering. In *Proceedings of the 42nd International Conference on Software Engineering: New Ideas and Emerging Results*, pages 1–4. IEEE, 2020.

[25] Muhammad Tanveer, Imran Hussain, Naveed Zahid, et al. A comprehensive analysis of ensemble learning models for software effort estimation. *IEEE Access*, 11:76590–76608, 2023.

[26] Yong Yu, Xin Xia, David Lo, and Ahmed E Hassan. Transfer learning in software engineering: A systematic mapping study. *Empirical Software Engineering*, 26(3):1–46, 2021.