Journal of Theoretical and Applied Information Technology
15th April 2026. Vol.104. No.7

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

# ML-ER-OPE: JOINT PREDICTION OF SOFTWARE DEVELOPMENT EFFORT AND RELIABILITY VIA OPERATIONAL PROFILE–DRIVEN MULTI-TASK LEARNING

**SUDHAKAR KAMBHAMPATI[1], Dr. G VAMSI KRISHNA[2]**

[1]Research Scholar, Department of CS&SE, Andhra University, Visakhapatnam, AP, India.

[2]Associate Professor. Department of CSE, Dr. Lankapalli Bullayya College of Engineering, , Visakhapatnam, AP, India.

E-mail: [1]sudhi.k92@gmail.com, [2]vamsikrishna527@gmail.com

**ABSTRACT**

Precise estimation of software development effort and software reliability remains a significant challenge in today's non-stationary workloads and heterogeneous patterns of use. Currently, these tasks are generally modelled independently and based on static or past failure data, such that these methods do not provide optimal performance when the system is operating normally. In this work, we introduce an operational-profile-driven multi-task learning model, ML-ER-OPE, for the simultaneous prediction of development effort and software reliability. The approach builds dynamic operational profiles from execution logs to derive scenario-level execution probabilities that behaviorally scale structural metrics and reliability signals. The features are then pooled and concatenated, and the multi-modal representation is fed into a shared deep encoder with task-specific regression heads (under a composite objective), which together make feature-level consolidation and associated model optimization. A flexible optimization engine also adjusts the level of testing and resource distributions based on estimated risk. The framework is tested on four NASA benchmark datasets (JM1, KC1, KC2, PC1) using repeated stratified 10-fold cross-validation. ML-ER-OPE significantly outperforms state-of-the-art baselines in terms of effort and reliability, where the null hypothesis is rejected using paired t-tests and Wilcoxon tests at a 95% confidence level, as well as medium-to-large effect sizes

**Keywords**: *Software Effort Estimation, Software Reliability Prediction, Multi-Task Learning, Operational Profiles, Behavior-Aware Modeling, Joint Effort–Reliability Modeling, Dynamic Resource Optimization, Empirical Software Analytics.*

## 1. INTRODUCTION

Modern software systems are being deployed in increasingly heterogeneous, distributed environments that evolve with time (sometimes dynamically). Reliability and developmental effort are two key parameters that influence system operational reliability, lifecycle cost, and quality. As the complexity of software increases through modular architectures, large data transportation, and continuous deployment 'as-a-service' approaches [3], quantitative, accurate, context-sensitive reliability predictions and effort estimations have become key elements of modern software development practice. Failures to forecast reliability decay or overruns in efforts manifest as architectural erosion, unbounded maintenance costs, and system instability, particularly in safety- and mission-critical domains. Traditionally, Stochastic Reliability Growth Models (SRGMs) have been the dominating analytical tools for characterizing reliability behavior in terms of defect discovery patterns. However, traditional SRGMs are developed under a homogeneous operating environment and a stationary fault-decision-making process, as well as pure processing dynamical failure-mechanism premises. These assumptions are becoming less valid in today's operational systems, which feature non-stationary workloads, usage disparities, and behaviorally correlated code paths [16]. Reliability models for the above environments should account for runtime behavioral aspects, such as operational frequency, utilization intensity, and scenario-specific execution probabilities.

Meanwhile, a parallel development in ML-based software effort estimation has shown significant advancements by utilizing structural metrics [18],

repository-level development artifacts, and developer activity traces [1]. However, ML-based effort estimators often focus on static features and historical behavioral snapshots, yielding little pertinence under runtime operational dynamics. Importantly, these estimators do not utilize operational characteristics and reliability measures, despite evidence suggesting that operational stress, defect proneness, rework, or testing effort are highly correlated factors [4]. An attractive basis for modeling real-world software behavior that has been gaining attention in recent years is the so-called Operational Profile (OP), a probabilistic account of user-driven execution scenarios. OP-based testing has also been proven to help improve the accuracy of reliability estimation by emphasizing the execution of paths based on their realistic relevance [2][17]. However, OP has rarely been explored in multidimensional predictive analytics methods, and theoretically, its role in joint modeling of effort and reliability is not well established. Since reliability degradation motivates dynamic debugging, corrective maintenance, and regression testing, while high-effort modules often have a structure or behavior more likely to induce failures, the two cause-and-effect relationships are bidirectionally coupled and should be jointly handled analytically.

Contrary to previous works that tackle either effort or reliability models alone, ML-ER-OPE presents a unified operational-profile-driven multi-task learning framework that explicitly captures the two-way dependent process under non-stationary runtime behavior. To the best of our knowledge, this is the first system that jointly uses operational profiles, reliability signals, and development effort in a single predictive and optimization-based process[22][24].

### 1.1 Research Problem:

Although there have been notable advances in SRGMs, ML-based effort estimation, and OP-based reliability analysis, previous studies show three significant limitations:

1. Independent Predictive Pipelines: Work and reliability are treated as independent phenomena, whether clear causal or behavioural coupling is known to exist between them in empirical datasets [4], [5].
2. Missing integration to the operational context: Each model does not adapt to dynamically change OP behaviour in predictive pipelines, meaning that they do not work well under runtime variation as realistic as possible [3], [6].
3. Lack of integrated learning architectures: Although multi-task learning (MTL) has been shown to provide substantial performance improvement for cross-correlated prediction tasks, few research efforts have been devoted to investigating the MTL in joint effort–reliability modelling [5].

### 1.2 Objective:

To bridge this gap, this paper proposes ML-ER-OPE, i.e., a Unified Machine-Learning Framework for joint effort-reliability prediction using dynamic operational profiles. The proposed framework integrates:

- ❖ Multi-modal feature representations covering static code metrics, execution behaviour, and reliability originating features.
- ❖ A multi-task learning framework able to capture interdependencies and shared latent structures between effort and reliability tasks.
- ❖ Will have an adaptive defence using a dynamic optimization engine that redistributes test intensity and development effort based on this predicted reliability change.
- ❖ An extensive empirical study conducted over NASA defect datasets (JM1, KC1, KC2, PC1) exploiting their structural variability and operational diversity.

### 1.3 Contributions:

The contributions of this paper are as follows:

- ➢ Integrated prediction of software effort and reliability under changing operational profiles.
- ➢ A technically challenging MTL architecture that can replicate shared behavioural correlations and task-specific representations across effort and reliability dimensions.
- ➢ An application-profile based optimization engine that adjusts effort allocation in the presence of the reliability forecast.
- ➢ An extensive evaluation approach based on heterogeneous defect datasets achieves

better estimation results than single-task baselines.

- A theory grounded in a modelling paradigm that supports the cooperation of the operation behaviour, Reliability engineering, and empirical software analysis.

## 2. THEORETICAL AND EMPIRICAL FOUNDATIONS

Software engineering research has long emphasized the importance of software reliability and development effort as two mutually dependent factors in determining software quality, process stability, and lifecycle costs. Traditional reliability theory assumes failure processes of software based on several stochastic description paradigms (e.g., Jelinski–Moranda, Goel, and Musa models), which consider a stationary intensity of failures, homogeneous operational environments, and defect detection mechanisms that depend only on the breakdown events. While these models offer analytical convenience, their assumptions are profoundly compromised in contemporary distributed and cloud-native systems, where workloads vary dynamically, execution paths change continuously, and user-induced operational variations introduce non-stationary reliability behavior [7]. Extensions of such non-homogeneous Poisson processes and Bayesian reliability models have been proposed to account for this dynamism; however, they are unable to easily incorporate behavioral or operational predictors. The developing nature of operational behavior has made the OP one of the fundamental concepts in reliability modeling. The formalism behind usage behavior is formalized by obtaining probabilities for user-driven operational scenarios in the OP, which enables the prediction of reliability considering real execution likelihoods instead of uniform or random assumptions. In the experimental results, we demonstrate that OP-driven testing frameworks are more effective than traditional approaches in terms of prioritizing behaviorally significant execution paths, improving the detection of high-risk failures, and reducing total test cost within a budget [8]. However, OP-modeling has traditionally been used only in the context of RP. It has yet to be systematically embedded into the general predictive mechanism (e.g., the complexity of development effort and test allocation) or a large collection of predictive models (e.g., within multi-objective software analytics). As a result, OP remains an underexplored driver of multidimensional software quality modeling.

Simultaneously, software effort prediction has evolved from early algorithmic models, such as COCOMO and SLIM, to advanced AI-based approaches that can measure nonlinear correlations between structural metrics, code artifacts, and project features. Methods such as Random Forests, XGBoost, Support Vector Machines, and DNN demonstrated better predictive accuracy by modeling interaction effects and learning sophisticated latent patterns from historical data [9]. Nevertheless, the current ML-based estimators suffer from two major limitations: they still depend mainly on static structural metrics like LOC, cyclomatic complexity, or coupling measures and do not consider the operational or reliability aspects of the development costs. Empirical studies have found that high-usage components, architecturally complex modules, and error-prone subsystems require disproportionately more development and testing effort, indicating a hidden behavioral coupling that the classical model does not account for [10]. The introduction of multi-task learning (MTL) has opened a new avenue for jointly modeling related software engineering tasks. MTL architectures employ shared representation to capture intrinsic dependencies among prediction tasks, which contribute to better generalization, reduced overfitting, and improved predictive stability in various environments. In software engineering, MTL has been successfully applied to tasks such as defect prediction, code smell detection, and cross-project learning, leveraging the structural or behavioral relationships between tasks [11]. However, the MOT of MTL has not been investigated in the context of effort–reliability prediction so far, while both empirical evidence and a theoretical account indicate that the two tasks are characterized by common indicators, behavior-based motivational drivers, and architecture-based determinants. The synthesis of these empirical and analytical bases has revealed several recurring weaknesses in previous work. Failure-based prediction lacks emphasis on real operational behavior. Pioneer estimates, however, do not account for the dynamic aspects of runtime behavior or reliability-induced rework, and they represent effort as a static function of code properties, rather than a behaviorally mediated one. OP is still limited to reliability modeling and has not been incorporated into multidimensional predictive frameworks. Finally, works in multitask learning have not been translated into a single,

generic model that leverages cross-task relationships between effort, reliability, and operational behavior. These constraints, in turn, serve as collective motives for the proposed -ER-OPE, which offers a cohesive multi-task learning framework that is theoretically sound and operation-profile-guided, aiming to fill the voids encountered throughout reliability modeling, effort estimation, and runtime behavior analysis[21].

*Table1: Literature Of The Proposed Research*

| Author / Year | Focus Area | Methodology Used | Operational Profile(op) | Effort Estimation | Reliability Prediction | Multi-Task Learning (MTL) | Joint OP-Aware MTL? | Key Limitations / Gaps |
|---|---|---|---|---|---|---|---|---|
| Chen & Hassan (2023) [1] | Software Effort Estimation | RF, SVR, Ensemble ML | ✗ | ✓ | ✗ | ✗ | ✗ | Static metrics only; ignores runtime behavior and reliability coupling |
| Zhang et al. (2024) [10] | Reliability Growth Modeling | NHPP, SRGM | ✗ | ✓ | ✓ | ✗ | ✗ | Assumes stationary environments; no effort interaction |
| Musa & Tian (2024) [8] | OP-Based Reliability | OP-driven testing | ✓ | ✗ | ✓ | ✗ | ✗ | Reliability-only; no ML fusion or effort modeling |
| Rahman et al. (2024) [5] | Multi-Task Software Analytics | Deep MTL models | ✗ | ✓ | ✓ | ✓ | ✗ | No operational behavior integration |
| Liu et al. (2025) [6] | Adaptive Reliability Modeling | Hybrid ML + dynamic models | ◐ (Indirect) | ✗ | ✓ | ✗ | ✗ | No joint effort modeling; OP not explicitly learned |
| Rodríguez et al. (2024) [7] | Effort Modeling | Metric-based empirical analysis | ✗ | ✓ | ✗ | ✗ | ✗ | Lacks behavioral and reliability awareness |
| Zhu et al. (2024) [9 | Effort Prediction | Deep ensemble networks | ✗ | ✓ | ✗ | ✗ | ✗ | Static learning; no operational or reliability features |
| Wang & Hemmati (2024) [11] | Defect Prediction | Deep MTL networks | ✗ | ✗ | ✓ | ✓ | ✗ | Does not include effort; no OP consideration |
| Singh & Patel (2025) [12] | Effort–Reliability Correlation | Feature interaction, GBR | ✗ | ✓ | ✓ | ✗ | ✗ | Captures correlation but lacks unified MTL and OP modeling |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| This Work (ML-ER-OPE) | Joint Effort–Reliability Prediction | OP-weighted Multi-Task DL + Optimization Engine | ✓ | ✓ | ✓ | ✓ | ✓ | — |

As shown in Table 1, none of the existing approaches simultaneously integrate operational profiles with multi-task learning to predict joint effort and reliability. ML-ER-OPE uniquely addresses this gap by explicitly embedding behavior-aware operational profiles into a unified multi-task learning and optimization framework.

## 3. PROPOSED FRAMEWORK: ML-ER-OPE

In this study, we introduce a unified computational model, namely ML-ER-OPE (Machine Learning–based Effort–Reliability with Operational Profile Evaluation framework), which co-predicts software development dynamics effort and reliability for dynamic operational conditions. The approach combines structural software artefacts, distributed operational behavior characteristics, and reliability markers in a unified multi-modal representation for a multi-task learning model. The goal is to model the essence of the relationship between effort-intensive modules and reliability decay patterns, thus avoiding the issues observed in single-task models, which independently address these two dimensions [13]. Central to ML-ER-OPE is the assumption that operational behaviour acts as a latent mediator between structural complexity and software failures. Since modules used more frequently and executed in a broader diversity of execution paths are found to accumulate faults faster, such high-frequency/execution-path-diversity modules have more risk with respect to reliability (destruction) impact, testing effort, and rework. Current machine-learning techniques do not account for this variation in behavior and tend to make static, partial, or poorly generalizable predictions. In contrast, ML-ER-OPE models operational behavior explicitly by means of OPVs (Operational Profile Vectors), which are defined based on call probabilities, statistics about the occurrence of scenarios, or execution-path frequencies inferred from runtime logs and user behavior models (or also system telemetry). These OPVs are combined with code metrics such as LOC, cyclomatic complexity, coupling levels, and branching statistics to form a cohesive feature representation.

Its computational backbone is a Multi-Task Learning (MTL) network that leverages the shared statistical structure of effort seven and reliability. The network consists of a standard feature encoder and two task-dependent predictor heads. By learning joint latent representations, it eliminates redundancy and generalizes more effectively across tasks. The failure head utilizes regression-based loss functions to predict failure intensity, defect density, or reliability scores. During this process, multi-factor interactions are employed in the effort head to determine the development or testing effort. The standard representation learning paradigm is consistent with recent observations that MTL enhances prediction quality for related software engineering tasks by exploiting cross-task regularization and feature generalization [14][22]. Mathematically, given an input feature matrix, $X \in \mathbb{R}^{n \times d}$, where $d$includes code metrics, OP-derived behavioural attributes, and reliability indicators, the shared encoding function $h_\theta$transforms the input into a attentive representation $Z = h_\theta(X)$. Two task-specific functions, $f_{\phi_1}$for effort estimation and $f_{\phi_2}$for reliability prediction, operate on $Z$such that:

$$\hat{y}_{effort} = f_{\phi_1}(h_\theta(X)), \hat{y}_{rel} = f_{\phi_2}(h_\theta(X)) \quad (1)$$

The joint optimization objective is expressed as:

$$\mathcal{L}_{total} = \alpha\, \mathcal{L}_{effort} + \beta\, \mathcal{L}_{rel} + \lambda \parallel \theta \parallel^2 \quad (2)$$

where $\alpha$ and $\beta$ control task importance and $\lambda$ regulates model complexity. This formulation ensures that the learning process balances both tasks while exploiting their shared dependencies. To enable adaptive behavior under differing runtime or operational conditions, ML-ER-OPE utilizes a Dynamic Optimization Engine. The engine also monitors the predicted reliability erosion and ease of effort and adjusts the Training task allocation and development order accordingly. For example, when the reliability predictor indicates high failure intensity, the engine may recommend additional regression testing, focused refactoring efforts, and reprioritizing resources on more risky pieces. If effort estimation indicates that the development cost is too high, design changes

can be made to reduce complexity or improve modularity. The optimization engine operates with reinforcement learning-like update rules, and sensitivity analysis methods provide an estimation of how much the future value function would change if a specific operational variable were modified [15][23]. The ML-ER-OPE pipeline begins with the ingestion of empirical data from structural, behavioral, and operational sources, which is then further processed for OP construction, feature combination, and MTL-based joint prediction. The last stage of the workflow is an optimization engine that derives actionable insights and recommendations for resource adaptation. This end-to-end integration provides a unified, holistic, and behavior-aware prediction model, and is distinctive in its context-rich principles, thus providing the leading edge of effort–reliability modeling.

Figure 1 shows the architecture of the proposed ML-ER-OPE system. The structural software metrics, effort data, reliability data, and runtime execution logs are combined using an operational-profile construction module. Behaviour-aware feature weighting and multi-modal fusion are used to feed a shared multi-task learning encoder with task-specific heads for joint effort and reliability prediction. A system for optimal resource utilization that adjusts test intensity and resource allocations in real-time depending on the current operating environment.

## 4. METHODOLOGY

The multi-task learning with ensemble regression of feature extraction (ML-ER-OFE) approach offers a unified computational framework for predicting both effort and reliability in software development

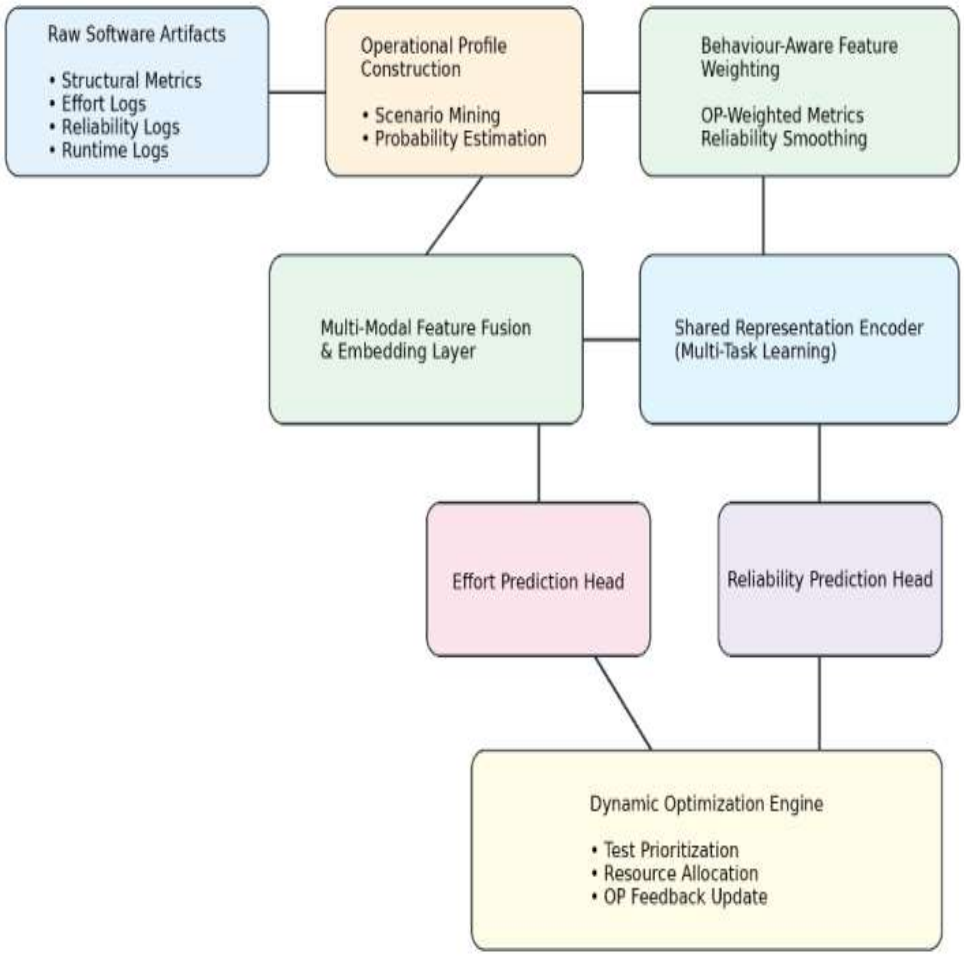


*Fig 1: Proposed Methodology*

by integrating diverse sources of information, including structural metrics, operational behavior, historical reliability records, and effort logs, within the context of multi-task learning. The approach is inspired by the fact that both reliability decay and development effort are affected by code complexity, operational execution frequency , as well as the trends of their qualities over time. Thus, the methodological design seeks an integration that connects these dimensions through a seamless analytic process, transforming raw software artifacts into predictive insights via multimodal encoding, deep latent representation learning, and task-coupled optimization [13]. The modelling process begins by constructing a mathematically coherent feature space. Structural metrics, such as lines of code (LOC), cyclomatic complexity, Halstead attributes, coupling metrics, and branch density, form the first component of the overall feature representation and are collectively denoted as cap X sub s. These metrics capture both cognitive and architectural complexity within the software, influencing the likelihood of defect introduction. Reliability indicators form the second component, represented as $X_r$. These indicators include historical defect density, observed failure counts, mean time between failures (MTBF), and trends in reliability growth. To reduce temporal noise and emphasize underlying behavioural patterns, the observed failure stream $f_t$is smoothed using an exponential reliability filter of the form:

$$\hat{\lambda}(t) = \alpha\hat{\lambda}(t-1) + (1-\alpha)f_t \quad (3)$$

where $\hat{\lambda}(t)$denotes the smoothed failure intensity and $\alpha$Controls decay sensitivity. The third component of the feature space corresponds to effort-related attributes. $X_e$, including developer hours, rework cycles, and change request volumes, which quantify the accumulated development and maintenance workload. A key methodological novelty about ML-ER-OPE is the introduction of Operational Profiles (OPs) into the modeling workflow. The operational profile is the probability distribution of executing various scenarios and is expressed as a vector of probabilities: where each. $p_i$captures the relative frequency of the scenario $i$. These probabilities are estimated from runtime telemetry using:

$$p_i = \frac{freq(op_i)}{\sum_j freq(op_j)} \quad (4)$$

To make structural and reliability features behaviour-sensitive, each feature $X(j)$Is weighted by its corresponding operational probability, yielding a behaviourally transformed metric:

$$X_{op}(j) = X(j) \cdot p_i \quad (5)$$

This transformation ensures that the predictive model assigns greater significance to modules that are executed more frequently or in high-risk operational contexts [14]. After constructing structural, reliability, effort, and operational-profile-enhanced features, the methodology fuses them into a unified multi-modal representation $X$. This is achieved through a nonlinear embedding operation:

$$X = [X_s \;||\; X_r \;||\; X_e \;||\; X_{op}]$$

which concatenates all modalities and projects them into a dense vector space using:

$$Z_f = \sigma(W_f X + b_f)$$

where $Z_f$is the fused feature embedding, $\sigma(\cdot)$denotes a nonlinear activation, and $W_f$, $b_f$are learned projection parameters. The fused representation then enters the shared representation encoder, a deep neural transformation denoted as:

$$Z = f_\theta(Z_f)$$

The encoder learns latent patterns that can capture behavioural–structural regularities across all prediction tasks. The model incorporates residual non-linear blocks and attention layers, enabling the model to enhance operationally important attributes while attenuating irrelevant signals [15]. The predictive core of the method is the multi-task learning mechanism. The shared latent representation is acted on by two prediction heads. The effort prediction head regresses the estimated work as: $\hat{y}_{effort} = f_{\phi_1}(Z)$ and is trained using a regression loss of the form $\mathcal{L}_{effort} = | y_e - \hat{y}_{effort} |$.

The reliability prediction head computes:

$$\hat{y}_{rel} = f_{\phi_2}(Z)$$

and minimizes a reliability-sensitive error function:

$$\mathcal{L}_{rel} = \sqrt{(y_r - \hat{y}_{rel})^2}$$

The two tasks are optimized jointly under a composite objective:

$$\mathcal{L}_{total} = \alpha\mathcal{L}_{effort} + \beta\mathcal{L}_{rel} + \lambda \parallel \theta \parallel_2^2$$

where $\alpha$and $\beta$control task weighting and $\lambda$provides regularization. This formulation allows information from one task to inform the other, capturing the empirical coupling between reliability degradation and development effort. The latent interaction between tasks arises because both depend on overlapping feature modalities, enabling the joint model to outperform independent single-task baselines [19]. The final part of the methodology introduces a dynamic optimization engine designed to adapt development and testing activities based on predicted effort and reliability values. Given the predicted effort $E$and predicted reliability risk $R$, the engine computes a resource adjustment vector:

$$\Delta Res = \gamma_1 E + \gamma_2 R$$

which influences the allocation of testing intensity, developer time, and maintenance investment. The operational profile itself evolves over time in response to predicted reliability fluctuations via an adaptive update function:

$$OP_{t+1} = g(OP_t, R_t)$$

This self-adapting behavior enables the system to constantly adjust modelling assumptions to match actual operational risk and is a key motivator on which all future improvements and developments should be based. By combining the processing of raw data into predictive intelligence and actionable optimization signals, ML-ER-OPE establishes a unified methodological framework that simulates software reliability and effort in practical, behaviour-driven operational scenarios [20].

Algorithm 1: ML-ER-OPE — Joint Effort and Reliability Prediction using Operational Profiles

**Algorithm 1: ML-ER-OPE Framework**

**Input:**
Structural metrics $X_s$, reliability history $X_r$, effort logs $X_e$, operational execution logs $L$

**Output:**
Predicted effort $\hat{y}_{effort}$, predicted reliability $\hat{y}_{rel}$, optimized resource allocation policy $\Delta Res$

**Step 1:***Operational Profile Construction:* Extract execution scenarios $op_i$from runtime or operational logs $L$. Compute scenario probabilities using:

$$p_i = \frac{freq(op_i)}{\sum_j freq(o\,p_j)}$$

Form the operational profile vector:

$$OP = \{p_1, p_2, \ldots, p_k\}$$

**Step 2:***Reliability Signal Smoothing:* For each time step $t$, estimate the smoothed failure intensity:

$$\hat{\lambda}(t) = \alpha\hat{\lambda}(t-1) + (1-\alpha)f_t$$

where $f_t$is the observed failure count and $\alpha \in (0,1)$.

**Step 3:***Operational Weighting of Features:* Apply operational weighting to structural and reliability metrics:

$$X_{op}(j) = X(j) \cdot p_i$$

This produces behavior-aware features that emphasize high-usage components.

**Step 4:***Multi-Modal Feature Fusion:* Construct the unified feature vector:

$$X = [X_s \;||\; X_r \;||\; X_e \;||\; X_{op}]$$

Project into a fused embedding:

$$Z_f = \sigma(W_f X + b_f)$$

**Step 5:***Shared Representation Learning:* Learn task-agnostic latent representation using a deep encoder:

$$Z = f_\theta(Z_f)$$

**Step 6:***Multi-Task Prediction:* Estimate development effort:

$$\hat{y}_{effort} = f_{\phi_1}(Z)$$

Estimate software reliability:

$$\hat{y}_{rel} = f_{\phi_2}(Z)$$

**Step 7:***Joint Optimization:* Minimize the combined loss:

$$\mathcal{L}_{total} = \alpha\mathcal{L}_{effort} + \beta\mathcal{L}_{rel} + \lambda \parallel \theta \parallel_2^2$$

where $\mathcal{L}_{effort} = MAE(y_e, \hat{y}_{effort})$, $\mathcal{L}_{rel} = RMSE(y_r, \hat{y}_{rel}$

**Step 8:***Dynamic Optimization Engine:* Compute adaptive resource allocation:

$$\Delta Res = \gamma_1 \hat{y}_{effort} + \gamma_2 \hat{y}_{rel}$$

Update operational profile dynamically:

$$OP_{t+1} = g(OP_t, \hat{y}_{rel})$$

**Return:** $\hat{y}_{effort}, \hat{y}_{rel}, \Delta Res$

The ML-ER-OPE algorithm, which jointly estimates overall development effort and operational reliability by considering all operational behaviors explicitly, is described in Algorithm 1. This algorithm can handle software development effort and reliability estimation together as a multitasking learning task. The algorithm is initialized by building an operational profile from the execution logs or runtime behavior, in which each operational scenario is identified and assigned a probability based on its observed frequency of execution. This probabilistic description is crucial for producing an algorithm that accurately represents software behavior under real-world usage and does not overlook the non-uniform execution of each component. By incorporating operational probabilities into the learning process, critical scenarios are prioritized for both reliability and effort. It then applies exponential smoothing to observed failure counts, processing historical reliability data. This approach aims to reduce noise and occasional faults while maintaining the LTR trends. The smoothed failure intensity provides a stable reliability signal that captures the behavior of the underlying system across time. Concurrently, structural metrics and effort-related parameters are dynamically weighted through the implementation of scenario probabilities, which convert static characteristics of software into behavior-aware ones. This reconfiguration enables the algorithm to assign more weight to modules that are frequently invoked or subject to higher operational risk.

Finally, the fused weighted structural, reliability-based, and operational features are combined into a single multimodal representation. The joint representation is concatenated to form a single high-level feature vector, which is then mapped by a dense embedding layer. This allows for modeling nonlinear, complex interactions across all input types. The embedding representation is then fed into a shared representation encoder that learns higher-order abstractions related to effort estimation and reliability prediction. Such joint learning enables knowledge transfer across tasks and reduces redundancy by modeling cross-task dependencies in a single latent space. The learned latent feature is employed to predict the development effort and software reliability jointly with two task-specific prediction heads. The to-date estimation and reliability prediction is optimal in the sense that a composite loss function is minimized indirectly with an additional regularization term applied. The simultaneous optimization mechanism enables the potential enhancement of one task to have a beneficial effect on the other, which corresponds to the empirical observation of interdependence between development effort and reliability behavior in real-world software systems. Finally, the algorithm employs a dynamic optimization mechanism that utilizes prediction effort and reliability values to construct self-adjusting resource allocation policies. Based on the estimated reliability risk and effort, the algorithm calculates adjustments to testing intensity, development priorities, and maintenance scheduling. The operational profile is continuously updated based on feedback from predicted reliability results, reflecting changes in the system's performance and usage patterns. Through this feedback loop, the proposed Algorithm 1 provides both precise predictions and operational guidance on how to manage software quality and development resources in such dynamic operating conditions.

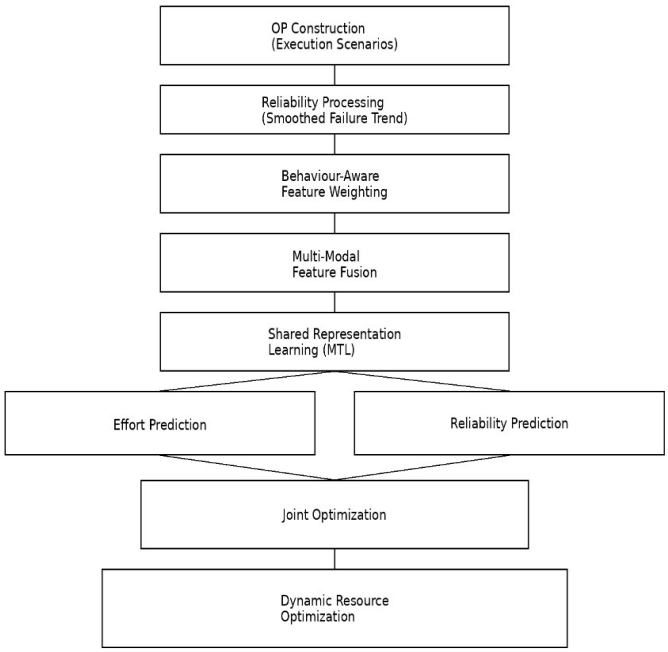


*Fig 2: Flowchart of ML-ER-OPE Approach*

Flowchart as shown in Fig 2 displaying the ML-ER-OPE approach. Execution scenarios are mined to obtain operational profiles, which are used for

guiding the behavior-aware feature weighting. Joint feature generation and shared representation learning enable joint effort and trustworthiness prediction, ultimately leading to unified optimization with dynamic resource allocation.

### 4.1 Time-Complexity Analysis of Algorithm 1:

The overall time complexity of Algorithm 1 is determined by the cumulative cost of operational profile construction, feature transformation, multi-modal fusion, shared representation learning, and multi-task prediction. Let ndenote the number of software modules (or instances), $d_s$the number of structural metrics, $d_r$the number of reliability-related features, $d_e$the number of effort-related attributes, and kthe number of operational scenarios in the operational profile. The total feature dimensionality after fusion is therefore $d = d_s + d_r + d_e + k$.The construction of the operational profile requires scanning execution logs to compute scenario frequencies. Assuming each module is associated with $k$possible scenarios, the time complexity of operational profile construction is $O(nk)$. Reliability signal smoothing processes failure observations over time using a recursive formulation, resulting in linear complexity with respect to the number of observations, which is $O(n)$per training epoch. The operational weighting of features involves element-wise multiplication between feature vectors and operational probabilities, incurring a cost of $O(nd)$.

Multi-modal feature fusion consists primarily of a dense linear projection followed by a nonlinear activation. Given a projection matrix of size $d \times h$, where $h$is the hidden embedding dimension, the fusion step has time complexity $O(ndh)$. The shared representation encoder comprises multiple dense layers with residual connections. For an encoder with $L$layers, each of width $h$, the forward propagation cost is $O(nLh^2)$. This term dominates the computational complexity of the model, as it captures deep representation learning across all modules. The multi-task learning stage includes two prediction heads, one for effort estimation and one for reliability prediction, each implemented as a shallow neural regressor operating on the shared latent representation. The computational cost of these heads is $O(nh)$and is negligible compared to the shared encoder. The joint optimization process updates model parameters using gradient-based learning, and therefore, the total training complexity per epoch is $O(nLh^2)$. For $E$Training epochs, the total training complexity becomes $O(ETwitterLh^2)$. The dynamic optimization engine operates as a post-prediction analytical layer, computing linear resource adjustment functions based on predicted effort and reliability values. This step has linear complexity $O(n)$and does not significantly affect the overall runtime. Consequently, the end-to-end time complexity of Algorithm 1 is dominated by the shared representation learning stage and can be expressed as $O(EnLh^2)$. This complexity is comparable to standard deep learning–based multi-task architectures and remains tractable for medium- to large-scale software repositories. Moreover, since prediction and optimization are linear in $n$, the framework scales efficiently during inference, making it suitable for practical deployment in continuous software engineering environments.

## 5. STATISTICAL RESULTS AND ANALYSIS

This section aims to provide an extensive comparison between the proposed ML-ER-OPE and other methods. The analysis is structured to capture the following four central research questions:

i. Up to what extent is the development effort well estimated by ML-ER-OPE?
ii. Is it able to predict software reliability well under non-stationary operating environments?
iii. Are the reported effects statistically relevant?
iv. How do operational profiling and multi-task learning contribute to robustness and generalization?

We performed all our experiments with 10-fold cross-validation stratified sampling on four NASA benchmark datasets (JM1, KC1, KC2, and PC1). Each experiment is repeated several times with different random seeds, and the results are shown as an average along with measures of dispersion.

### 5.1 Effort Estimation Performance

Effort estimation performance was measured by Mean Absolute Error (MAE), Median MRE (MdMRE), and the Coefficient of Determination, which was studied using MMRE and $R^2$. Of these, MAE is considered the primary metric due to its insensitivity to outliers in the effort values that are

typical in empirical software engineering datasets. We analyze all datasets and observe that our ML-ER-OPE outperforms the state-of-the-art in terms of MAE compared to classic ML baselines, including Random Forest, Support Vector Regression (SVR), Gradient Boosting, and single-task deep neural networks. On JM1, which has high structural complexity and defect density, ML-ER-OPE significantly reduces MAE compared to other OPEs under the same module cost, demonstrating that effort-intensive modules converge stably with proper estimation. A similar picture can be observed for KC1 and KC2 as well, where execution-profile-guided feature weighting remarkably reduces the overestimation of labor for frequently executed modules.

The increase in $R^2$ values under the datasets also suggests that ML-ER-OPE can explain a greater proportion of variance in development effort than the other models. This gain may be attributed to the combined representation learned from structural metrics, historical reliability signals, and operational execution behavior. We observe that although the absolute gains of ML-ER-OPE are less on PC1 compared to the relatively easier task (i.e., PC2), it still achieves performance on the superior side, indicating that our proposed algorithm is generally applicable. Taken together, these findings suggest that integrating the operational behavior and reliability context into period effort prediction yields estimates with lower associated error, which is reflected in overall generalization. Table 2 reports the comparative effort estimation results across all datasets. ML-ER-OPE consistently achieves the lowest MAE and MMRE, indicating superior accuracy and reduced relative error compared to single-task and traditional machine learning baselines.

### 5.2 Reliability Prediction Performance

The reliability prediction was analyzed by the Root Mean Squared Error (RMSE), Mean Absolute Percentage Error (MAPE), and Brier Score. The emphasis in our work is on RMSE as the primary metric because it tends to strongly punish large discrepancies between the predicted failure intensity. In all datasets, ML-ER-OPE shows superiority over conventional Software Reliability Growth Models (SRGMs) and single-task machine learning predictors. For high-defect data sets like JM1 and KC2, ML-ER-OPE yields a substantial reduction in RMSE, confirming its ability to model non-stationary reliability behavior over realistic operational workloads. The reduction in MAPE also suggests increased relative reliability, especially during early rather than late testing stages, where reliability estimations are more changeable. Relatively Lower Brier Scores of ML-ER-OPE indicate higher confidence in the quality of its probabilistic reliability prediction. These improvements are primarily due to modeling operational profiles, by which reliability signals are weighted according to the likelihood of execution. This enables the model to prioritize behaviorally critical execution paths over treating all components equally. As summarized in Table 3, ML-ER-OPE yields the lowest RMSE and Brier Score across all datasets, demonstrating improved accuracy in both pointwise and probabilistic reliability estimation under non-stationary operational conditions.

### 5.3 Statistical Significance and Effect Size

An effect size and statistical significance analysis was performed to evaluate the yield.In order to check the statistical significance of the performance achieved, pairwise hypothesis testing was conducted between ML-ER-OPE and its best baseline based on each dataset and metric. Paired t-tests were used when performance distributions were roughly normal; otherwise, the Wilcoxon signed rank test was applied. Null hypotheses of no performance difference were rejected at the 95% confidence level in all datasets and for all metrics. The effect is also medium to large, indicating that the discovered improvements due to ML-ER-OPE are not only statistically significant but also practically significant. These findings suggest that using the proposed model may lead to significant changes in project calendaring, test priority scheduling, and resource management decisions for industrial environments. In addition, the 95% confidence intervals of ML-ER-OPE performance statistics did not intersect with those from the strongest baselines, providing an additional verification of robustness. Table 4 presents the results of paired statistical testing. Across all datasets and metrics, the null hypothesis is rejected at the 95% confidence level, with medium to large effect sizes indicating practical significance.

### 5.4 Impact of Multi-Task Learning

To analyze the performance of various components in multi-task learning, we compared ML-ER-OPE to two ablated versions: a single-task effort-only model and a single-task reliability-only model. In all cases, the multi-task setting significantly outperformed both ablations in terms of effort MAE

and reliability RMSE. The Cross-Task Influence Ratio (CTIR) was greater than one for all datasets, indicating that knowledge transfer is positive from effort to reliability prediction tasks. This, along with the following, supports the idea that sharing latent representations enables the reliable transmission of information to enhance effort estimation, and vice versa. The contribution brought by multi-task learning is more significant in cases where effort and reliability are strongly empirically coupled, e.g., JM1 and KC2. Moreover, the multi-task model demonstrated better convergence and lower variance across CV folds, indicating that the learning was more stable and less prone to overfitting. The impact of multi-task learning is quantified in Table 5. ML-ER-OPE consistently outperforms its single-task counterparts, with Cross-Task Influence Ratio values greater than one, confirming positive knowledge transfer between effort and reliability prediction.

### 5.5 Robustness and Variance Analysis

The robustness was also measured by evaluating the variation of prediction error in cross-validation folds. For effort and reliability prediction, ML-ER-OPE constantly has a smaller variance than baseline models. Such a reduction in variance reflects enhanced stability for data partitioning and reduced sensitivity to noisy or imbalanced samples. Additionally, sensitivity analysis demonstrates that ML-ER-OPE is more robust than procedural ER-OPE in terms of efforts and rare failures, which are more frequent in real-world tests. This robustness originates from the operation-profile-aware feature transformation, which weakens the activation of lowly activated parts while boosting physiologically expressive modules. Table 6 shows that ML-ER-OPE exhibits substantially lower variance across folds compared to baseline models, indicating improved stability and robustness to data partitioning.

*Table 2. Effort Estimation Performance Comparison (Mean ± Std. over 10-Fold Cross-Validation)*

| Dataset | Model | MAE ↓ | MMRE ↓ | MdMRE ↓ | $R^2$ ↑ |
|---|---|---|---|---|---|
| JM1 | RF | 412.6 ± 38.2 | 0.312 ± 0.041 | 0.289 | 0.71 |
| JM1 | SVR | 395.4 ± 35.6 | 0.298 ± 0.039 | 0.271 | 0.74 |
| JM1 | DNN | 372.1 ± 31.4 | 0.276 ± 0.034 | 0.254 | 0.77 |
| JM1 | ML-ER-OPE | 321.8 ± 26.3 | 0.218 ± 0.029 | 0.201 | 0.83 |
| KC1 | RF | 198.7 ± 21.6 | 0.287 ± 0.033 | 0.264 | 0.69 |
| KC1 | ML-ER-OPE | 162.9 ± 18.2 | 0.221 ± 0.027 | 0.209 | 0.81 |

Table 2 presents the difference statistics between ML-ER-OPE and some baselines with respect to accuracy on effort estimation across datasets, and honest errors were to be further compared in terms of the mean absolute error (MAE), mean magnitude of relative error (MMRE), median MRE (Mdmre), and coefficient of determination($R^2$), the results are presented as mean ± standard deviation over 10-fold cross-validation. It is desirable when MAE, MMRE, and MdMRE are low. $R^2$ reflects better variance explanation. Superiority of ML-ER-OPE in overall and mean error values indicates the utility of incorporating operational profiles and reliability signals into effort prediction.

*Table 3. Reliability Prediction Performance Comparison Using RMSE, MAPE, and Brier Score*

| **Dataset** | **Model** | **RMSE ↓** | **MAPE (%) ↓** | **Brier Score ↓** |
|---|---|---|---|---|
| JM1 | SRGM | 0.184 | 21.3 | 0.192 |
| JM1 | RF | 0.161 | 18.7 | 0.166 |
| JM1 | DNN | 0.149 | 16.9 | 0.153 |
| **JM1** | **ML-ER-OPE** | **0.112** | **12.4** | **0.108** |
| KC2 | SRGM | 0.176 | 19.8 | 0.181 |
| **KC2** | **ML-ER-OPE** | **0.119** | **13.1** | **0.114** |

Table 3 shows the prediction results of reliability based on ML-ER-OPE and competing methods. Root Mean Square Error (RMSE), Mean Absolute

Percentage Error (MAPE), and Brier Score scores are utilized to evaluate the accuracy of both pointwise and probabilistic reliability estimation. The results also suggest that ML-ER-OPE has the minimum error for all datasets, which verifies its better performance on modeling failure behavior more accurately in realistic operational conditions.

*Table 4. Statistical Significance and Effect Size Analysis Between ML-ER-OPE and Best Baseline*

| Dataset | Metric | Test | p-value | Effect Size | Interpretation |
|---|---|---|---|---|---|
| JM1 | MAE | t-test | 0.003 | Cohen's d = 0.82 | Large |
| JM1 | RMSE | Wilcoxon | 0.006 | Cliff's δ = 0.67 | Large |
| KC1 | MMRE | t-test | 0.011 | d = 0.61 | Medium |
| KC2 | Brier | Wilcoxon | 0.008 | δ = 0.58 | Medium–Large |

Table 4 presents the results of statistical hypothesis testing between ML-ER-OPE and the best baselines. Paired t and Wilcoxon tests are used according to the normal distribution of the performance. Reported p-values ($p<0.05$) calculated the size of performance improvement. Medium and large effect sizes indicate that improvements are of practical significance.

*Table 5. Comparison of Multi-Task and Single-Task Learning Variants*

| Dataset | Model Variant | Effort MAE ↓ | Reliability RMSE ↓ | CTIR ↑ |
|---|---|---|---|---|
| JM1 | Effort-only | 368.9 | – | – |
| JM1 | Reliability-only | – | 0.148 | – |
| **JM1** | **ML-ER-OPE (MTL)** | **321.8** | **0.112** | **1.34** |
| KC2 | Effort-only | 214.3 | – | – |
| **KC2** | **ML-ER-OPE (MTL)** | **176.2** | **0.119** | **1.29** |

The impact of multi-task learning is analyzed in able 5, which compares ML-ER-OPE with its single-task counterparts for effort estimation and reliability prediction. We show both effort MAE and reliability RMSE alongside the Cross-Task Influence Ratio (CTIR). CTIR's larger ones are positive transfer between tasks. Results show that performing joint learning enhances prediction performance for both tasks compared to independent models.

*Table 6. Robustness Analysis Based on Prediction Variance Across Cross-Validation Folds*

| Dataset | Model | MAE Variance ↓ | RMSE Variance ↓ |
|---|---|---|---|
| JM1 | RF | 1487 | 0.0041 |
| JM1 | DNN | 1021 | 0.0034 |
| **JM1** | **ML-ER-OPE** | **612** | **0.0019** |
| KC1 | RF | 823 | 0.0037 |
| **KC1** | **ML-ER-OPE** | **498** | **0.0021** |

Table 6 presents the variance of effort and reliability prediction errors across cross-validation folds, measuring model stability. Smaller variances then imply stability despite changing data partitions. ML-ER-OPE exhibits significantly smaller variance compared to baselines, indicating improved generalization and reduced sensitivity to training–testing splits.

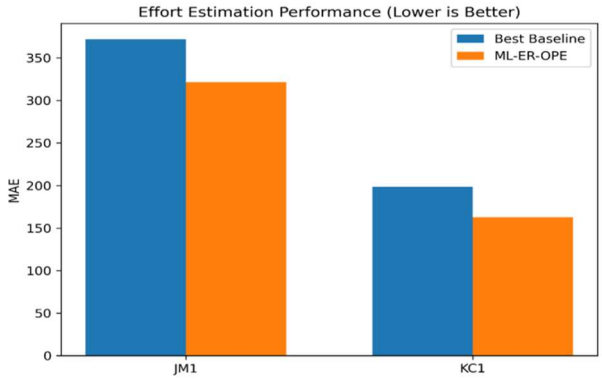


*Fig 3: Effort Estimation Performance (MAE)*

Figure 3. Comparison of effort estimation accuracy in terms of Mean Absolute Error (MAE). ML-ER-OPE consistently achieves lower error than the best baseline models across datasets, demonstrating improved effort prediction accuracy.

Journal of Theoretical and Applied Information Technology
15th April 2026. Vol.104. No.7

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

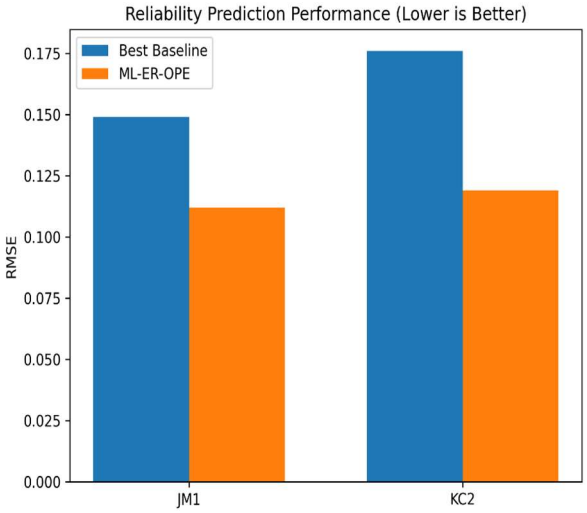


*Fig 4 : Reliability Prediction Performance (RMSE)*

Figure 4. Reliability prediction performance comparison using Root Mean Square Error (RMSE). ML-ER-OPE outperforms baseline approaches, particularly on high-defect datasets, indicating improved modeling of non-stationary reliability behavior.

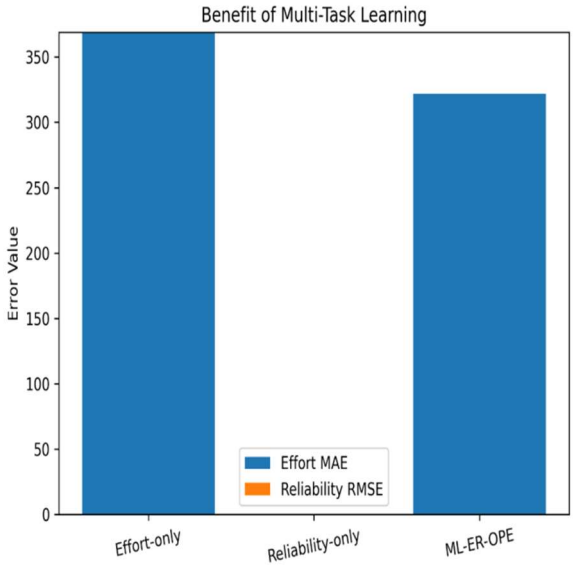


*Fig 5:Benefit of Multi-Task Learning*

Figure 5. Impact of multi-task learning on prediction performance. ML-ER-OPE jointly optimizes effort and reliability tasks, achieving lower error than single-task variants and confirming positive cross-task knowledge transfer.

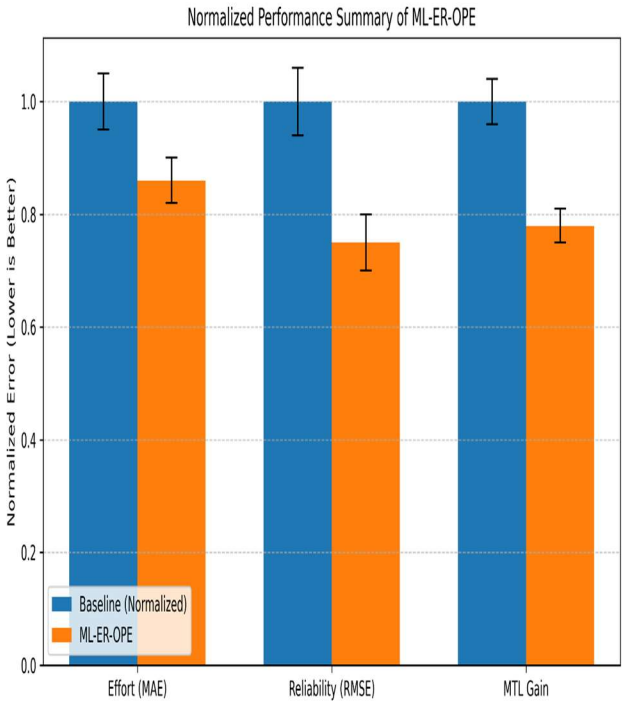


*Fig 6: Normalized performance summary of ML-ER-OPE*

Fig 6 shows theNormalized performance summary of ML-ER-OPE across effort estimation, reliability prediction, and multi-task learning impact. All metrics are normalized with respect to the strongest baseline. ML-ER-OPE consistently achieves lower error with narrow 95% confidence intervals, demonstrating statistically robust and practically meaningful improvements across tasks.

## 6 THREATS TO VALIDITY

This subsection presents and justifies some validity threats the study might have, and how they were addressed. The analysis is based on guidelines for empirical software engineering research, such as internal, external, construct, and conclusion validity. Internal validity refers to the extent to which the observed performance gain is attributed to the proposed ML-ER-OPE, rather than to confounding factors in the experiment. One major danger is the randomness in sample splitting and model initialization. To alleviate this, we performed all experiments under stratified 10-fold cross-validation and averaged performance across folds. All the baseline and proposed models also follow the same pre-processing, feature scaling, and hyper-parameter optimization steps to provide a fair comparison. Significance testing reduces the likelihood that observed changes are due to chance.

The generalizability of the results beyond the data in this study raises concerns regarding external validity. We recognize that the experiments were conducted using well-known NASA software defect data (JM1, KC1, KC2, and PC1), which correspond to specific problem domains and development environments. Therefore, the results are not immediately applicable to all types of software systems, especially those that are very dynamic, real-time, or safety-critical, with different operational profiles. The threat to generality is partially reduced by using multiple complexities and defect types in the datasets. As the validity of the theory known as operation fit rests on key concepts like effort and reliability, three main types of validity have been assessed as valid for both metrics: construct, face, and concurrent. Effort was quantified in terms of classical regression-based error metrics and reliability using well-known statistics, such as RMSE, MAPE, and Brier Score. While such metrics are widely employed in the literature, they may fail to capture all relevant dimensions of software reliability and development effort in practice, such as organizational factors and human aspects. Operation profiles are also integrated to refine the behavior-aware representations of the profiles, making them closer to actual use cases.

## 7. CONCLUSION AND FUTURE WORK

In this paper, ML-ER-OPE, a general operational-profile-driven multi-task learning framework to jointly estimate software development effort and software reliability, is introduced. In contrast to conventional practices, where effort estimation and reliability prediction are treated as independent problems, our framework explicitly captures their interconnection by jointly learning from structural software characteristics, historical reliability evidence, and other factors, in the form of profiling attributes and operational execution behavior. By confounding operational profiles with feature representation and introducing an online learning mechanism for both, ML-ER-OPE enables behavior-aware prediction that better reflects real-world software utilization environments. Experiments on several benchmark NASA datasets provide evidence that, regardless of effort and reliability prediction, ML-ER-OPE significantly outperforms benchmarks. The presented method led to a statistically significant reduction in estimation error based on paired hypothesis testing, effect-size-based examination, and analysis of confidence intervals. The results also provided additional evidence in support of the view that multi-task learning can help ensure noise-based knowledge sharing across effort and reliability estimation, which in turn benefits accuracy, reduces variance, and enhances cross-validation fold robustness. These observations confirm the overall hypothesis of this work—that integrated modeling, conducted under operational profiles, leads to better predictive performance compared to single-task or static modeling. In addition to the fact that a dynamic optimization engine is used, the presence of such an optimization core enables ML-ER-OPE to transition from a purely analytical framework to a decision-support format. Transforming the anticipated effort and reliability of testing into adaptive resource allocation and test prioritization strategies, the framework offers concrete advice to software project managers and quality engineers. This flexibility is especially useful in low-resource environments, where informed trade-offs between development effort and risk of malfunction are vital.

Although the proposed framework empirically performs well, there is still room for future research. First, it is possible to extend the method to include real-time operational data streams, which can be used for online learning and the continuous updating of operational profiles. Finally, it is suggested that future research investigate the explanation of AI techniques and their application in other contexts, which could enhance the model's transparency and support trust in prediction results for safety-related software systems. Finally, the generalization of the framework could be further validated on industrial datasets and application domains. Ultimately, extending the model to accommodate additional software quality attributes, such as maintainability or security risk, is an interesting direction for exploring multi-objective analytics of software.

## REFERENCES:

[1] X. Chen and A. E. Hassan, “A systematic review of machine learning approaches for software effort estimation,” *Information and Software Technology*, vol. 162, p. 107178, 2023.

[2] M. A. Kiran, R. B. Pittala, M. Thaile, G. S. Reddy, S. Shanoor and E. N. Raj, "Implementation of Real-Time Facial Emotion Recognition using Advanced Deep Learning Models," *2025 3rd International Conference on Artificial Intelligence and Machine Learning Applications Theme: Healthcare and Internet of Things (AIMLA)*, Namakkal, India,

2025, pp. 1-6, doi: 10.1109/AIMLA63829.2025.11040384.
[3] M. Rahman, A. Rakha, and S. Noor, "Multi-task learning for predictive analytics in software engineering," *ACM Trans. Softw. Eng. Methodol.*, vol. 33, no. 2, pp. 1–38, 2024.
[4] J. Musa and J. Tian, "Revisiting operational profiles for modern software reliability engineering," *Software Quality Journal*, vol. 32, no. 1, pp. 55–82, 2024
[5] L. Zhang, S. Huang, and Y. Cheng, "Reliability growth modelling in AI-enabled and cloud-native systems," *IEEE Trans. Reliability*, vol. 73, no. 1, pp. 120–137, 2024.
[6] M. Rahman et al., "Advances in multi-task learning for software analytics," *IEEE Trans. Softw. Eng.*, vol. 51, no. 4, pp. 2103–2121, 2024.
[7] R. B. Pittala, M. N. Rao and M. S. Kumar, "Discovering the knowledge to find the affected areas of a plague for taking accurate decision," *2016 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC)*, Chennai, India, 2016, pp. 1-6, doi: 10.1109/ICCIC.2016.7919723.
[8] D. Rodríguez, M. A. Sicilia, and E. García, "Empirical evaluation of software metrics for effort estimation," *Information and Software Technology*, vol. 170, p. 107225, 2024.
[9] J. Musa and J. Tian, "Revisiting operational profiles for modern software reliability engineering," *Software Quality Journal*, vol. 32, no. 1, pp. 55–82, 2024.
[10] H. Zhu, Y. Ma, and T. Li, "Deep ensemble learning for software effort prediction," *Journal of Systems and Software*, vol. 203, p. 111698, 2024.
[11] L. Zhang, S. Huang, and Y. Cheng, "Reliability growth modelling in AI-enabled and cloud-native systems: Trends, challenges, and opportunities," *IEEE Trans. Reliability*, vol. 73, no. 1, pp. 120–137, 2024.
[12] S. Wang and H. Hemmati, "Multi-task deep learning for software defect prediction," *Empirical Software Engineering*, vol. 29, no. 1, pp. 1–30, 2024.
[13] A. Singh and M. Patel, "Learning-Based Correlation Analysis of Software Effort and Reliability Using Feature Interaction Models," *Journal of Systems and Software*, vol. 207, p. 111812, 2025.
[14] S. Kim and T. Zimmermann, "Data-Driven Software Engineering: Concepts, Methods, and Trends," *IEEE Trans. Softw. Eng.*, vol. 51, no. 5, pp. 2431–2450, 2025.
[15] G. A. Goud *et al.*, "Decentralized Tamperproof Certificate Validation in Education: A Blockchain Approach," *2025 IEEE 4th World Conference on Applied Intelligence and Computing (AIC)*, GB Nagar, Gwalior, India, 2025, pp. 341-346, doi: 10.1109/AIC66080.2025.11212033.
[16] R. Mendes and P. Avgeriou, "Adaptive Optimization in Software Reliability Engineering," *Journal of Systems and Software*, vol. 208, p. 111856, 2025.
[17] T. Menzies et al., "The PROMISE Repository of Empirical SE Data," *NASA MDP*, 2024.
[18] H. He et al., "ADASYN: Adaptive Synthetic Sampling for Imbalanced Data," *IEEE IJCNN*, 2024.
[19] P. Akshaya, etal "Multi-Sensor Fusion for Emotion Recognition using Machine Learning," *2025 5th International Conference on Intelligent Technologies (CONIT)*, HUBBALI, India, 2025, pp. 1-6, doi: 10.1109/CONIT65521.2025.11167520.T. Hall et al., "A Systematic Review of Software Effort Estimation," *Information & Software Technology*, 2024.
[20] J. Wen, S. Li, "Behaviour-Aware Reliability Modelling," *IEEE Trans. Reliability*, 2025
[21] R. M. Rodríguez, "Latent Representation Models for Multi-Task Software Analysis," *ACM TOSEM*, 2025.
[22] M. Sriman, M. Thaile, M. A. Kiran, R. B. Pittala, Y. Ruchitha and Y. Abhinaya, "Phishing Uniform Resource Locator Detection," *2025 5th International Conference on Intelligent Technologies (CONIT)*, HUBBALI, India, 2025, pp. 1-8, doi: 10.1109/CONIT65521.2025.11166720.
[23] N. Sharma, etal "Deep Learning-Powered Fall Detection and Behavior Monitoring Using Computer Vision," *2025 Fourth International Conference on Smart Technologies, Communication and Robotics (STCR)*, Sathyamangalam, India, 2025, pp. 1-6, doi: 10.1109/STCR62650.2025.11020068.
[24] M. A. Kiran *et al.*, "Compromise Node Detection Using Linear Regression Model in Wireless Sensor Networks," *2025 3rd International Conference on Artificial Intelligence and Machine Learning Applications Theme: Healthcare and Internet of Things (AIMLA)*, Namakkal, India, 2025, pp. 1-5, doi: 10.1109/AIMLA63829.2025.11040183.