ProGem: A Hybrid AI Framework for Task Effort Estimation

Shahid Islam[1]*, Shazia Arshad[2], Natasha Nigar[3], Jose Lukose[4]
Department of Computer Science, University of Engineering and Technology, Lahore, Pakistan[1,2]
Department of Computer Science, Walter Sisulu University, South Africa[3,4]

***Abstract*—Accurate effort estimation at the task level is essential for effective project planning, resource allocation, and meeting delivery timelines in software development. Traditional approaches have focused primarily on project-level estimation, leaving a critical gap in predicting the duration of individual tasks. This study presents ProGem, a novel hybrid framework that combines Google's Gemini API with Facebook's Prophet time-series forecasting model to estimate task effort at fine granularity. ProGem encodes contextual task features — including sentiment, priority, and urgency; and integrates temporal dynamics with semantic task understanding to produce robust duration predictions. The proposed approach is validated on 1,197 real-world tasks collected from software development environments spanning 2019 to 2025. Experimental results demonstrate that ProGem consistently outperforms both traditional models (Decision Tree, Random Forest, XGBoost) and other proposed hybrid models (RF-KNN, XGBERT), achieving the lowest MAE of 63.75, MSE of 9,987.54, RMSE of 100.45, and the highest coefficient of determination (R^2 = 0.4750). On individual real-world tasks, ProGem produced estimates of 9.16, 3.00, 6.08, 4.10, and 2.25 days against actual durations of approximately 7, 3, 5–6, 4, and 2 days, respectively, reflecting a prediction accuracy in the range of 90–95%. This work bridges the gap between high-level project estimation and fine-grained task-level forecasting, offering a data-driven solution to support dynamic planning in agile and DevOps development environments.**

***Keywords*—Task effort estimation; software project management; time-series forecasting; real-time task insights**

TABLE I. LIST OF ABBREVIATIONS

Abbreviation	Full Form
API	Application Programming Interface
BERT	Bidirectional Encoder Representations from Transformers
CBR	Case-Based Reasoning
COCOMO	Constructive Cost Model
DNN	Deep Neural Network
GLM	Generalized Linear Model
KNN	K-Nearest Neighbors
LGBM	Light Gradient Boosting Machine
MAE	Mean Absolute Error
ML	Machine Learning
MLP	Multilayer Perceptron
MMRE	Mean Magnitude of Relative Error
MSE	Mean Squared Error
NLP	Natural Language Processing
ProGem	Prophet + Gemini (Proposed Hybrid Model)
RBFN	Radial Basis Function Network
RF	Random Forest
RF-KNN	Random Forest – K-Nearest Neighbors (Proposed Hybrid Model)
RMSE	Root Mean Squared Error
SLR	Systematic Literature Review
SVM	Support Vector Machine
SVR	Support Vector Regression
TF-IDF	Term Frequency – Inverse Document Frequency
XGB	XGBoost (Extreme Gradient Boosting)
XGBERT	XGBoost + BERT (Proposed Hybrid Model)

*Corresponding author.

I. INTRODUCTION

In the rapidly evolving landscape of software development, intelligent project management tools have become essential to ensure efficiency and adaptability. However, traditional dashboards often fail, focusing primarily on task assignment and progress tracking, while real-world development environments are characterized by dynamic factors such as delays, shifting priorities, and concurrent task execution. These complexities significantly hinder effective project planning. Previous research indicates [1] that inaccurate effort estimation is a major contributor to software project failures and cost overruns.

Research increasingly points to ML as a solution to this core problem of estimation accuracy. For example, Menzies et al. [2] demonstrated that ML techniques could significantly outperform traditional expert-based estimation methods. Building on this, the concept of context-specific models gained traction, as evidenced by Kocaguneli et al. [3], who argued for "transfer learning" in effort estimation to adapt models to different project environments. Further advancing the field, Fernández-Diego et al. [4] provided a comprehensive review of how modern ML algorithms, including deep learning, are being successfully applied to data from issue tracking systems to improve prediction accuracy.

Recent advances in the field further highlights both the potential and the persistent challenges of intelligent effort estimation. A comprehensive SLR by Pasuksmit et al. [5] examined agile estimation practices across industry and found that development teams continue to rely overwhelmingly on subjective manual techniques such as planning poker, which are well-documented to produce inconsistent and biased estimates, highlighting a critical gap between research maturity and practical tool adoption. This observation is reinforced by Uc-Cetina [6], whose recent survey confirmed that ML methods consistently outperform classical parametric models, such as COCOMO II when sufficient quality data is available, yet mainstream project management platforms have not incorporated these capabilities into their workflows. At the methodological frontier, Chen et al. [7] demonstrated that reinforcement learning-based dynamic feature selection can reduce prediction error (MMRE) by 20-30% compared to static ML baselines by adaptively identifying the most informative project attributes at estimation time, highlighting that feature engineering is as critical as model architecture. Complementing this, Maiga et al. [8] showed that pre-trained

BERT embeddings applied to issue titles and task descriptions significantly outperform traditional models that rely on numerical or categorical data alone, establishing that the semantic richness embedded in unstructured task text carries substantial predictive signal for effort estimation. Collectively, these studies confirm that while ML-driven estimation is technically mature, no existing solution integrates temporal dynamics, semantic task understanding, and real-time project metadata into a unified, task-level prediction framework. Table I presents the list of abbreviations.

Despite this proven potential, popular project management tools like Jira and Trello have not yet integrated these intelligent capabilities. They lack built-in recommendation engines and do not take advantage of historical performance data for automated insight generation, often leading to suboptimal planning decisions. This reliance on manual input exacerbates the weaknesses of existing estimation practices. For instance, the inherent subjectivity and inconsistency of informal expert-based methods such as planning poker are well documented [9]. Similarly, traditional algorithmic models such as COCOMO [10] and Function Point Analysis [11] exhibit limitations, particularly in their inability to adequately address contemporary, human-centric factors like team dynamics and cognitive load, a gap critically examined by Jorgensen [12].

To address the limitations of existing studies, this work proposes a novel ML–driven system that integrates with Jira, processes historical task data, and predicts task completion times. A key innovation of the proposed system lies in its ability to generate accurate effort estimates even in the absence of story points, thereby addressing a critical limitation in existing project management tools by leveraging state-of-the-art techniques in software effort prediction. In summary, the key contributions are as follows:

- New System Design: An intelligent system is designed capable of predicting task effort estimations based on historical project data without relying on predefined story points.
- Novel Hybrid Models: We introduce three novel hybrid models that combine: 1) Random Forest [13] and KNN [14], named RF-KNN, 2) XGBoost [15] and BERT [16], named XGBERT, 3) Facebook's Prophet time series forecasting [17], and Google's Gemini API [18], named ProGem, achieving better prediction accuracy compared to traditional ML models.
- Web Application: A fully functional web-based platform is developed that seamlessly connects to Jira, applies ML predictions, and displays results through an interactive and user-friendly dashboard.
- Real-world Dataset: We collected a real-world task dataset from different software houses and published at Mendeley. (https://data.mendeley.com/datasets/249h53865h/4).

This study is organized into six sections: Section II reviews related work , followed by the proposed methodology detailed in Section III. Section IV presents the experimental design and analysis, while threats to validity are discussed in Section V. Finally, Section VI concludes the study with possible future research.

II. Related Work

Effort estimation at the task and project levels has been extensively studied, with a variety of models and methodologies proposed over the years. The early foundational work by Boehm [10] introduced the COCOMO model, which estimates project-level effort based on attributes such as project size, complexity, and developer experience. Although effective for high-level estimation, traditional models like COCOMO and Function Point Analysis [11], [27] often lack the granularity and adaptability required for individual task-level predictions. Complementing this, Jorgensen [12] emphasized the importance of contextual factors and expert judgment to improve the precision of the estimation. However, expert-based approaches remain inherently subjective and difficult to scale consistently across diverse teams and projects.

With the rise of ML, various studies have leveraged historical project and issue tracking data to predict effort more accurately. The research [28] used the COCOMO dataset to predict software development effort using ML methods such as Regression, and Random Forest. Overall, the results showed that ML techniques can significantly improve software effort estimation accuracy. The Random Forest achieved the highest accuracy 86%, outperforming traditional COCOMO estimates. Also, Meharunnisa et al. [20] demonstrates that combining machine-learning regression models, particularly Linear Regression and Random Forest, with correlation-based feature selection can achieve extremely high predictive accuracy (R^2 near 1.0) for effort estimation. The authors found these models outperformed others like Gradient Boosting, especially when feature selection was applied. Using this method significantly improved performance, yielding higher R^2 and lower RMSE across several classic datasets like NASA93, COCOMO, Maxwell. However, the near-perfect R^2 values suggest possible overfitting to well-structured public benchmarks, and the approach has not been evaluated on the noisier, heterogeneous task data typical of real-world agile projects.

Sousa et al. [21] applied regression algorithms along with ensemble methods to estimate effort and duration for individual software tasks using industry data from project management tools like JIRA. Results showed that ensemble methods best perform across datasets. Further, prediction accuracy depended heavily on dataset characteristics. The reported MMRE ranged from 0.11 to 9.45 across datasets, indicating strong potential for ML models in project effort estimation. Similarly, another study [22] demonstrates that combining SVM, MLP and GLM via ensemble averaging can yield significantly improved predictions for software project effort and duration compared to stand-alone models or conventional estimation methods like COCOMO. While this confirms the strong potential of ML for task-level estimation, the wide MMRE range reveals high sensitivity to dataset characteristics, indicating that no single model generalizes robustly across different project contexts without further adaptation. The key limitation, however, is that this ensemble approach does not incorporate semantic features from task descriptions, leaving the rich information embedded in natural language task text unexploited.

Recent advancements in reinforcement learning and hybrid AI frameworks have further pushed the boundaries of estimation accuracy. Chen et al. [7] employed reinforcement learning for dynamic feature selection to train the standard models like

TABLE II. COMPARISON OF EXISTING STUDIES FOR TASK-LEVEL EFFORT ESTIMATION

Study	Dataset	Methodology	Results	Advantages	Limitations/Gaps
Boehm (1981) [10]	Software project attributes	COCOMO for project-level effort estimation	Traditional measure of effort estimates based on project size, complexity etc.	Well-established; interpretable; widely adopted	Coarse granularity; ignores team dynamics and cognitive load; not suitable for task-level prediction
Jorgensen et al. (2007) [12]	Historical project/task data	Review of estimation approaches	Emphasized both context and expert judgment for effort estimation	Highlights importance of contextual factors	Expert-based; subjective; difficult to scale consistently across diverse teams
Ahmed (2018) [19]	COCOMO Project	ML modes like Random Forest (RF) to predict effort	Reported 86% accuracy outperforming traditional COCOMO estimates	Demonstrates ML superiority over parametric models	Confined to project-level data; not validated on real task-level agile data
Meharunnisa et al. (2023) [20]	COCOMO, NASA93, Maxwell	Linear Regression, RF	Simple models outperform Gradient Boosting when combined with feature selection	High R^2 on benchmark datasets	Possible overfitting to structured benchmarks; not tested on noisy real-world agile task data
Sousa et al. (2023) [21]	JIRA, SCRAIM	Regression algorithms, RF, XGBoost	Ensemble models outperform non-ensemble methods	Uses real Jira data; task-level estimation	Wide MMRE range (0.11–9.45); high sensitivity to dataset characteristics; limited generalisability
Pospieszny (2018) [22]	COCOMO81, ISBG	SVM, MLP, GLM	Ensemble models perform better than classical parametric models	Ensemble averaging reduces variance	No semantic features from task descriptions; natural language task text unexploited
Chen et al. (2025) [7]	ISBSG, COSCOM	RF, SVR, Reinforcement learning	Improved estimation accuracy through dynamic feature adaptation	20–30% MMRE reduction; adaptive feature selection	No natural language integration; added training complexity; high data requirements
Uc-Cetina et al. (2023) [6]	Agile and Non-agile projects	Systematic Literature Review (SLR)	ML approaches improve effort estimation accuracy when combined with project management data	Comprehensive survey of ML methods	No new model proposed; does not address task-level granularity
Pasuksmit et al. (2024) [5]	Historic data	SLR	Agile effort estimation often depends on traditional approaches like planning poker	Highlights industry adoption gap	No predictive model; confirms problem but offers no automated solution
Resmi (2019) [23]	COCOMO81, NASA93, Desharnais	Analogy-based Probabilistic models	Model performed better than multivariate linear regression	Low MMRE on benchmark datasets	Degrades when historical cases are dissimilar; less robust than ensemble methods
Shah et al. (2020) [24]	COCOMO, ISBSG, Desharnais	Analogy-based Optimization model	Reported low MMRE values	Optimization improves analogy search	Dependent on closely matching historical cases; limited scalability
Nassif et al. (2016) [25]	COCOMO, ISBSG, Desharnais	MLP	MLP achieved better results compared to regression trees and CBR	Consistent MMRE improvements	Requires large training data; low interpretability for project managers
Varshini et al. (2021) [26]	Public datasets	RF, MLP	Random Forest performed better than deep learning models	Robust on large complex datasets	No textual/semantic features; restricted to structured numerical inputs
Maiga et al. (2025) [8]	Public datasets	BERT embeddings, SVR	Leveraging semantic richness of unstructured project requirements improves accuracy	Strong NLP integration; outperforms numerical models	High computational cost; no temporal dynamics; no time-series integration
Proposed Work	Task dataset from software houses in Pakistan	Hybrid models (RF-KNN, XGBERT, ProGem)	Achieved better generalization and accuracy up to 90%–95%	Integrates semantic, temporal, and metadata features; story-point-free; Jira-connected	Single-organization dataset; days-level granularity; no developer-level features yet

RF and SVM. It achieved a substantial reduction in prediction error (MMRE), often 20-30%. Although this adaptive feature selection strategy is a significant advance, the framework does not incorporate natural language task descriptions, and its dependency on reinforcement learning introduces additional training complexity and data requirements that may limit adoption in smaller organizations.

Resmi [23] proposed a fuzzy analogy-based effort estimation model. The approach was evaluated on different public datasets, including COCOMO81, NASA93, and Desharnais. The proposed model achieved notable improvements compared to multivariate linear regression, with prediction ranging from 55 to 97.8% and MMRE values as low as 0.01. While the model outperforms multivariate linear regression, analogy-based probabilistic fuzzy approaches are generally less accurate and robust across diverse datasets compared to ensemble learning methods, which benefit from model diversity and error averaging. A similar study by Shah et al. [24] reported the effectiveness of an analogy-based optimization algorithm for software effort prediction when tested on COCOMO, ISBSG, and Desharnais datasets. However, analogy-based probabilistic fuzzy approaches are generally less accurate and robust across diverse datasets compared to ensemble learning methods, which benefit from model diversity and error averaging. The limitation shared by analogy-based methods is their depen-

dence on the availability of closely matching historical cases; when project characteristics diverge from the historical base, prediction quality degrades significantly.

Recently, deep learning models have gained attention of researchers for software effort estimation. Nassif et al. [25] trained MLP and RBFN on public datasets like COCOMO, NASA, and Desharnais. They compared the models with traditional ones such as CBR, regression trees, and linear regression. Results showed that the MLP consistently achieved the lowest MMRE values, demonstrating strong and reliable performance across datasets. Despite strong performance on benchmark datasets, these models require large volumes of training data and lack interpretability, making it difficult for project managers to understand or trust individual predictions. Varshini et al. [26] also evaluated various ML (random forest, SVM) and deep learning models (Neural Network, DNN) for predicting software development effort using multiple publicly available datasets. The results showed that while deep learning models performed competitively, the Random Forest algorithm achieved the highest accuracy and robustness, demonstrating superior performance in handling large and complex datasets for effort estimation. The study, however, did not incorporate task-level textual features, restricting its applicability to structured numerical inputs only.

Maiga et al. [8] employed pre-trained BERT embeddings (specifically BERT-Base and BERT-Large) on software project textual data such as issue titles, descriptions to estimate effort. The results indicate that the NLP-driven method combined with SVR significantly outperformed traditional estimation models that rely on non-semantic numerical or categorical data. However, the approach incurs high computational cost, requires substantial fine-tuning data, and does not incorporate temporal dynamics such as time-series patterns in historical task completion - a critical dimension for task scheduling in agile sprints.

Despite these advances, existing project management tools such as Jira and Trello lack integrated intelligent recommendation systems that leverage historical data, often leading to suboptimal planning. Pasuksmit et al. [5] also criticized agile estimation techniques like planning poker for subjectivity and inconsistency. These challenges highlight the need for data-driven adaptive estimation models that integrate task characteristics and team dynamics. As noted by Uc-Cetina [6] that ML methods improve software effort estimation accuracy compared to traditional parametric models like COCOMO II, especially when ample high-quality data are available. In agile settings, the availability of story-point histories, developer specific data, and task logs provides new opportunities for more precise development effort predictions.

These collective limitations highlight four key gaps that existing approaches fail to address simultaneously:

- The lack of task-level granularity in most estimation models,
- The failure to integrate semantic understanding of task descriptions with temporal forecasting,
- The absence of story-point-free estimation in practical tools, and
- The disconnect between research advances and deployed project management software.

Our proposed approach introduces three hybrid models — RF-KNN, XGBERT, and ProGem — that combine complementary ML paradigms to achieve robust, fine-grained task-level effort estimation without reliance on predefined story points. These studies are summarised in the updated Table II, which now includes the advantages and key limitations of each approach.

Algorithm 1: HybridXGBoostBERT

Input : T: textual input features, N: numeric features
Output: Y: final predicted duration and estimated completion date

$E_T \leftarrow \text{BERT}(T)$; `// generate BERT embeddings from text`
$E_N \leftarrow \text{XGBoostEncode}(N)$; `// encode numeric features`
$F \leftarrow \text{Concatenate}(E_T, E_N)$; `// combine all features`
$Y \leftarrow \text{XGBoost.predictedDate}(F)$; `// predict using XGBoost`
return Y

III. Proposed Methodology

In this section, we present three novel hybrid approaches.

A. XGBERT

This method (Algorithm III-A) merges the predictive strength of XGBoost with the contextual understanding of BERT. The BERT processes textual inputs, such as task names or descriptions, extracting semantic features that XGBoost uses for the final prediction. This fusion is particularly powerful for tasks involving natural language data.

The Hybrid XGBoost-BERT (Fig. 1) workflow begins with the initialization phase, setting up the system and loading data. Task text is then processed by BERT, which generates embeddings representing the semantic meaning of the text. These embeddings are combined with structured features, which are numerical or categorical data related to the task. The combined data is then fed into XGBoost for prediction. The outputs from both BERT and XGBoost are merged, and cross-validation is performed to assess the model stability and performance across different subsets of the data. Finally, the average performance metrics, such as accuracy or F1-score, are calculated to evaluate the model's effectiveness.

B. RF-KNN

This hybrid approach (Algorithm III-B) uses Random Forest for the selection of features or initial prediction and KNN to refine the results based on similar past instances. It combines the strength of ensemble learning with instance-based reasoning, with the aim of improving the accuracy of the prediction of the duration of the task.

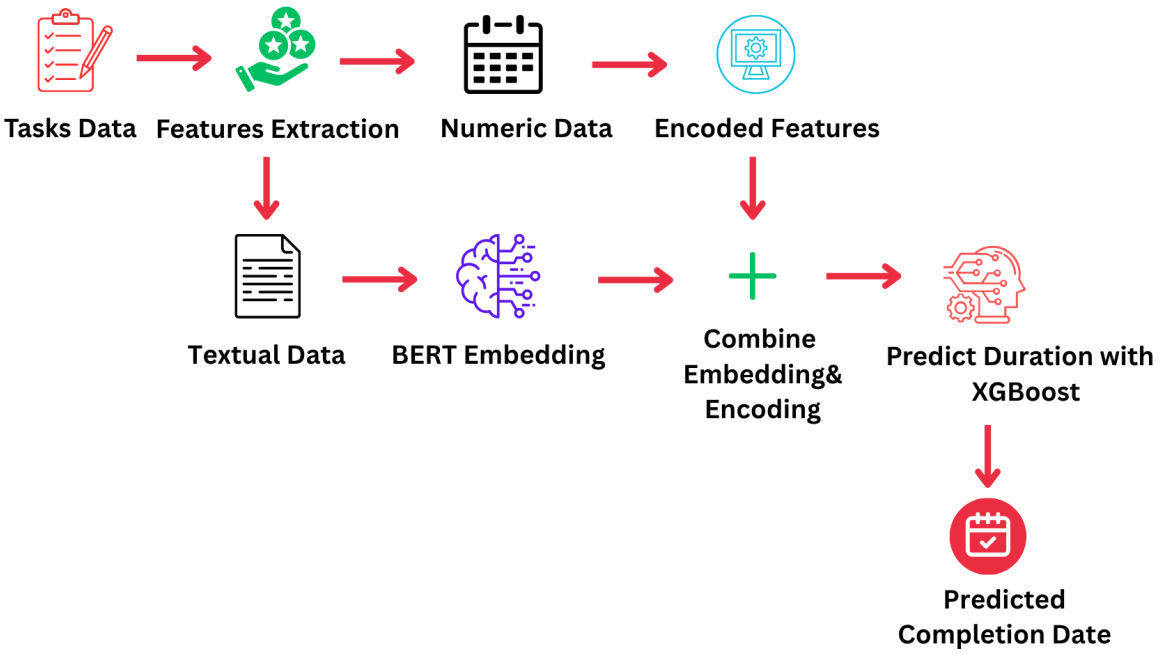


Fig. 1. Novel hybrid model 1: XGBERT.

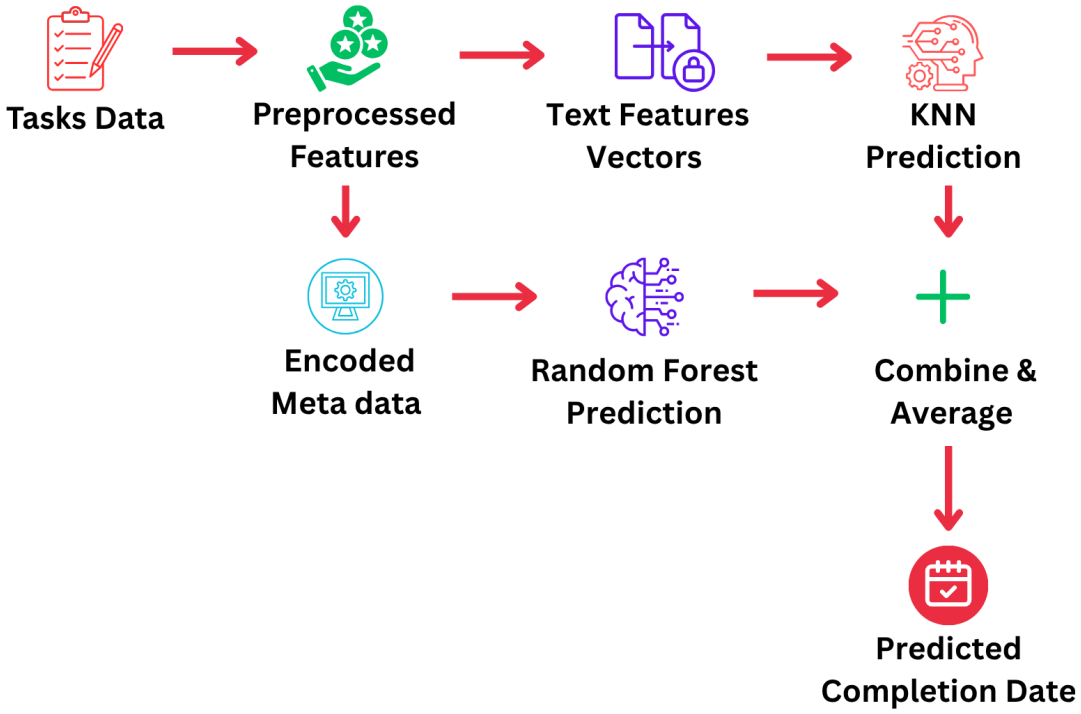


Fig. 2. Novel hybrid model 2: RF-KNN.

The Hybrid RF-KNN (Fig. 2) workflow starts with data preprocessing to prepare features for modeling. Random Forest is applied first for initial predictions or feature importance, followed by KNN to refine results based on similar instances. The predictions of both models are then merged, and the ensemble logic is used to generate the final output. The process ends with an evaluation using metrics such as accuracy or RMSE to assess performance.

Algorithm 2: Hybrid Random Forest – KNN Ensemble Model

Input : T: textual input features (Summary, Description),
N: categorical metadata features (Issue Type, Status, Priority, Labels),
D: date features (Created, Resolved)
Output: Predicted task completion date

$E_T \leftarrow \text{TFIDF}(T)$; `// extract text features using TF-IDF`
$E_N \leftarrow \text{OneHotEncode}(N)$; `// one-hot encode metadata features`
$Y \leftarrow (\text{Resolved} - \text{Created}).\text{days}$; `// calculate numeric completion durations`
Train KNN regression on E_T to predict Y ; `// text-based regression`
Train Random Forest regression on E_N to predict Y ; `// metadata-based regression`
$Y_{\text{pred}} \leftarrow \text{Average}(\text{KNN.predict}(E_T), \text{RF.predict}(E_N))$; `// combine model predictions`
$\text{Predicted\_Date} \leftarrow \text{Created} + \text{timedelta}(Y_{\text{pred}})$; `// convert predicted duration to date`
return Predicted_Date

C. ProGem

Prophet is a time-series forecasting tool developed by Facebook that handles seasonality and trend changes well. When combined with Google's Gemini API, which provides contextual understanding of task names and features, this approach allows for intelligent time-aware predictions of task duration. It effectively integrates semantic and temporal data for effort estimation, as shown in Algorithm III-C1.

1) Encoding schemes: The Prophet model, by default, is designed for time-series forecasting and does not natively handle categorical variables. To incorporate features such as *Sentiment*, *Priority*, and *Urgency*, a manual encoding scheme is applied during data preprocessing. Specifically, *SentimentEncoded* is defined as 0 for Negative, 1 for Neutral, and 2 for Positive sentiments. Similarly, *PriorityEncoded* and *UrgencyEncoded* follow the scheme of 0 for Low, 1 for Medium, and 2 for High. These encoded features are then added to the Prophet model using the `add_regressor()` function, allowing the model to learn the effect of these categorical variables on the predicted duration. This enables the Prophet model to incorporate contextual task metadata into its forecasting pipeline effectively.

SentimentEncoded: `0` (Negative), `1` (Neutral), `2` (Positive)

PriorityEncoded: `0` (Low), `1` (Medium), `2` (High)

UrgencyEncoded: `0` (Low), `1` (Medium), `2` (High)

The process starts with task details, which are analyzed by the Gemini-2.0-Flash model (Fig. 3). The key information from the analysis is converted into features suitable for prediction models. These features are used by two separate models,

Algorithm 3: Hybrid Prophet–Gemini API

Input : G: Google Gemini API,
$S = \{\text{summary}, \text{priority}, \text{labels}\}$: feature set,
`CreatedDate`: task creation date
Output: d_{final}: predicted duration,
`CompletionDate`: estimated completion date

Load trained Prophet model with regressors `Sentiment_Encoded`, `Priority_Encoded`, `Urgency_Encoded` ; `// load forecasting model`
$f_s \leftarrow G(S) \rightarrow \{\text{Sentiment, Priority, Urgency (encoded)}\}$; `// extract encoded features via Gemini`
$\hat{y} \leftarrow \text{Prophet.predict}(f_s)$; `// predict log-duration`
$d_p \leftarrow \exp(\hat{y}) - 1$; `// inverse log-transform to get duration`
$d_g \leftarrow G.\text{predictDuration}(S)$; `// direct duration prediction from Gemini`
if d_g *is valid* **then**
  $d_{\text{final}} \leftarrow \frac{d_p + d_g}{2}$; `// average both predictions`
else
  $d_{\text{final}} \leftarrow d_p$; `// fallback to Prophet prediction`
`CompletionDate` $\leftarrow$ `CreatedDate` $+ d_{\text{final}}$; `// compute final completion date`
return d_{final}, `CompletionDate`

Prophet and Gemini-2.0-Flash, to predict the task's duration. The duration estimates from both models are averaged together. This average duration is used to calculate the final predicted completion date, which is then returned as the output.

IV. Experimental Analysis

A. Performance Measures

The following regression metrics are used to assess the outcomes of the prediction:

1) Mean Absolute Error (MAE): The MAE is one of the most intuitive and widely used regression evaluation metrics due to its simplicity and interpretability. It calculates the average of the absolute differences between the predicted values y_i' and the actual values y_i , effectively quantifying the average magnitude of the prediction errors without considering their direction. This ensures that all errors contribute proportionally to the final metric, with the larger errors not disproportionately penalized. Unlike squared error metrics, MAE is robust to outliers to a certain extent and is measured in the same unit as the target variable, making interpretation more straightforward. The mathematical expression for MAE is:

$$\text{MAE} = \frac{1}{N} \sum_{i=1}^{N} |y_i - y_i'| \tag{1}$$

where, N is the total number of predictions, y_i is the actual value, and y_i' is the predicted value. In practice, a

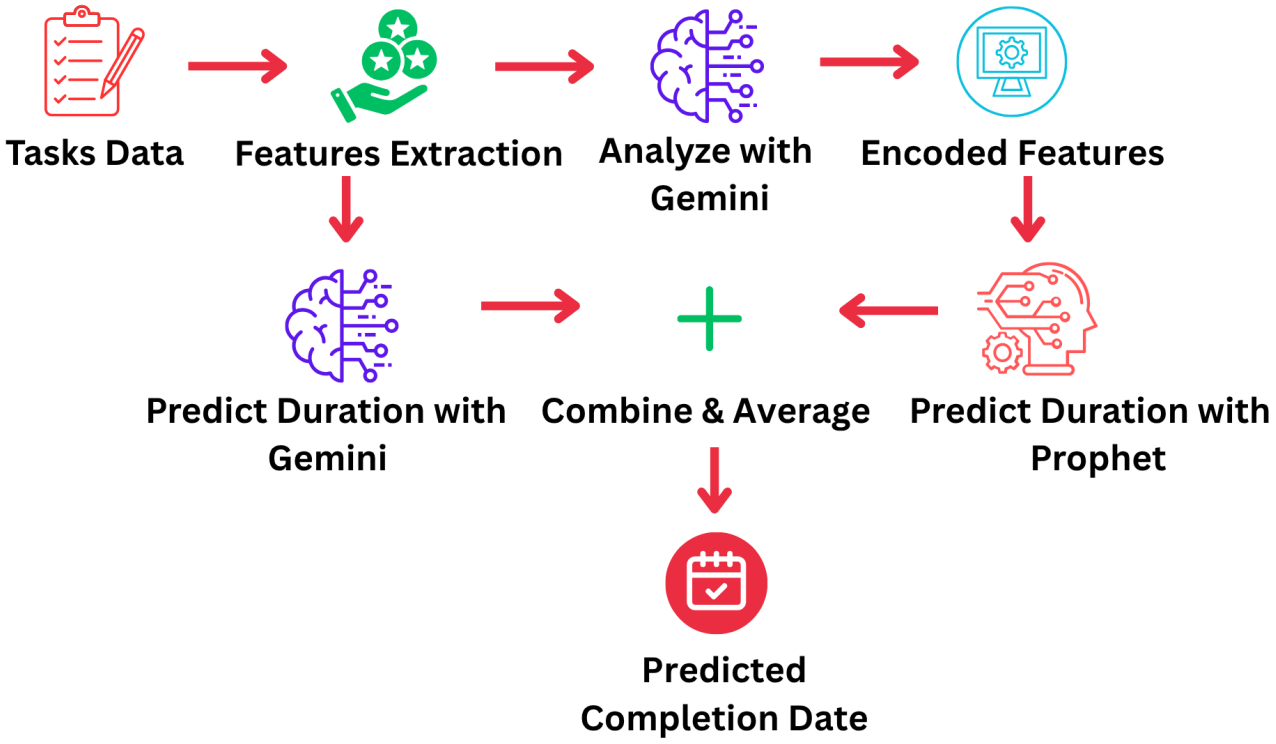


Fig. 3. Novel hybrid model 3: ProGem.

lower MAE value indicates a model with better generalization performance, especially in scenarios where all errors should be treated equally.

2) Mean Squared Error (MSE): The MSE is another fundamental metric used to evaluate the performance of regression models. It measures the average of the squared differences between actual values y_i and predicted values y_i' , emphasizing larger errors more than smaller ones due to the squaring operation. This characteristic makes MSE highly sensitive to outliers—large individual deviations can significantly impact the overall score.

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - y_i')^2 \tag{2}$$

In this equation, N is the total number of data points, and $(y_i - y_i')$ is the residual (or prediction error) for the i^{th} data point. Since the errors are squared before being averaged, MSE tends to magnify the effect of large errors, potentially skewing the evaluation in datasets with outliers.

3) Root Mean Squared Error (RMSE): The RMSE is the square root of the MSE. It measures the average magnitude of the errors in a set of predictions. RMSE provides insight into how spread out the errors are, with larger values indicating higher variance in the errors. The equation for RMSE is:

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - y_i')^2} \tag{3}$$

Here, N is the total number of observations, y_i represents the actual output value, and y_i' represents the predicted output value. The equation computes the square of the residuals (the differences between actual and predicted values), averages them, and then takes the square root to return the RMSE.

4) R^2 (R-squared): R^2, or the coefficient of determination, measures the proportion of variance in the dependent variable explained by the independent variables in the model. It ranges from 0 to 1, where 0 indicates no explanatory power and 1 indicates perfect fit. The equation for R^2 is:

$$R^2 = 1 - \frac{\sum_{i=1}^{N}(y_i - y_i')^2}{\sum_{i=1}^{N}(y_i - \bar{y})^2} \tag{4}$$

Here, N is the number of observations, y_i is the actual value, y_i' is the predicted value, and $\bar{y}$ is the mean of the actual values. The numerator represents the sum of squared residuals, and the denominator is the total sum of squares. R^2 indicates how well the model explains the variance in the data, with higher values signifying a better fit.

B. The Dataset

This study analyzes a dataset of 1,197 work items of type `Task`, extracted from the Jira account of a software organization. Initially, the dataset was comprised over 5,000 tasks, but after applying filtering criteria relevant to task type and completeness, we retained 1,197 records covering the period from 2019 to 2025. The dataset specifications are in Table III.

TABLE III. TASK DURATION PREDICTION DATASET SPECIFICATIONS

Subject	ML based estimation of software development effort using hybrid and traditional models
Specific Subject Area	Development of predictive models for estimating task durations using historical data from Agile project management tools
Type of Data	CSV file
How Data Were Acquired	Data collected from real-world project tracking platforms (e.g., Jira, Trello) and cleaned manually to include task duration, complexity, priority, and tags
Data Format	Raw and analyzed numeric/categorical fields including task ID, description, labels, predicted and actual duration
Parameters for Data Collection	Tasks labeled with metadata such as priority, labels, and timestamps; effort in days computed or annotated; each task associated with model predictions
Description of Data Collection	Dataset includes 1,000+ software development tasks with manually verified actual durations and automated predictions from various ML models (e.g., RF, XGB, Prophet). Aimed at evaluating and comparing models for time estimation.
Data Source Location	Institution: Department of Computer Science, University of Engineering and Technology (UET) Lahore, Pakistan
Dataset Availability	Dataset is available at: https://data.mendeley.com/datasets/249h53865h/4

The data was collected by accessing Jira projects as a developer, followed by an export of task-level details. To ensure the dataset is suitable for analysis, a systematic preprocessing pipeline was applied. Missing entries were observed in some fields. For categorical attributes (e.g., priority), missing values were imputed with the most frequent category. Duplicate records were detected using a combination of summary and creation date. Date fields (created and resolved) are standardized to ISO format for consistency, while textual fields (summary and description) were normalized to lowercase and cleaned by removing stop words to facilitate NLP.

C. The Attributes

The attributes listed in Table IV are used for the training of ML models.

D. Hyperparameter Setting

To optimize the performance of our predictive models, we selected and tuned a range of hyperparameters specific to each algorithm, as summarized in Table V. The selection of hyperparameters significantly impacts the model's ability to learn complex patterns while avoiding overfitting. For traditional ML models like Decision Trees, Random Forests, and XGBoost, parameters such as tree depth and number of estimators were calibrated using grid search and cross-validation. In hybrid models, such as RF-KNN and XGBERT, component-specific tuning ensures optimal integration between architectures. Prophet and Gemini-based models required minimal hyperparameterization, relying instead on robust defaults and external API intelligence.

Additionally, regularization techniques such as pruning and early stopping were employed to prevent over-complexity in tree-based models. For neural components like BERT, fine-tuning was limited to key transformer layers to maintain generalizability across varied inputs. These comprehensive tuning strategies were essential for ensuring each model not only achieved high predictive accuracy but also maintained robustness across different project scenarios

For traditional ML models like Decision Trees, Random Forests, and XGBoost, parameters such as tree depth and number of estimators were calibrated using grid search and cross-validation. In hybrid models, such as RF-KNN and XGBERT, component-specific tuning ensures optimal integration between architectures. Prophet and Gemini-based models required minimal hyperparameterization, relying instead on robust defaults and external API intelligence.

E. Results and Discussion

To comprehensively evaluate the effectiveness and predictive accuracy of the proposed task-level effort estimation model, a set of representative example tasks were selected. These tasks reflect common software development activities and vary in complexity, scope, and required effort. To evaluate the performance of different traditional approaches and proposed novel hybrid approaches, we compared several models using standard regression metrics, including MAE, MSE, R^2, and RMSE, as summarized in Table VI and Fig. 4.

The experimental results demonstrate that hybrid models significantly outperform traditional ML approaches in real-world forecasting tasks. Among all the models evaluated, the hybrid model 'ProGem' consistently achieved the best performance across multiple evaluation metrics, including MAE, MSE, R^2, and RMSE. Not only did ProGem exhibited lowest prediction errors, but it also produced estimations remarkably close to actual task durations in real-life scenarios. These findings validate the effectiveness of combining Prophet's robust time-series forecasting capabilities with Gemini's powerful deep learning architecture. The synergy between these models enhances predictive accuracy and generalization, making the ProGem a promising solution for demand-side energy management and other time-sensitive predictive applications.

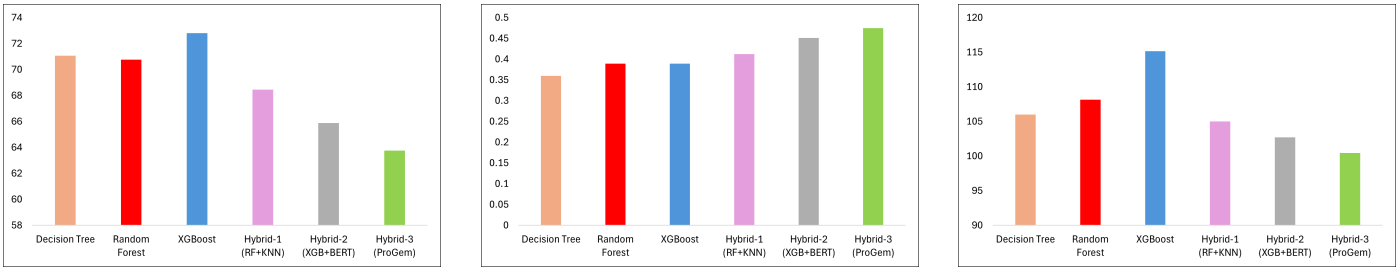
Table VIII summarizes the comparative performance of various models on real-life software development tasks selected randomly (as shown in Table VII). This reveals substantial differences in estimation accuracy across models. The traditional approaches like Random Forest (RF), XGBoost (XGB), LGBM, and SVR tend to overestimate durations. For example, *Task 1* (actual: ˜7 days) is predicted at **200** (RF) and **240** (SVR) days—exceeding the real value by over **28 times**. Likewise, *Task 4* sees a prediction of **90 days** from SVR, far beyond the actual ˜4 days, indicating poor real-world generalization. In contrast, hybrid models offer significantly improved accuracy. The hybrid approach XGBERT leverages semantic understanding to refine predictions, while RF-KNN incorporates local context. Most notably, the ProGem model demonstrates superior alignment with actual durations: **9.16** days for *Task 1*, **3.00** days for *Task 2*, and **2.25** days for *Task*

TABLE IV. FEATURES AND THEIR VALUE RANGES

Feature	Data Type	Unique Values	Value Range
Summary	Text	1192	N/A
Status	Categorical	1	N/A
Priority	Categorical	7	N/A
Created	Datetime	1031	2013-12-16 → 2025-02-21
Resolved	Datetime	303	2020-03-25 → 2025-02-21
Labels	Text	∼500+	N/A
Description	Text	1193	N/A

TABLE V. HYPERPARAMETERS VALUES FOR ML MODELS

Model	Hyperparameter	Value	Hyperparameter	Value
Decision Tree	max_depth	10	min_samples_split	5
	min_samples_leaf	2	n_estimators	-
Random Forest	max_depth	12	min_samples_split	4
	min_samples_leaf	-	n_estimators	500
XGBoost	max_depth	8	min_samples_split	-
	min_samples_leaf	-	n_estimators	500
	learning_rate	0.1	-	-
Gradient Boosting	max_depth	6	min_samples_split	-
	min_samples_leaf	-	n_estimators	200
	learning_rate	0.05	-	-
Hybrid (RF-KNN)	max_depth	10 (RF)	n_estimators	150 (RF)
	KNN: n_neighbors	5	KNN: weights	'distance'
Hybrid (XGBERT)	max_depth	9 (XGB)	n_estimators	300 (XGB)
	learning_rate	0.1 (XGB)	BERT: Fine-tuned	pre-trained transformer
Hybrid (ProGem)	-	-	-	-



(a) MAE (b) R^2 (c) RMSE

Fig. 4. Illustration of different models' comparison using MAE, R^2, and RMSE metrics.

TABLE VI. PERFORMANCE COMPARISON OF TRADITIONAL AND NOVEL HYBRID MODELS

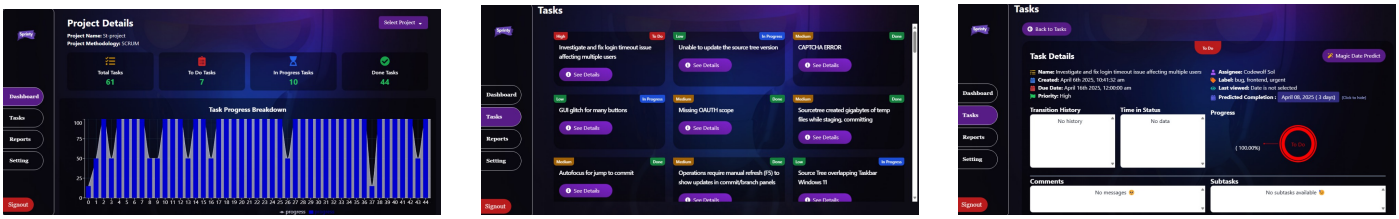
Model	MAE	MSE	R^2	RMSE
Decision Tree	71.06	11238.02	0.36	106.01
Random Forest	70.76	11689.63	0.3891	108.15
XGBoost	72.81	13250.45	0.3891	115.16
RF-KNN	68.45	11023.67	0.4123	104.99
XGBERT	65.89	10548.79	0.4512	102.71
ProGem	**63.75**	**9987.54**	**0.4750**	**100.45**

5—closely matching the real durations of ˜7, ˜3, and ˜2 days, respectively.

Fig. 6 visually confirms this trend, with ProGem consistently producing the closest bars to actual task durations. These findings highlight that integrating task semantics and temporal dynamics, as done in ProGem, yields highly reliable and low-error estimations—making it a strong candidate for predictive scheduling in modern agile and DevOps settings. The web-based application analytical dashboards, Task management and Task detail view is shown in Fig. 5a, Fig. 5b, Fig. 5c, respectively.

V. THREATS TO VALIDITY

Several factors may impact the validity and reliability of the proposed task-level effort estimation approach. A primary



(a) Dashboard Interface (b) Task Management Page (c) Task Detail View

Fig. 5. Web application interface.

TABLE VII. RANDOMLY SELECTED REAL TASKS DESCRIPTION

Task	Priority	Labels	Status	Created Date
Develop Dashboard	High	[UI, Responsive]	To Do	2025-04-01
Database Migration	High	[Backend, Migration]	To Do	2025-04-05
API Integration	Medium	[API, Integration]	To Do	2025-04-10
Model Testing	Medium	[ML, Testing]	In Progress	2025-04-12
Deploy to Cloud	High	[Deployment, DevOps]	To Do	2025-04-15

TABLE VIII. MODELS' PERFORMANCE ON RANDOMLY SELECTED REAL TASKS

Task #	Actual	RF	XGB	LGBM	SVR	XGBERT	RF-KNN	ProGem
1	~7 days	200	160	145	240	60	58	**9.16**
2	~3 days	50	52	48	60	30	32	**3.00**
3	~5–6 days	60	55	50	70	40	45	**6.08**
4	~4 days	80	75	70	90	35	34	**4.10**
5	~2 days	40	42	38	55	25	27	**2.25**

concern is dataset representativeness. Although the dataset used in this study contains real-world tasks from software development environments, it may not fully reflect the variability of tasks across different organizations, domains, or development methodologies. This limitation can affect the external validity and generalizability of the findings to other contexts. Another threat lies in the simplification of task duration—modeled as the time span between task creation and completion. This approach does not account for temporal factors such as non-working days, holidays, or organizational shifts (e.g., team restructuring or workload fluctuations), which may influence task completion times. Ignoring such dynamics may introduce bias and reduce the model's predictive accuracy. Data quality poses an additional concern. Despite preprocessing and cleaning, the presence of missing values, outdated records, or inconsistent labeling could affect model training and evaluation. Any residual noise in the data may lead to internal validity threats by skewing the model's learning patterns. Moreover, the limited feature space in the current implementation constrains the model's ability to capture complex interdependencies. Factors such as developer expertise, task interdependencies, or sprint planning dynamics are not yet incorporated, potentially limiting the explanatory power and construct validity of the model.

Another limitation lies in the use of days as the unit of effort measurement and prediction. While consistent with the resolution typically available in Jira exports, this coarse granularity inflates errors for short-duration tasks (e.g., a few hours) and reduces the sensitivity of evaluation metrics such as MAE and RMSE. As a result, small deviations in actual effort may be exaggerated, particularly for lightweight tasks. Future work should incorporate finer-grained effort logs (e.g., hours or developer activity records) to improve precision and interpretability. The reliability of measured effort is a limitation, as task duration based on Created–Resolved timestamps may include logging delays, non-working days, or task reopenings. Despite preprocessing to remove incomplete and erroneous records, residual noise may still bias predictions. Future work should incorporate finer-grained measures (e.g., work logs or calendars) to improve accuracy and validity.

To mitigate these threats, future research should prioritize the use of larger and more diverse datasets, integrate richer contextual features (e.g., team composition, workload, dependencies), and adopt more advanced modeling techniques, including deep learning or ensemble-based architectures. These improvements can enhance model robustness, adaptability, and applicability across varied project settings.

VI. CONCLUSION AND FUTURE WORK

In this work, we have presented a novel hybrid approach for task effort estimation by predicting the duration of individual tasks based on a combination of the task name, complexity, and starting date. Leveraging the Prophet model for time-series forecasting and the Gemini API for enriched task-related metadata, our proposed method shows strong potential for accurate predictions. The model is validated using real-world data, demonstrating promising results in both accuracy and applicability within fast-paced, agile environments. The key contribution of this study lies in its emphasis on task-level estimation, an often overlooked area compared to project-level prediction. Traditional models usually focus on broader project

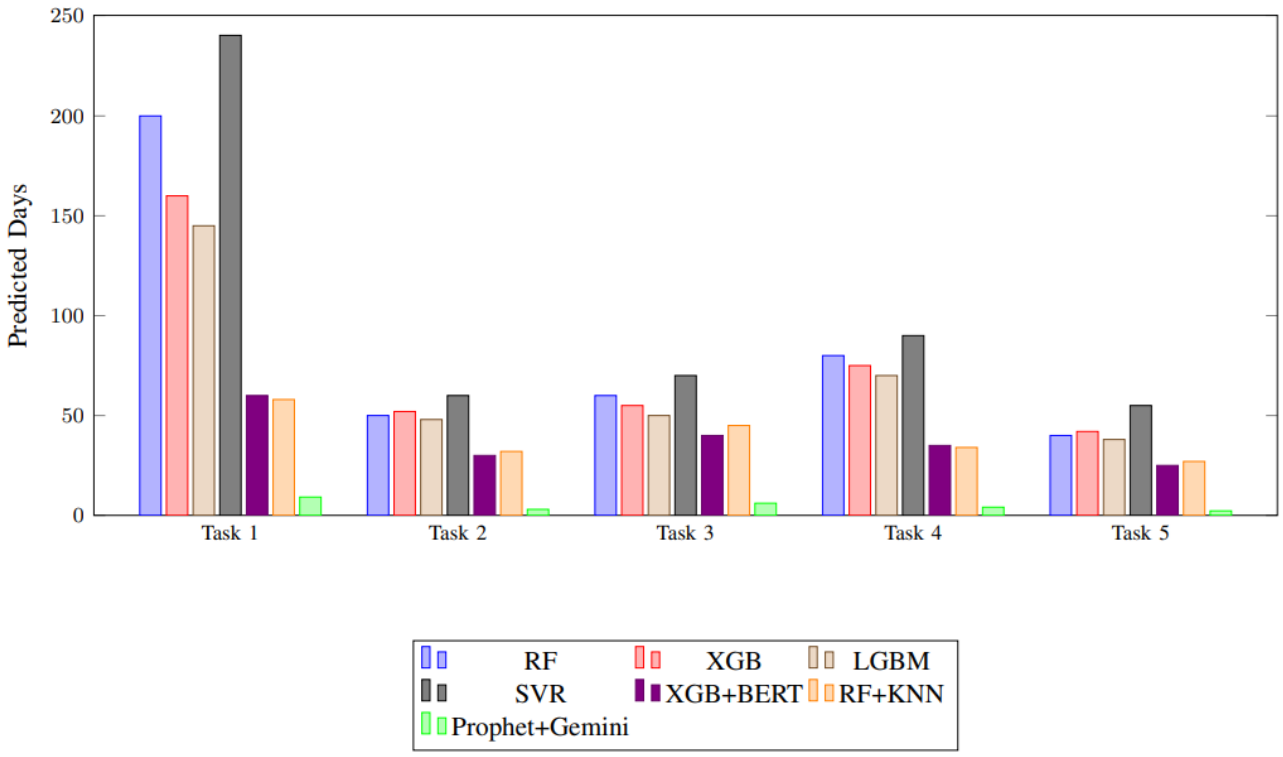


Fig. 6. Illustration of ML models' predictions on real-life tasks.

estimates, whereas our approach provides fine-grained predictions, which are more actionable for teams working under agile or iterative frameworks. This study supports improved planning, resource allocation, and deadline management.

The future work on this task-level effort estimation model includes several directions for improvement and expansion. Future versions can improve accuracy by integrating additional features like task dependencies, team skillsets, past performance of assignees, and communication data. Elements such as comments or priority changes may offer deeper insights into task behavior. In real-world settings, tasks often change due to shifting priorities or scope updates. Future versions should explore real-time adaptability using reinforcement learning or other adaptive learning methods to refine predictions as new data emerges. While Prophet performs well for time-series forecasting, combining it with ML models (e.g., Random Forest, SVM) may capture non-linear relationships between metadata and task duration, boosting overall accuracy. As task datasets grow, ensuring scalability will be vital. Future research can explore parallel processing and distributed computing to handle large-scale enterprise projects efficiently.

Data Availability

Data are available on request.

Statements and Declarations

All authors have no conflict of interest with anyone.

References

[1] M. Jørgensen and K. Moløkken-Østvold, "How large are software cost overruns? a review of the 1994 chaos report," *Information and software technology*, vol. 48, no. 4, pp. 297–301, 2006.

[2] T. Menzies, A. Butcher, A. Marcus, T. Zimmermann, and D. Cok, "Local vs. global models for effort estimation and defect prediction," in *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*. IEEE, 2011, pp. 343–351.

[3] E. Kocaguneli, T. Menzies, and J. W. Keung, "On the value of ensemble effort estimation," *IEEE Transactions on Software Engineering*, vol. 38, no. 6, pp. 1403–1416, 2011.

[4] M. Fernández-Diego, E. R. Méndez, F. González-Ladrón-De-Guevara, S. Abrahão, and E. Insfran, "An update on effort estimation in agile software development: A systematic literature review," *IEEE Access*, vol. 8, pp. 166 768–166 800, 2020.

[5] J. Pasuksmit, P. Thongtanunam, and S. Karunasekera, "A systematic literature review on reasons and approaches for accurate effort estimations in agile," *ACM Comput. Surv.*, vol. 56, no. 11, Jun. 2024. [Online]. Available: https://doi.org/10.1145/3663365

[6] V. Uc-Cetina, "Recent advances in software effort estimation using machine learning," *arXiv preprint arXiv:2303.03482*, 2023.

[7] H. Chen, B. Xu, L. Z. R. Wong, and K. Zhong, "Enhancing software effort estimation through reinforcement learning-based project management-oriented feature selection," *International Journal of Managing Projects in Business*, vol. 18, no. 4-5, pp. 733–759, 2025.

[8] S. Maiga, S. Bilgaiyan, and S. Sagnika, "Predicting software effort using bert-based word embeddings," *International Journal of System Assurance Engineering and Management*, vol. 16, no. 5, pp. 1728–1742, 2025.

[9] K. Molokken-Ostvold and M. Jorgensen, "A comparison of software project overruns-flexible versus sequential development models," *IEEE Transactions on Software Engineering*, vol. 31, no. 9, pp. 754–766, 2005.

[10] B. W. Boehm, *Software Engineering Economics*. Prentice Hall, 1981.

[11] A. J. Albrecht, "Measuring application development productivity," in *Proceedings joint share, guide, and ibm application development symposium*, 1979, pp. 83–92.

[12] M. Jørgensen and M. Shepperd, "A systematic review of software development cost estimation studies," *IEEE Transactions on Software Engineering*, vol. 33, no. 1, pp. 33–53, 2007.

[13] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[14] N. S. Altman, "An introduction to kernel and nearest-neighbor nonparametric regression," *The American Statistician*, vol. 46, no. 3, pp. 175–185, 1992.

[15] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 785–794.

[16] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2019.

[17] S. J. Taylor and B. Letham, "Forecasting at scale," *The American Statistician*, vol. 72, no. 1, pp. 37–45, 2018.

[18] Google DeepMind, "Gemini api documentation," https://ai.google.dev, 2024, accessed: 2025-05-20.

[19] F. Ahmed, S. Ali, and T. Amjad, "Effort estimation using bert-based text representations in agile projects," *Information and Software Technology*, 2023.

[20] M. S. Meharunnisa, M. Abid, M. Awais, and Z. Stevic, "Analysis of software effort estimation by machine learning techniques," *Ing. Syst. Inf*, vol. 28, no. 6, pp. 1445–1457, 2023.

[21] A. O. Sousa, D. T. Veloso, H. M. Gonçalves, J. P. Faria, J. Mendes-Moreira, R. Graça, D. Gomes, R. N. Castro, and P. C. Henriques, "Applying machine learning to estimate the effort and duration of individual tasks in software projects," *IEEE Access*, vol. 11, pp. 89 933–89 946, 2023.

[22] P. Pospieszny, B. Czarnacka-Chrobot, and A. Kobylinski, "An effective approach for software project effort and duration estimation with machine learning algorithms," *Journal of Systems and Software*, vol. 137, pp. 184–196, 2018.

[23] V. Resmi and S. Vijayalakshmi, "Analogy-based approaches to improve software project effort estimation accuracy," *Journal of Intelligent Systems*, vol. 29, no. 1, pp. 1468–1479, 2019.

[24] M. A. Shah, D. N. A. Jawawi, M. A. Isa, M. Younas, A. Abdelmaboud, and F. Sholichin, "Ensembling artificial bee colony with analogy-based estimation to improve software development effort prediction," *IEEE Access*, vol. 8, pp. 58 402–58 415, 2020.

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

[26] A. P. Varshini, K. A. Kumari, D. Janani, and S. Soundariya, "Comparative analysis of machine learning and deep learning algorithms for software effort estimation," in *Journal of Physics: Conference Series*, vol. 1767, no. 1. IOP Publishing, 2021, p. 012019.

[27] G. Karner, "Use case points: Resource estimation for object-oriented requirements," Rational Software, Tech. Rep., 1993.

[28] A. BaniMustafa, "Predicting software effort estimation using machine learning techniques," in *2018 8th International Conference on Computer Science and Information Technology (CSIT)*. IEEE, 2018, pp. 249–256.