AI-Powered Dynamic Task Ranking of User Stories in Agile Software Development

Khushnood Adil Rafique, Saroja Jigani Nagaraja, Christoph Grimm
University of Kaiserslautern-Landau (RPTU), Chair of Cyber-Physical Systems, Kaiserslautern, Germany
[khushnood.rafique|grimm]@cs.uni-kl.de, pip91bir@rptu.de

***Abstract*—In Agile software development, effective task prioritization is essential for ensuring timely delivery and efficient resource management throughout the software development life cycle. Traditional prioritization methods often depend on subjective decision-making, which can lead to inefficiencies. This paper discusses the challenges associated with conventional prioritization methods. It presents an AI-driven approach to automating the process, addressing new real-world challenges such as reducing subjectivity and adapting to changing requirements. The proposed methodology includes data collection, model selection, training the model, and tuning model parameters in machine learning models. It incorporates textual descriptions where feature extraction is essential, utilizing pre-trained Transformer models alongside structured features. These models will be evaluated using performance metrics, and inferences will be drawn from the results obtained. The results to be obtained are Task ranking, which has continuous values. It is a Regression model, where Linear Regression is considered a Base model compared with a tree-based model such as Random Forest and the XGBoost model. Our results demonstrate which XGBoost model has the strongest accuracy of 97%.In contrast, the Random Forest has 91%, and the Base model has 87% test accuracy. The Task is to be performed with high priority with the help of prioritization techniques, considering the specific factors to be fulfilled for the successful completion of the project across the organization. Explainable AI techniques, such as SHAP and LIME, will be implemented to enhance task ranking, transparency, and explainability. These techniques help interpret model predictions and highlight the contribution of each feature to the final rankings. The proposed approach improves task prioritization automating by reducing manual effort, enhancing decision accuracy and scalability, and ultimately contributing to the efficiency of Agile software development. This work contributes to the intersection of Agile software development and AI, paving the way for intelligent Agile project management and opening new avenues for AI-driven software engineering.**

***Index Terms*—Agile Software Development, Task Prioritization, Machine Learning, Linear Regression, Random Forest, XGBoost, Explainable AI(XAI), SHAP, LIME, RoBERTa, Software Engineering, Requirement Engineering, Task Ranking, AI in Agile**

I. Introduction

Agile software development is a well-known methodology characterized by iterative processes, flexibility, adaptability, a customer-centric approach, and continuous feedback, allowing for responsiveness to change[1]. Requirement Engineering is one of the most critical components of software engineering, and it consists of elicitation, analysis, Prioritization, and documentation of stakeholder requirements[2].

However, a pressing issue in Requirement engineering is that task prioritization presents a significant challenge, as it often depends on manual input and does not keep pace with changing dynamics. In real-world scenarios, teams frequently encounter many user stories, varying in urgency, business value, stakeholder influence, complexity, and required effort. Commonly used Prioritization techniques are: The analytical hierarchy process(AHP), MoSCow, cost value ranking, cumulative voting, planning game(PG), Kano model, numerical assignment, binary search tree(BST), bubble sort, value-oriented prioritization (VOP), Quality functional deployment (QFD), and Wieger's method. Each has its advantages and disadvantages. However, no single approach is always advantageous because it depends on the project environment and requirements. Another challenge is choosing the best technique for Prioritization, which can be selected by the project context, stakeholder understanding, and relevance of project criteria [2].

Agile methodologies have gained popularity in software engineering. Scrum and extreme programming are most used because they can integrate the requirement of Prioritization into the development process.

Introducing AI into agile software development has significantly enhanced decision-making, pattern recognition, and natural language processing while reducing human errors and mitigating risks. Agile methodologies have a pronounced impact on user requirements, which leads to the backlog being divided into smaller, actionable tasks for the development team [3].

Over the past few decades, the software development life cycle has shifted from traditional methods to agile processes, mainly due to the integration of AI. Artificial intelligence (AI) techniques, such as fuzzy logic, machine learning, and optimization, can help overcome the constraints in requirement prioritization. Machine Learning offers scalable solutions but may lack accuracy in minimizing disagreements. In the RE Prioritization process, there is a lack of automation, which leads to delays, greater expenses, and increased risk [2].

In agile development, numerous user stories need to be addressed, and effective Prioritization becomes crucial, especially when time is limited and project criteria are considered. Often, multiple tasks may be concealed within a single user story, making Prioritization essential for meeting deadlines based on domain expertise. There are several prioritization techniques present in agile software development. This thesis builds upon the foundational concepts and aims to bridge the gap between traditional agile methodologies and prioritizing software requirements at the task level within user stories.

It reduces time consumption, which leads to automated processes, and the involvement of domain expertise in ranking the tasks would be less compared to previous sitting and evaluation of tasks based on criteria and ranking them.

However, challenges arise in applying and collecting real-world datasets. In many cases, data sparsity and limited dataset size pose significant issues. Additionally, the imbalance within the datasets needs to be addressed.

Applying Machine Learning models for Prioritization of user stories at the Task level, in which Explainable AI like SHAP and LIME tells how each feature contributes to Rank predictions, enhancing Transparency

II. Related Work

A. Agile Software Development Background

Agile is the most crucial methodology used in the software development Life cycle(SDLC). It ensures rapid production and fast delivery in project management. It helps the Product to be released quicker at a lower cost. It is considered the workpiece in the form of iteration cycles. It ensures that the customer's requirements are fulfilled. Hence, the requirements keep changing. The requirement engineer plays an important role . It requires continuous customer feedback. In Agile methodology, team members will directly engage in the development activities and cross-functional teams, which requires skills. The agile method includes the client throughout the phases of product development[4]. Software development in agile projects involves the user requirements known as user stories. These user stories are built on a Principle known as INVEST, which is Independent, Negotiable, Valuable, Estimable, Small, and Testable[5]. In the latest year, agile methodologies provided flexibility, adaptability, and efficiency in delivering high-quality software[6].However, it introduces a few specific challenges in conflicts among the stakeholders, such as requirement prioritization techniques, rework due to changes in priority, and some factors that affect those appropriate requirements[7]. the "agile manifesto" was written by the practitioners reveals which items are considered Valuable by ASDM[8]

More Valuable Items		Less Valuable Items
Individuals and Interactions		Processes and tools
Working software	over	Comprehensive Documentation
Customer collaboration		Contract negotiation
Responding to change		Following a plan

Fig. 1: Agile Manifesto ***This figure is reproduced from*** [8].

B. Major Agile benefits in comparison to the Traditional approach

In this section, we highlight some of the advantages of the agile approach compared to traditional methods:

Evolutionary Approach: The entire application is developed in incremental units known as iterations. Each iteration builds upon the previous one. These iterations are small and fixed, focusing on development time. Agile employs a short, iterative cycle, while traditional approaches are data-oriented and follow a serial process that lacks agility.

Lightweight methods: Agile practice is best suited for small-scale, customer-focused, and lightweight development. The traditional approach is a heavyweight development method, involving extensive planning, heavy documentation, long timelines, and multiple design phases.

High Tolerance for Changes in Requirements: The ability to respond to changing requirements often determines a project's success or failure. Traditional methods tend to freeze product functionality, whereas Agile is designed to accommodate changes flexibly, thanks to its iterative nature. One of the core principles of the Agile Manifesto is to "Welcome changing requirements, even late in development."

Prioritizing Requirements in Agile Development: Agile methods decompose tasks and prioritize requirements during development using various prioritizing techniques. These techniques effectively accommodate frequent changes in priority. The client continuously provides requirements and is responsible for prioritizing these features. In contrast, the traditional approach delivers features only upon completion of the project.

Active Customer Involvement and Feedback: Active involvement from customers reduces risks by addressing the most critical problems first. In the agile approach, there is a significant emphasis on face-to-face communication and the participation of the product owner, unlike in the traditional approach. Feedback plays a crucial role in agile software methodologies.

Reduce cost and time:The development team reported that the waterfall model was an "unpleasant experience." In contrast, they found Extreme Programming (XP), an agile method, to be "beneficial and a good initiative from management." The agile approach incurs lower rework costs compared to traditional methods, which helps reduce overall costs and time.rework compared to the traditional approach.

The short design phase involves early client feedback: In conventional methods, follow-up is based on Big Design Up Front and Big Requirements Up Front development techniques. In these Techniques, the requirement and design document are developed in the early life cycle of the project, usually taking months if not years. In the agile approach, design is simple, which involves designing for battle, not the war. Due to a short development life cycle, it is achieved through an iterative and incremental process.

Self-organized Team: Agile teams are cross-functional and cooperative without considering the hierarchy. Team members take responsibility for the tasks to deliver functionality in the iteration."Fewer people are needed for the lighter methodology is used, and more people are needed if the heavier methodology is used", is needed.

Documentation: Agile approach had a strong belief in minimal documentation for the view exchange of the product, which gives more importance to developing the application. It has improved productivity, reduced development cost, and time-to-market.

Design Simplicity and improved software quality: Reduced software project overruns, fulfillment of customer satisfaction, Lower defect rates, and faster changing requirements. Its primary objective is the regular and continuous customer interaction between the customer and developers—the strong technical focus results in testing the agile project module, which provides a positive impact.

Enhanced business value, visibility, adaptability, and reduced costs are achieved through continuous feedback and development. This process accelerates the delivery of initial business value and enables the frequent rollout of implemented features. Agile methodologies offer these features much earlier in the project lifecycle by providing working, tested, and deployable software in an incremental manner.

Success possibility rate: Most software projects fail because they fail to meet the objectives, such as time, cost, features, and all the above. The conventional approach fails to meet the predictability, but agile practices have gained higher insights into project success rate, risk management, and user acceptance of the changing requirements. Agile determines which parts are needed for the customer.

C. Methodologies in Agile

1) Traditional Methodologies: Traditional software development methods, such as Waterfall, Prototyping, Spiral, and Rapid Development models, are often preferred because of their clear and structured approach. These models emphasize planning, process, documentation, and detailed design. However, they can result in inflated budgets, missed deadlines, and products that do not meet expectations[3]. As a result, these methods may not be suitable for today's fast-paced business environment, which demands greater adaptability and quicker growth.[6]. Traditional methodologies, such as life-cycle-based approaches and object-oriented techniques, dominate system development. Traditional Software Development Methods (TSDM), including waterfall and spiral models, are often referred to as heavyweight development methods. Their heavyweight nature can lead to a rigidity in product functionality, making it difficult to implement changes. These methods typically deliver software only after the entire development process is complete, often resulting in a lack of clarity for clients throughout the project[8].

2) Agile Methodologies: Scrum is one of the widely adopted, most dynamic, and adaptable integral frameworks today aspects in agile software development. Ken Schwaber introduced this model[6]. It is an Iterative approach. The most attractive part of the scrum is the "sprint," which typically lasts for two weeks in iterative cycles. The product owner will prioritize requirements[9]. It follows a customer-centricity[6].

Extreme Programming (XP) is a key methodology in Agile software development. It Focuses on Quality and responsibilities for evolving customer requirements, and it has four stages: Design, code, Test, and release. A maximum of two to ten members can be involved in the project This model shows the Drastic change from conventional approaches and gives broader Transparency.

Feature Development(FDD) is also an iterative and incremental approach but lightweight. It consists of five basic features: advance model, construct feature list, plan feature, design by feature, and build by feature.It is not only client-centric but also follows the process.Compare to other agile methodology it has upfront planning, Design and stakeholder feedback.It follows the disciplined approach which accomplish the success of the project. Development System Development Methodology (DSDM) is not client-centric but functionality-centric. The Main Focus is to identify resources and constraints of the functional components. It is a structured approach that consists of various software packages and non-IT solutions areas. It attains high quality and fulfillment of the project within the deadline period.It provides continuous communication.

Kanban is a dynamic and adaptable framework for agile methodology. It contains central component named as work-in-progress(WIP). It produces the outcome of the highest quality in place and phase. The term Kanban indicates visual signs or cards in Japanese.

Adaptive software development (ASD) overcomes the challenges that arise due to high performance and modifications in the outer world, and it targets the complexities present in large-scale software development. James A. introduced this approach in traditional approaches fallback due to business benefits and emerging markets. ASD has overcome these challenges by adopting a dynamic approach[6].

D. Prioritization Techniques in Agile

Several studies have been devoted to requirements prioritization techniques, and knowledge of experts states that business value is the most critical factor in prioritizing agile requirements.

The most cited techniques in agile software Development are: Moscow stands for must-have, should-have, could-have, and won't-have requirements.[10] It has high importance towards business benefits and is easily scalable for more minor and more significant projects; however, it fails to categorize the should or could requirements as essential than others[11]. According to the case study, the MoSCoW method outperformed the AHP Prioritization Technique [2].

The analytical hierarchy process(AHP) calculates the relative importance of the Requirements using a pairwise comparison matrix. It gives reliable results but is a time-consuming process. This technique lags in larger-scale projects [11].AHP gives trustworthy results; hence, it is considered for the decision-making process, helps reduce bias, clarifies the decision, and notes communication and dependency in distributed agile teams [2].

Cumulative Voting, which has a simple and effective way to prioritize the requirements, the highest Voting among all the other requirements in the list based on domain expertise, will be considered for further implementation in the project. It is also known as the 100 Point Method cite alhenawi2024choosing. Its practical application is used in the student finance system to prioritize the criteria. There was an issue with practical Voting, and it doesn't work well in larger datasets [12].

Kano Model regards quality as persisting in five components: satisfaction, attractiveness, indifference, and reverse.

Customer satisfaction is paramount with trust-based service, but it can not provide practical new features, but is good at analyzing [11]. The results of the Kano model are visualized with the help of TreeMaps, but they lack precise quantitative prioritization in real-world applications.

Cost-value ranking defines which tasks or features to prioritize based on the potential cost and value, and is a good fit for agile. In software requirement prioritization, which facilitates the incorporation of experts' qualitative assessment, considers cost and benefit constraints [13]. It is a long process of longing, and managing interdependencies and complexity in requirements is challenging. cite saher2018review.

Planning Game is used in software planning and is key for practicing extreme programming. It consumes less time and is user-friendly compared to pairwise analysis. Prioritizing the requirements through sorting and ranking. It has a demerit of a scalability issue [2].

Pairwise analysis is a decision-making Technique in which requirements are compared in pairs. This approach is helpful in ranking features, User stories, or tasks based on their relative importance and value. It is a tedious process and lacks scalability[11]. This method was compared with the other method, and findings found that tool-wise supported pairwise comparisons were the fastest and easiest comparison with user-friendly[14].

Value-oriented prioritization is critical in business value and customer choice. However, it often neglects the requirements' dependencies, which may not apply to a larger project.

Karl Wiegers introduced Wiegers' prioritization Method. It scales from the Range 1 to 9. We are penalized if the requirement is missing; the client and the customer will provide the required value[10]. Agile emphasizes and values effort, which ranks requirements according to value, cost, and risk. It improves decision-making and reduces dependence on subjective judgment.

Weighted Short Job First(WSJF) helps the team members get insights into which tasks or features must be worked on and first, based on two Factors: Effort, also known as the Job size required for implementation, and cost of delay, which contains factors like Business value, Urgency, and Risk Reduction.

The numerical assignment is a straightforward and rapid requirement prioritization technique, making it well-suited for a medium-sized dataset. It is flexible enough to accommodate stakeholders with varying levels of expertise. However, this method relies on subjective input and may lead to increased complexity [2].

A binary search tree is applicable for larger datasets. A real-world implementation of an agile-based e-governance project prioritized geographically distributed stakeholders, but it came with challenges related to complexity and bias [2].

Quality function deployment meeting the client's features is a main priority in this method . It improves decision efficiency and cooperation with customer needs. It also lacks the scalability and complexity cite tasneem2025enhancing.

The MoSCow technique is the most cited paper compared to the other methods. Second, the most reliable technique, the analytic hierarchy process(AHP), is used since it has the feature of pairwise comparison, which optimizes the decision-making process because it has complexity, slow calculations, and scalability issues. MoSCow stands first—each technique has a unique strength and weakness. According to the project team, they can choose a suitable prioritization technique

E. Challenges in Requirement prioritization Techniques

Agile software development has many advantages and a high success rate. Requirement engineering is an essential part of any software development life cycle. The significant challenges associated with the requirement prioritization in agile software development:

Scalability Concerns: One primary concern is the inability to manage the growing requirements. As the volume of requirements increases, issues arise, such as a shortage of human resources and the lack of an intelligent automated process to effectively evaluate the criteria.

Continuous Requirement change: The most critical challenge is handling the dynamic change in the requirements. Existing techniques are insufficient to capture the changes in requirements and adapt them to the project lifecycle. The literature lacks the prioritization methods that can handle the continuous changes. Dealing with constant changes requires an adaptive approach so that the project aligns with changing priorities.

Stakeholder communication and collaboration: Most prioritization strategies fail due to ineffective communication and collaboration among stakeholders, which is essential for the project's success. Each stakeholder brings their perspective to the discussion, which can result in delays, overspending, or failure to meet customer needs. Nonetheless, the involvement of multiple stakeholders is essential; it creates opportunities for conflict and presents another complex aspect to address in achieving the project's objectives [2].

Ranking the Requirements: The rank of the requirements needs to be updated continuously whenever new requirements are added. Many prioritization techniques fail to update the rank list or display the rank status for the new requirement.

Handling interdependencies and dependencies: The sequential chain from one requirement to another requirement, which carries impact as well from the previous one, in which several components are interconnected, leading to an increase in costs, due to a lack of attention towards non-functional requirements, which indirectly affects the software quality of the products.

By addressing all these challenges, research may focus on improving the prioritization techniques in agile software development.

F. AI Driven Approaches

AI has significantly enhanced computer engineering by automating processes, decision-making, improving performance, and solving complex problems. Several prioritizing techniques require a lot of human effort, are time-consuming processes, and are not scalable for efficiency[15] Some AI-based techniques developed for prioritizing the requirement include fuzzy logic, Machine learning, and optimization algorithms

1) Fuzzy Logic Based Approach: Fuzzy logic-based approaches: It defines uncertainty by transforming ambiguous concepts into mathematical solutions. They simplify control theory and highlight real-world objectives. Fuzzy logic is applied in software engineering for effort estimation, development, and requirements engineering. The techniques we find are based on fuzzy-based approaches in prioritization concepts[2].

Few authors explained the MosCoW approach to fuzzy environment for requirement prioritization. It was applied to the Library Management system to prioritize functional and non-functional factors by using the logic of triangular fuzzy numbers, obtaining decision markers, applying graded mean integration, computing the ranking values of requirements, and improving prioritization accuracy[16].

In the Value-Based Intelligent Requirement prioritization(VIRP) study, fuzzy logic is used, yielding improved accuracy and consistency in ranking. Studies show that VIRP is a flexible and reliable method in software development . It ensures the criticality of the software products is met successfully[17].

To overcome the current prioritization issues, such as a lack of automation and dependence on domain expertise in decision-making. The proposed approach of value-based Fuzzy requirement prioritization provides automated prioritization and less dependence on domain expertise[2].

In multi-person decision making using Fuzzy AHP, in which stakeholder weights are determined using eigenvalue, systematic constraints may not be fulfilled to satisfy the customer needs. Even though solutions were complex, they require expert input for fuzzy rule formations, as applied in website travel management[18].

2) Machine Learning: Conventional Requirement prioritization Techniques are quite time-consuming. It isn't easy to learn the patterns in larger datasets. Machine learning has been applied in other applications to understand patterns and larger datasets after training the model to recognize patterns and evaluate them.

Software Requirement Prioritization Techniques using Machine Learning tell about software release planning to tackle the conflict between stakeholder priorities, which was applied to decision trees, Random Forests, and (Support Vector Machine) SVM, evaluating using metrics such as accuracy, F1 score, and cross-validation. It concludes machine learning models can predict accurately and improve software planning. They concluded that hyperparameter tuning should be done in future study investigations[19].

Approach for workflow optimization in which an attempt was made to exploit the lowest level information extraction by prioritizing the higher level granular flow. They created the model to provide valuable information in workflow optimization rather than passing the whole documentation. This study found that XGBoost performed better than all other models based on classification metrics. It has a true positive. They obtained a high degree of granularity. In this paper, they trained the model using consumer financial domain data from 2018[20].

Hybridized Approach Based on K-means and Evolutionary Algorithms: The significant disadvantage is a mismatch in ranked requirements and stakeholder linguistic ratings, which leads to an eventual fall in user requirements. This paper highlighted the correlation between prioritized requirements and stakeholder linguistics. The Algorithm uses the dataset RALIC with relative weights and requirements[21].

Facing scalability issues in machine learning, Case-based Ranking presents a unique framework for requirement prioritization that differs from AHP (Analytic Hierarchy Process). Using a paired preference elicitation approach allows for a broader selection of requirements. This method is enhanced by deriving ranking techniques from a machine learning algorithm that overcomes the scalability issues. The results from the case-based ranking paradigm were compared to AHP's on average, demonstrating that it performs better, particularly when handling extensive requirements[22].

Case-Based Ranking for Decision Support Systems: To reduce the difficulty in rank evaluation planning, a Case-Based procedure to reduce the time consumption on elicitation and simplify comparison emphasizes the strict, precise prediction rules. It uses two automated steps: one to create the case selection strategy and the second to make comparisons to collect preferences. The rank boost method successfully compares better elicitation effort and precision[23].

3) Optimization Algorithms: Compares the different solutions repeatedly to obtain the best optimal or appropriate solutions. There are several techniques to provide software requirement prioritization.

One effective method is the Ant Colony Optimization Approach, a swarm intelligence framework used in software release planning in the presence of different requirements and tackles the traveling salesman problem. Experiments have been conducted to compare its efficiency against Simulated Annealing and Genetic Algorithms. This approach generates accurate solutions while minimizing computational costs. Future research may involve using real-world samples for further analysis[24]

Interactive Requirement Prioritization using Genetic Algorithms utilizes the pairwise comparison technique for prioritization. It outperformed the AHP and the conventional genetic algorithm by handling the larger dataset with numerous requirements. It can scale up to 50-100 requirements pairwise in case the original ranking was lacking. It is suitable for reliable input and compromise where user effort and performance are performed in case of failures. In the case of the Interactive Genetic Algorithm, it overcomes the current limitations, such as dependencies and priorities, and adds an informational gain. One good thing about IGA is that it can provide a good approximation of the cost, desirability, development time, and revenue and maintain complicated constraints The EVOLVE method provides continuous software development and planning to rank the software requirements. It has a risk optimizer tool within the genetic algorithm, which helps make the final decisions. Non-dominated Sorting Genetic Algorithm II (NSGA-II)is a fast and multi-objective genetic algorithm that improves computational efficiency and maintains the optimized solution by eradicating high computing

costs. Its strength includes faster sorting and better constraint handling[2].

G. Scaling Agile Practices

In Agile practices, weighted score of each user story is calculated. C_j represents the criteria that can be considered by the product owner during the calculation of the weighted score of user stories. w_j denotes the relative weight of the criterion, and a_{ij} is the importance value of user story U_i with respect to criterion C_j .

$$\text{Weighted Score}(U_i) = \sum_{j=1}^{n} w_j \cdot a_{ij}, \quad \text{for } i = 1, 2, 3, \ldots, n$$

The importance value, i.e., a_{ij}, can be calculated using an ordinal importance scale[25].

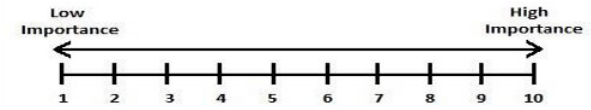


Fig. 2: Ordinal Importance Scale ranging from Low (1) to High (10) ***This figure is reproduced from*** [25].

The effort estimation technique is Agile Software Development using Planning Poker, which uses the Fibonacci scale to evaluate complexity, risk, and uncertainty. There is a widespread use of the Fibonacci scale in agile. There was a comparison between the linear scale and the Fibonacci scale. Studies have found that the Fibonacci scale outperforms in a few criteria. However, the linear scales are applicable and convenient for different criteria[26].

The pairwise comparison technique estimates the relative importance between all unique pairs by comparison to decide which requirement is the most important using a scale from 1 to 9 in the Analytic Hierarchy Prioritization Technique. Another Technique is Wieger's method, where the value of the requirement depends upon the value provided by the client to the customer, and penalty occurs if the requirement is missing were attributes are scaled from 1 to 9 All the prioritization Techniques can be scaled under Three categories: nominal scale, ordinal scale, and ratio scale[10].

III. METHODOLOGY

Our study presents a machine learning–based approach for task prioritization by combining structured features (Business Value, Urgency, Stakeholder Priority, Complexity, Effort Estimation) with unstructured task descriptions. The methodology begins with critical steps such as dataset collection, transformation, and preprocessing. Task ranks are normalized per user using min-max scaling to ensure fair user comparison. Feature engineering includes a Weighted Scoring Model (WSM) with predefined positive weights and semantic embeddings of task descriptions using RoBERTa. To address the high dimensionality of RoBERTa outputs, we compare performance with and without Principal Component Analysis (PCA), observing that using full embeddings improves model accuracy. A ColumnTransformer processes text, categorical, and numerical features in parallel, which are then passed to regression models; the Linear Regression model is considered a base model, Random Forest, and XGBoost. Hyperparameter tuning is performed using RandomizedSearchCV, and the model performance is evaluated using metrics such as R^2, MAE, and RMSE. Finally, SHAP and LIME are applied to explain model predictions and understand feature influence, supporting interpretability and transparency.

A. Dataset

Data acquisition is vital in the research process, particularly for individual researchers with limited resources. I obtained the dataset from the Mendeleev online platform https://github.com/RELabUU/revv-light. It consist of Total 1546 samples in which 1191 samples are considered as an Training samples and remaining 355 samples are taken as test samples. Agile software development uses the INVEST criteria for user stories, which should be Independent, Negotiable, Valuable, Estimable, Small, and Testable.

A typical user story follows this format:

As a *¡type of user¿*,
I want *¡to perform some task¿*,
So that *¡I can achieve some value/goal/benefit¿*.

TABLE I: EXAMPLES OF USER STORIES FROM THE MENDELEEV DATASET

User Story
As a UI designer, I want to move on to round 2 of Homepage edits, so that I can get approvals from leadership.
As a UI designer, I want to move on to round 3 of the Help page edits, so that I can get approvals from leadership.
As a Developer, I want to be able to log better, so that I can troubleshoot issues with particular submissions and functions.

B. Attribute Scaling Framework

Each task in the dataset was assessed using a five-point scale across five core Agile estimation attributes: Business Value, Urgency, Stakeholder Priority, Complexity, and Effort Estimation. The scale ranges from 1 (very low) to 5 (very high), capturing the relative significance or complexity of each attribute. Table II summarizes the interpretation of each rating value.

TABLE II: SCALE MEANING FOR ATTRIBUTE RATINGS

Score	Interpretation
1	Very Low – Minimal impact, urgency, stakeholder concern, or effort required.
2	Low – Slight relevance; unlikely to affect outcomes significantly.
3	Medium – Average or typical importance or complexity.
4	High – Important, visible to stakeholders, or requires focused effort.
5	Very High – Critical, time-sensitive, complex, or resource-intensive.

TABLE III: ASSIGNED ATTRIBUTE VALUES AND RATIONALE FOR TASK US01T2

Attribute	Value	Rationale
Business Value	4	Updating FABS records is vital for maintaining financial accuracy and compliance, contributing significantly to organizational objectives.
Urgency	4	The timeliness of updates affects reporting schedules and regulatory submissions, making the task time-sensitive.
Stakeholder Priority	4	Financial and compliance stakeholders rely heavily on accurate data, leading to high priority from their perspective.
Complexity	3	The task involves moderate technical effort such as interfacing with existing systems and handling data formats, without involving deep algorithmic work.
Effort Estimation	2	Assuming existing infrastructure, the development effort is relatively low—likely limited to implementing or adjusting a data reception interface.

C. Task Attribute Estimation

Without Domain experts or product stakeholders, the assignment of key attributes of Business Value, Urgency, Stakeholder priority, Complexity, and Effort estimation was performed using OpenAI's ChatGPT.This method simulated expert-level judgment in Agile software estimation, providing a consistent, interpretable, and replicable approach to task prioritization.

Let us consider one sample from the Dataset:

Task ID: US01T2

Task Description: "Implement functionality to receive updates to FABS records."

ChatGPT Prompt

You are a domain expert in Agile software development. Based on the following task description, assign values from 1 (lowest) to 5 (highest) for the following attributes: **Business Value**, **Urgency**, **Stakeholder Priority**, **Complexity**, and **Effort Estimation**. Provide a brief rationale for each score.

Task: "Implement functionality to receive updates to FABS records."

Fig. 3: Prompt shown to ChatGPT for task attribute estimation.

It is essential to note that these estimations should not replace the expert validation. The values are based on the general industry logic and should be reviewed and refined by domain experts when inputs are available. An AI-assisted method provides a scalable and step-by-step way to assign attribute values.

D. Prioritization Technique for Rank

Multi-criteria decision-making (MCDM) prioritizes tasks based on various factors to allocate resources effectively. The weighted Score model is a widely used prioritization technique chosen for the priority score. It is a structured and quantitative approach that allows decision-making to rank the alternatives by combining the scores across multiple criteria.

In this study, the normalized Weighted Scoring Model is applied to prioritize using the following criteria:

$$\text{Priority Score} = 0.5{\cdot}\text{BV}+0.4{\cdot}\text{U}+0.3{\cdot}\text{SP}+0.2{\cdot}\text{C}+0.1{\cdot}\text{E} \quad (1)$$

where:

- **BV (Business Value)**: Represents the anticipated impact or benefit of the item.
- **U (Urgency)**: Indicates how quickly the item must be addressed.
- **SP (Stakeholder Priority)**: Reflects the importance assigned by relevant stakeholders.
- **C (Complexity)**: Denotes the estimated technical or operational difficulty.
- **E (Effort Estimation)**: Refers to the resources (e.g., time, cost, personnel) required to implement the item.

The weights assigned to each criterion are normalized to sum to 1.0, ensuring a balanced prioritization strategy. A higher Priority Score indicates greater overall importance, integrating both strategic value and implementation feasibility

E. Distribution of Feature Importance

The distribution of feature importance shows how much each feature contributes to the model's predictions. It helps in identifying which features play the most significant role and which ones have minimal influence on the output.

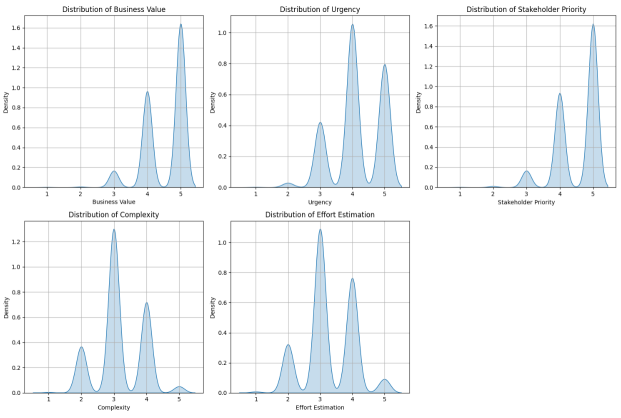


Fig. 4: Distribution of Feature Importance

As shown in Figure 4, the most important features are identified, and their impact on the model's decisions is visualized. The plot helps in determining whether the model is relying heavily on a few features or if the importance is spread across multiple features.

F. Machine Learning Models for Predicting Task Priority

To evaluate the predictability and consistency of the task scoring system, a set of machine learning models was implemented to forecast overall task priority; since the target variable is a continuous ranking, regression-based models are used. The dataset has a labeled supervised regression model, such as linear regression, which is considered a baseline model, a tree-based model as a random forest, and XGBoost as an ensemble-based approach to capture the non-linear relationship. The dataset was divided into 80/20 train-test split, and each model was trained using standard machine learning libraries such as sci-kit-learn.

G. Preprocessing and Tokenization

The raw task descriptions in the dataset were subjected to minimal preprocessing to retain the richness of the semantic relationship. This process includes removing the whitespaces, converting the text to lowercase, and preserving the punctuation marks and the stop words. Tokenization was performed with the help of the RoBERTa tokenizer. Each task was padded with a fixed length of 32 tokens, ensuring the input sizes were consistent with the model. RoBERTa tokenizer uses Byte Pair Encoding (BPE). Initially, embeddings were extracted using the BERT-based model; RoBERTa consistently outperformed BERT due to its training techniques and masking strategies. The custom transformer class based on the RoBERTa-base model was used to extract the embeddings from the [CLS] token of the model's final hidden layer. It works in evaluation mode without gradient calculations; this approach produces dense vector representations to encode the semantic meaning of each task description. These embeddings obtained from the RoBERTa embedder serve as a rich feature input for regression models to predict task prioritization.

H. Feature Infusion Technique

This work extracts diverse features from user story dataset, including semantic textual embeddings, categorical identifiers, and numerical estimations. To effectively transform and integrate these heterogeneous data types into a unified format for machine learning models, a ColumnTransformer is employed. The ColumnTransformer allows for parallel preprocessing of different columns by applying specific transformers to designated subsets of the data.

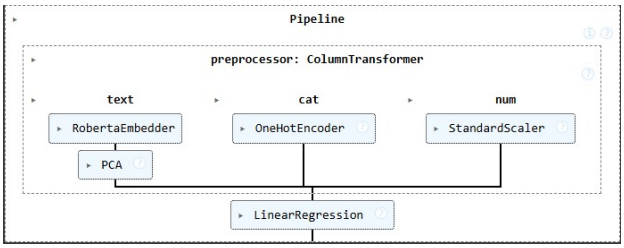


Fig. 5: Architecture of ColumnTransformer used for feature conversion.

1) ColumnTransformer-Based Feature Conversion: The dataset contains heterogeneous features that require distinct preprocessing. A `ColumnTransformer` is employed to apply appropriate transformations per feature type, ensuring consistent integration within the machine learning pipeline. The dataset consists of the following input features:

- **Textual Feature:** Task descriptions (Tasks column)
- **Categorical Features:** User ID,Task ID
- **Numerical Features:** Business Value, Urgency,Stakeholder Priority,Complexity,Effort Estimation

Textual Features (Task Descriptions): The `Tasks` column contains natural language descriptions (e.g., "Implement user login functionality"). Each task is encoded into a 768-dimensional vector using the [CLS] token from a pre-trained RoBERTa model, capturing its semantic meaning. To enhance efficiency, dimensionality reduction is applied using Principal Component Analysis (PCA).

Example:

$$\begin{aligned} \text{Original Embedding:} &\quad [0.02, -0.11, 0.45, \ldots, 0.12] \\ \text{After PCA (e.g., 50D):} &\quad [0.34, -0.12, 0.56, \ldots, 0.21] \end{aligned}$$

Categorical Features (User ID, Task ID): `User ID` and `Task ID` are nominal variables, encoded using one-hot encoding. For example, given:

$$\begin{aligned} \text{User ID:} &\quad \text{US01} \rightarrow [1,0,0], \\ &\quad \text{US02} \rightarrow [0,1,0], \\ &\quad \text{US03} \rightarrow [0,0,1] \end{aligned}$$

$$\begin{aligned} \text{Task Number:} &\quad \text{T1} \rightarrow [1,0], \\ &\quad \text{T2} \rightarrow [0,1] \end{aligned}$$

The combined encoding:

$$\begin{aligned} &\text{US01T1} \rightarrow [1,0,0,1,0], \\ &\text{US01T2} \rightarrow [1,0,0,0,1], \\ &\text{US02T1} \rightarrow [0,1,0,1,0] \end{aligned}$$

Numerical Features (Business Value, Urgency, Stakeholder Priority, Complexity, Effort Estimation): These features are integers ranging from 1 to 5 and represent task-specific estimates. `StandardScaler` is used to normalize them with zero mean and unit variance.

Example:

$$\begin{aligned} \text{Raw Values:} &\quad [4,3,5,2,1] \\ \text{Standardized:} &\quad [0.78, -0.45, 1.22, -1.10, -1.35] \end{aligned}$$

2) Summary: The combined use of RoBERTa embeddings, PCA, one-hot encoding, and feature scaling within a ColumnTransformer enables efficient preprocessing of mixed data types, ensuring compatibility with machine learning models and maintaining pipeline consistency.

I. Model Training

1) Train-Test Spilt Strategy: To evaluate model performance in machine learning, a standard train-test splitting strategy is inappropriate for this process. This analysis compares the train-test split strategy with GroupShuffle split to determine which performs better based on given criteria.

2) Standard Train-Test Spilt: The train test strategy, which splits the dataset in a random partition into training and test based on individual samples, does not provide grouping dependencies. In the dataset, each task is associated with a specific User ID, and splitting the tasks into random groups by applying this strategy leads to data leakage. Models may learn particular patterns during the training phase, and in the testing phase, it is not an advantage; it leads to a drop in performance estimation.

3) GroupShuffle Spilt: To overcome the problem of data leakage, I deploy the GroupShuffle split strategy from the scikit learn. It makes sure that all the tasks within the user story belong either to the train or test set, so that there is no random split across the train and test set, also the model will generalize well to the Unseen data. Also, ensure that the data doesn't overlap with each other.

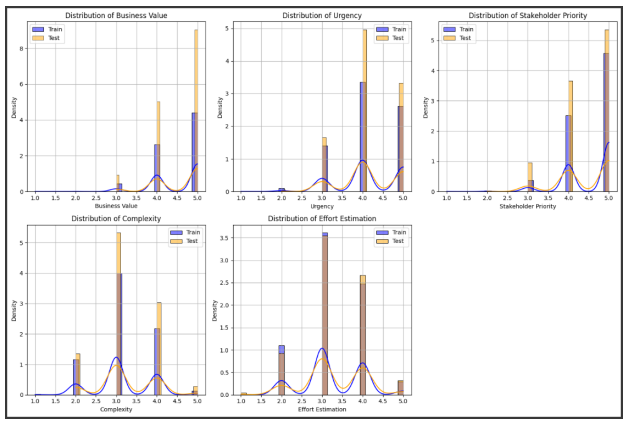


Fig. 6: Train-test split using GroupShuffleSplit method

J. Importance of Rank Normalization and Inverse Transformation

During the model training, rank normalization was applied to the target variable rank on the user stories because of the variation in the ranking scale across different users. Each user's stories have different numbers of tasks using different ranking scales (e.g., 1–5, 1–10, or 1–20), and model learning the different ranks may introduce biases and reduce the model's ability to generalize. To ensure it performs well, min-max normalization was applied to each group using the formula:

$$\text{Normalized Rank}_i = \frac{\text{Rank}_i - \min(\text{Rank}_u)}{\max(\text{Rank}_u) - \min(\text{Rank}_u)} \tag{2}$$

where Rank_i is the original rank of the i^{th} task associated with user u, and $\min(\text{Rank}_u)$, $\max(\text{Rank}_u)$ denote the minimum and maximum rank values for that user.

This normalization process enables the target variable rank to learn the intra-user task prioritization patterns with absolute values by maintaining the model stability. It also prevents the model from out-of-bond predictions during inference.

After the rank normalization, inverse rank normalization is applied to convert these predictions back to the original user-specific scale to obtain interpretability. The formula is given by:

$$\text{Predicted Rank}_i = \text{Normalized Prediction}_i \times (\max(\text{Rank}_u) - \min(\text{Rank}_u)) + \min(\text{Rank}_u) \tag{3}$$

Incorporating both normalization and inverse transformations during the model training process maintains consistent learning behaviour across the user's original ranking range.

K. Hperparameter Tunning

To determine the model's accuracy, cross-validation is applied to the training and test data across the various subsets. When you use the algorithm from the sklearn library, specific parameters need to be processed. Machine Learning models are composed of two parameters: Hyperparameters are all the parameters users can arbitrarily set before starting the Training. Model parameters are instead learned during the model training. Model tuning is an optimization problem that requires finding the right combination of values. There are several types of hyperparameter optimization: Manual, Grid search, Random search, and Bayesian model-based optimization. Setting the parameters manually is tedious when running a model, and the consumption time is also very high. Advanced techniques such as GridSearchCV and RandomizedSearchCV The package of GridSearchCV can be imported by trying each combination in a Grid manner, which is applicable for smaller sets of data but is Exhausting and computationally expensive. To overcome this, RandomizedSearchCV can be used for Larger datasets. Choosing the variables by defining the number of iterations involves a random combination based on the argument. Both are expensive processes in the machine learning pipeline in the model-building part.

TABLE IV: HYPERPARAMETER SETTINGS FOR LINEAR REGRESSION, RANDOM FOREST, AND XGBOOST

Linear Regression	Random Forest	XGBoost
No hyperparameters	n_estimators: 200 max_depth: 18 min_samples_split: 2 min_samples_leaf: 2 max_features: sqrt	n_estimators: 100 max_depth: 3 learning_rate: 0.01 subsample: 0.8 colsample_bytree: 0.8

L. Performance Evaluation Metrics

In regression tasks, various evaluation metrics are used to assess the performance of the model. The four most commonly used metrics are the Coefficient of Determination (R^2), Mean Squared Error (MSE), Mean Absolute Error (MAE), and Root Mean Squared Error (RMSE).

1) Coefficient of Determination (R^2): The Coefficient of Determination, denoted as R^2, measures the proportion of the variance in the dependent variable that is predictable from the independent variables. It is calculated as:

$$R^2 = 1 - \frac{\sum(y_{\text{true}} - y_{\text{pred}})^2}{\sum(y_{\text{true}} - \bar{y}_{\text{true}})^2}$$

Where: - y_{true} is the true value, - y_{pred} is the predicted value, - $\bar{y}_{\text{true}}$ is the mean of the true values.

A higher R^2 indicates that the model is better at explaining the variance in the data.

2) Mean Squared Error (MSE): The Mean Squared Error (MSE) is a commonly used metric that measures the average squared difference between the true values and the predicted values. It is computed as:

$$\text{MSE} = \frac{1}{n}\sum(y_{\text{true}} - y_{\text{pred}})^2$$

Where: - n is the number of data points, - y_{true} is the true value, - y_{pred} is the predicted value.

The MSE penalizes larger errors more heavily due to the squaring term, making it sensitive to outliers.

3) Mean Absolute Error (MAE): The Mean Absolute Error (MAE) measures the average absolute difference between the true values and the predicted values. It is calculated as:

$$\text{MAE} = \frac{1}{n}\sum|y_{\text{true}} - y_{\text{pred}}|$$

Where: - n is the number of data points, - y_{true} is the true value, - y_{pred} is the predicted value.

Unlike MSE, the MAE does not square the differences, making it less sensitive to large errors and providing a more intuitive measure of average prediction error.

4) Root Mean Squared Error (RMSE): The Root Mean Squared Error (RMSE) is the square root of the Mean Squared Error (MSE). It measures the average magnitude of the error in the same units as the original data, making it more interpretable than MSE. It is calculated as:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum(y_{\text{true}} - y_{\text{pred}})^2}$$

Where: - n is the number of data points, - y_{true} is the true value, - y_{pred} is the predicted value.

The RMSE is sensitive to large errors, similar to MSE, but is in the same unit as the target variable, making it easier to interpret

M. Post-Processing

A sequential tie-breaking strategy was deployed as a post-processing step to address duplicated predicted ranks during the evaluation phase. This method was applied at the user level to ensure a strict total ordering of tasks in each user story. Specifically, within the reach user group, functions with the identical predicted ranks were secondarily sorted based on the predefined hierarchy of domain-relevant features: Business value, Urgency, Stakeholder Priority, Complexity, and Effort estimation, all in descending order. This hierarchical sorting reflects the relative significance of these attributes in practical task prioritization scenarios. Following the multi-criteria sorting, unique adjusted ranks were assigned sequentially(from 1 to n)to the tasks within each group. This approach ensures the final predicted rankings are unique and consistent, provides reliable performance evaluation, and maintains alignment with real-world task priorities.

N. Explainable AI

O. Visualization using Explainable AI

Machine Learning exhibits remarkable performance across a variety of tasks. However, some state-of-the-art models lack transparency, trustworthiness, and explainability, often functioning as ”black boxes” that are difficult to understand. In recent years, the field of AI has introduced Explainable AI (XAI), which enables humans to comprehend better how models interpret results through methods like LIME (Local Interpretable Model-Agnostic Explanations) and SHAP (Shapley Additive Explanations).

XAI taxonomy considers transparency a key aspect, allowing both the models and their outputs to be understood by others. Interpretability refers to the ability to generate results that are comprehensible to humans. At the same time, explainability pertains to the interaction between humans and AI, ensuring that results are accurate and understandable.

During implementation, XAI has been applied to Machine Learning models such as Random Forest, Linear Regression, and XGBoost to analyze feature importance related to Business Value, Urgency, Stakeholder Priority, Complexity, and Effort Estimation.The results are presented in the form of SHAP plots.

P. Challenges

The greatest challenge is the availability of limited samples of quality datasets. Although the dataset consist of the number of samples per user, it was constrained, and the diversity of task types was relatively narrow. This posed a significant limitation for training machine learning models that generalize well across users. The issue became even more pronounced due to the inclusion of RoBERTa-based embeddings, where each task description was transformed into a 768-dimensional vector. While these embeddings capture rich semantic information, they significantly increase the feature dimensionality, making the model prone to overfitting and losing more data, particularly in algorithms sensitive to the curse of dimensionality like Random Forest and XGBoost. The target variable rank, often varying across users, made obtaining a consistent target variable difficult. These limitations lead to a careful balance between model complexity and generalizability. As a result, linear regression was used as a base model, whilst ensemble models like XGBoost and Random Forest were used with regularization and interpretability methods to control complexity. To mitigate data scarcity, several strategies were implemented: group-wise train-test splits ensured user-level segregation, rank normalization per user helped standardize targets and sequential tie-breaking was applied to refine predictions post-inference. Finally, SHAP and LIME explainability techniques were incorporated to validate and interpret the model behavior. These enhanced the results' trustworthiness and helped learn meaningful patterns rather than noise.

IV. RESULTS

A. Performance Evaluation of Models

Based on the performance metrics shown in Table V and Table VII, it is evident that models trained and tested without

applying the PCA consistently outperform those using PCA. For instance, in Table III (without PCA), Linear Regression achieves a near-perfect R^2 of 1.000 on the training set and 0.7576 on the test set. At the same time, XGBoost and Random Forest also demonstrate relatively strong generalization with lower RMSE and MAE values. In contrast, Table IV shows a decline in performance for all models after PCA is applied, with a noticeable drop in R^2 scores and increased error metrics (MSE, MAE, RMSE) across both training and test datasets. The R^2 for Linear Regression drops from 0.7576 to 0.523 on the test set, and RMSE increases from 3.7816 to 3.505. This degradation in performance suggests that PCA reduces dimensionality, but it may lead to the loss of essential features and variance critical to prediction accuracy. Therefore, considering these results, I have decided to proceed without using PCA to preserve the richness of the original feature set and maintain model performance.

TABLE V: PERFORMANCE EVALUATION FOR THE LINEAR REGRESSION, RANDOM FOREST, AND XGBOOST (TRAIN AND TEST SETS WITHOUT PCA)

Linear Regression				
Dataset	**R^2**	**MSE**	**MAE**	**RMSE**
Train	1.0000	2.1721	3.0365	4.6605
Test	0.7576	14.3006	2.6417	3.7816
Random Forest				
Dataset	**R^2**	**MSE**	**MAE**	**RMSE**
Train	0.9442	3.8050	1.3541	1.9507
Test	0.5809	24.7296	3.6347	4.9729
XGBoost				
Dataset	**R^2**	**MSE**	**MAE**	**RMSE**
Train	0.7815	14.8908	2.6917	3.8589
Test	0.7260	16.1647	2.9297	4.0205

TABLE VI: PERFORMANCE EVALUATION FOR LINEAR REGRESSION, RANDOM FOREST, AND XGBOOST (TRAIN AND TEST SETS WITHOUT TIE-BREAKING STRATEGY APPLIED)

Linear Regression				
Dataset	**R^2**	**MSE**	**MAE**	**RMSE**
Train	0.9999	8.0612	0.0058	0.0089
Test	0.5266	27.9329	3.6471	5.2852
Random Forest				
Dataset	**R^2**	**MSE**	**MAE**	**RMSE**
Train	0.7170	19.2797	3.0516	4.3908
Test	0.6297	21.8506	3.4056	4.6745
XGBoost				
Dataset	**R^2**	**MSE**	**MAE**	**RMSE**
Train	0.7848	14.6616	2.6951	3.8290
Test	0.7422	15.2114	2.8233	3.9002

In Table VI presents the evaluation metrics—R^2 Score, Mean Squared Error (MSE), Mean Absolute Error (MAE), and Root Mean Squared Error (RMSE) for Linear Regression, Random Forest, and XGBoost models evaluated on both the training and test datasets before the application of the tie-breaking strategy. The Linear Regression model achieves an R^2 of 0.999 on the training set with negligible error metrics (MSE = 0.000, RMSE = 0.012), which indicates extreme overfitting. However, the model's test performance drops sharply (R^2 = 0.523), accompanied by a significant increase in error (RMSE = 5.305). This suggests that while the model fits the training data closely, it fails to generalize effectively on unseen data. The Random Forest model performs more consistently across training and test sets. With an R^2 of 0.719 on training and 0.634 on testing, the model avoids overfitting and maintains stable prediction capability. The test RMSE (4.646) is lower than Linear Regression, showing better prediction reliability, especially in scenarios with non-linear feature relationships. The XGBoost model shows the highest R^2 values on both train (0.785) and test (0.742) sets. It also reports the lowest test MSE (15.202) and RMSE (3.899), along with the most petite MAE (2.823) among the three models. These values highlight XGBoost's superior capability in learning complex interactions within the data while maintaining robust generalization.

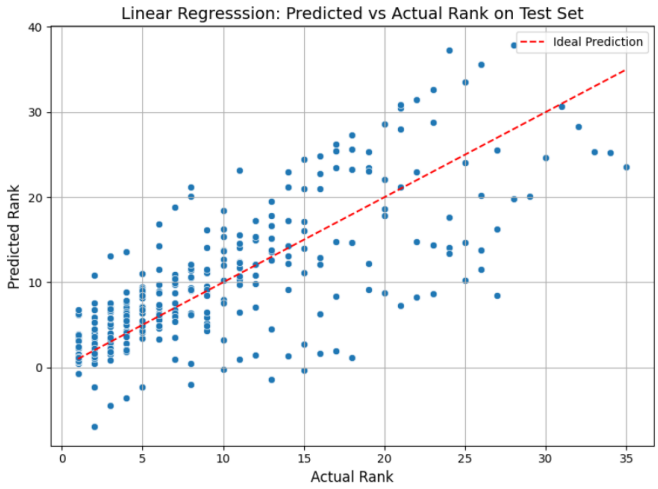
B. Visual Analysis of Actual and Predicted Ranks

Before applying the tie-breaking rule, the visual analysis of three models, Regression, Random Forest, and XGBoost, was evaluated using the scatter plots of actual vs predicted task ranks on the test set. In the Linear Regression model Figure 7a the predictions generally followed an upward trend, indicating that the model captures the overall pattern to some extent. However, there is a significant spread around the ideal prediction line, particularly at higher actual ranks. This indicates that the model struggles to predict tasks of higher priority accurately. In contrast, the Random Forest model Figure 7b demonstrates a dense clustering of predictions within a narrow band, especially between ranks 5 and 15. This saturation effect suggests that the model has difficulty distinguishing between higher-priority tasks, often assigning multiple tasks similar to ranks. The XGBoost model Figure 7c performs comparatively better, displaying more distribution patterns across the entire rank. Although it still unpredictable in the upper range, XGBoost demonstrates improved differentiation between mid and higher-rank tasks compared to the other models. Despite this improvement, all three models exhibit prediction ties and overlapping ranks. This makes it challenging to clear task order. Hence, a tie-breaking rule becomes essential to resolve these ambiguities and ensure more accurate downstream prioritization tasks.

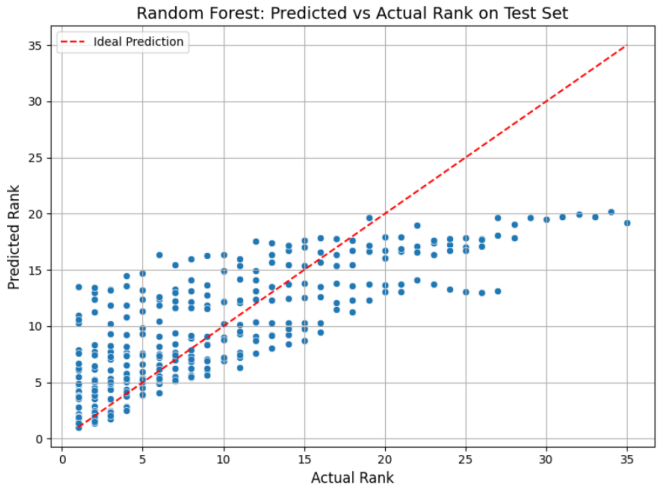
TABLE VII: PERFORMANCE EVALUATION FOR LINEAR REGRESSION, RANDOM FOREST, AND XGBOOST (TRAIN AND TEST SETS WITH TIE-BREAKING STRATEGY APPLIED)

Linear Regression				
Dataset	**R^2**	**MSE**	**MAE**	**RMSE**
Train	0.9999	0.0002	0.0099	0.0154
Test	0.8731	7.4873	1.5437	2.7363
Random Forest				
Dataset	**R^2**	**MSE**	**MAE**	**RMSE**
Train	0.7191	19.1388	3.0473	4.3747
Test	0.9141	5.0704	1.0141	2.2518
XGBoost				
Dataset	**R^2**	**MSE**	**MAE**	**RMSE**
Train	0.7849	14.6531	2.6950	3.8279
Test	0.9769	1.3634	0.5183	1.1676

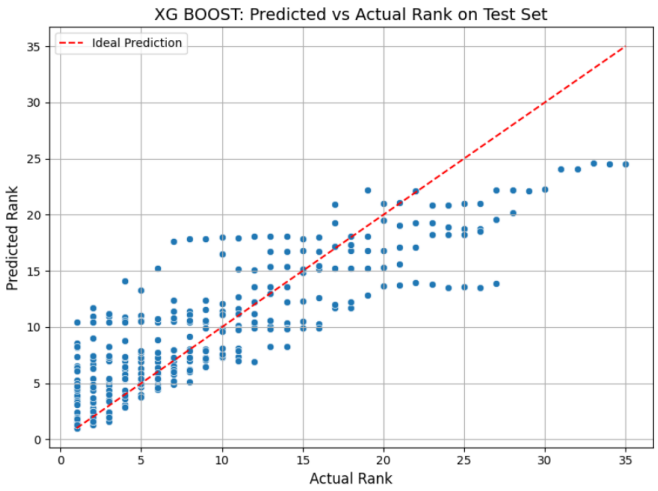
In Table VII, the Linear Regression model shows an improvement in generalization after applying the tie-breaking strategy, where the R^2 score on the test set increases from 0.523 to 0.8741, and RMSE decreases significantly from 5.305 to 2.726. This shows that tie-breaking helped the model to



(a) Actual vs Predicted Ranks (Linear Regression Model)



(b) Actual vs Predicted Ranks (Random Forest Model)



(c) Actual vs Predicted Ranks (XGBoost Model)

Fig. 7: Actual vs Predicted Ranks for All Models Before Applying the Tie-Breaking Strategy

capture the ranking distribution, reducing prediction variance. The random Forest model also benefits from improved R^2 from 0.943 and RMSE dropping to 1.1832, Reflecting the enhanced prediction accuracy and stability. The XGBoost model, which already performed best before tie-breaking, shows further gains with the test R^2 of 0.9763 and lower RMSE of 1.182 among all models. Its MAE also reduces to 0.523, suggesting exact rank prediction with minimal deviation.This enhanced performance across all three models confirms the effectiveness of the tie-breaking strategy in real-world task prioritization.

V. Visual Analysis of Actual and Predicted Ranks After Applying the Tie-Breaking Strategy

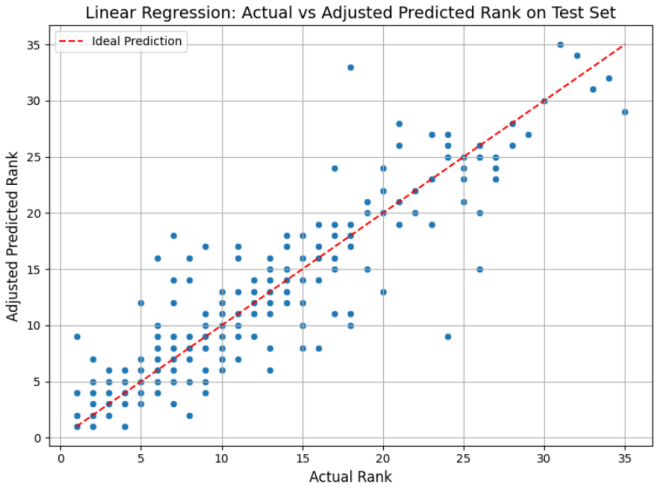
After implementing the tie-breaking strategy, we visually compared the actual ranks, and the adjusted predicted ranks for three models: Linear Regression, Random Forest, and XGBoost. This step aimed to resolve any ties and identical predicted ranks. The results are shown in Figure 8, how closely the adjusted predictions align with the ideal prediction line (depicted in red). The Linear Regression model Figure 8a shows a well-distributed spread of predictions with improved alignment toward the diagonal, especially compared to its earlier output before tie-breaking. This suggests the model more consistently captures the underlying rank structure, although minor dispersion still exists for higher-ranked tasks. The Random Forest model Figure 8b demonstrates a more refined distribution after tie-breaking, particularly in the mid-range ranks. While a few points still fall from the ideal line, the previously observed saturation between ranks 5 and 15 has been largely resolved. Finally, the XGBoost model Figure 8c exhibits the most substantial alignment among all three models, with most points closely hugging the ideal line. The adjusted predictions better represent a continuous and ordered rank structure, highlighting XGBoost's ability to benefit most from the tie-breaking adjustment. Applying the tie-breaking strategy resolves the repetition of rank by ensuring precise prioritization and clearer visual alignment to the ideal rank line.

1) Inference: To enhance the validity of our inference, we recognized that the limited number of samples in our dataset was inadequate for an effective validation set. To address this challenge, we generated an additional 30 samples using ChatGPT by crafting targeted prompts. This approach not only strengthens our dataset but also boosts the reliability of our findings.

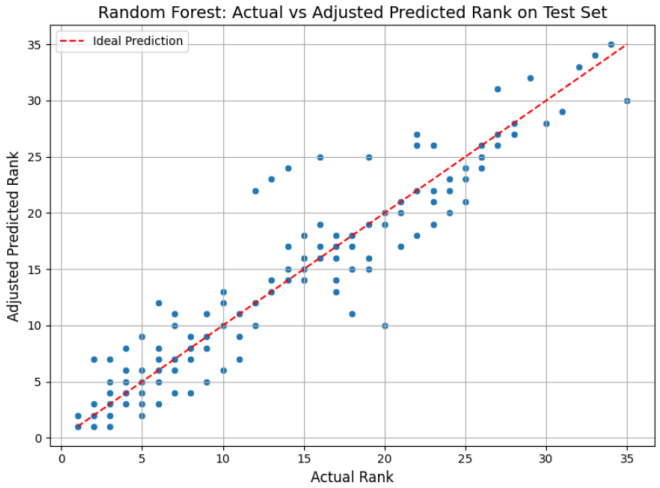
We have loaded the trained models and the preprocessed data saved in .pickle files for inference. We evaluated the performance of three models on 30 unseen samples to assess their generalization capabilities performance evaluation as follows metrics:

TABLE VIII: Inference Performance Metrics of Regression Models

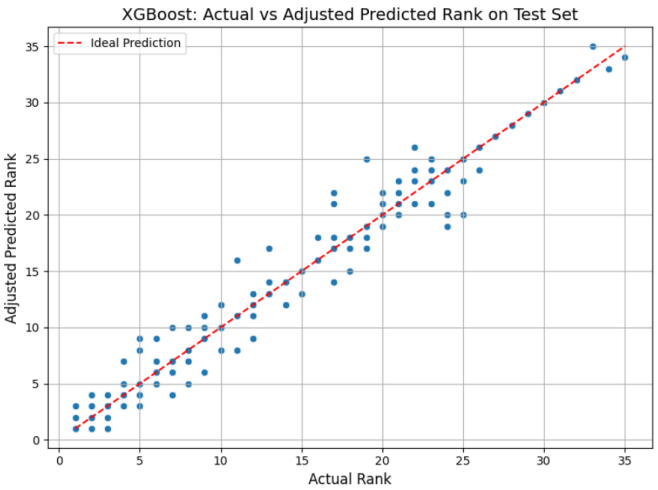
Model	R^2	MSE	MAE	RMSE
Linear Regression	0.5952	4.3680	1.5840	2.0900
Random Forest	0.3525	6.9867	2.0480	2.6432
XGBoost	0.5848	4.4800	1.5947	2.1166



(a) Actual vs Predicted Ranks (Linear Regression Model)



(b) Actual vs Predicted Ranks (Random Forest Model)



(c) Actual vs Predicted Ranks (XGBoost Model)

Fig. 8: Actual vs Adjusted Predicted Ranks for All Models After Applying the Tie-Breaking Strategy

ChatGPT Prompt to Generate User Stories

Role: You are a domain expert in Agile software development. Your task is to generate realistic and professional software user stories for a system in development.

Task: label=0)

1) Generate 30 unique user stories in the format: `As a [user role], I want to [action], so that [benefit].`
2) For each story, assign a task count ranging from 1 to 15.

Instructions: label=Step 0:, leftmargin=*, nosep

1) Use domain-relevant roles such as: developer, data analyst, UI designer, QA engineer, customer, product owner, etc.
2) Write a clear action that describes a specific technical or functional goal (e.g., create API endpoint, implement logging, view report).
3) Clearly state a benefit that reflects the purpose or business value (e.g., monitor performance, enable integration, improve fault recovery).
4) Avoid vague or AI-sounding phrases. Keep it realistic and enterprise-appropriate.

Output Format: Return user stories in the format below. Do not number them.

```
As a [user role], I want to
[action], so that [benefit].
Tasks: [1--15]
```

Notes: nosep

- Keep the tone consistent and practical.
- Ensure a mix of technical depth and business value.
- Do not output explanations—only the final user stories.

Fig. 9: Prompt shown to ChatGPT for generating user stories and task estimations.

The performance of the regression models was first evaluated on the test set, as presented in Table VI. This test set consists of data not seen during training but still originates from the same distribution as the training set. XGBoost showed the highest predictive performance on the test set, achieving an R^2 score of 0.9763 and an RMSE of 1.182, followed by Random Forest and Linear Regression. Although Linear Regression had a lower R^2 score, it demonstrated relatively stable and consistent predictions. These test results reflect how well the models have learned from the training data and how accurately they can generalize within a known data context.

To evaluate the readiness for real-world deployment, we assessed the inference performance using the results presented in Table VIII. This synthetic inference data was designed

to mimic realistic yet unseen scenarios, which tested the models' generalization capabilities. When evaluated on this synthetic dataset, all three models experienced a decline in performance. XGBoost Showed the strongest performance in inference, with an R^2 value of 0.6628 and an RMSE of 1.8776, confirming its generalizing capabilities. In contrast, Random Forest showed a sudden drop, with the R^2 from 0.943 to 0.4875, indicating that the model is overfitting. Although Linear Regression also experienced a drop in R^2, it slightly improved RMSE, suggesting more constant error margins. Overall, these results highlight XGBoost as the most reliable model for inference, particularly when faced with unfamiliar, real-world-like conditions.

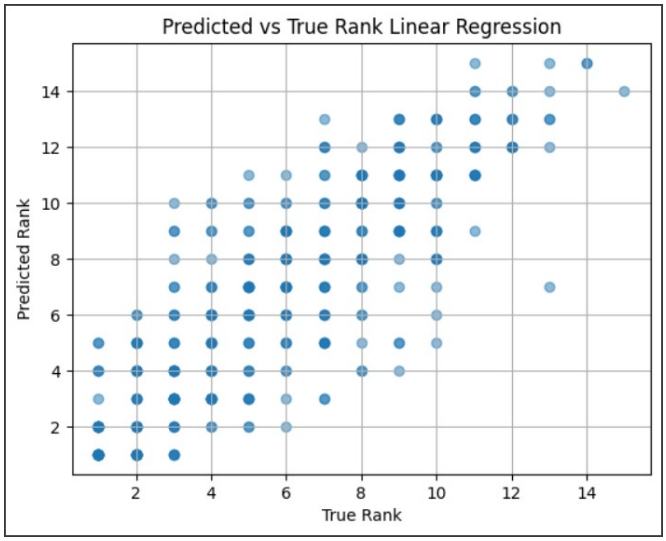
Figure 10 illustrates the predicted vs. True ranks during inference for each model. Linear Regression shows a fairly linear trend but with noticeable vertical dispersion, especially at higher ranks. Random Forest predictions are more scattered, indicating weaker generalization and less consistency in rank alignment. In contrast, XGBoost demonstrates the closest alignment to the diagonal (ideal) line, suggesting superior accuracy and stability in predicting ranks across the entire range. This visual analysis supports the earlier quantitative findings that XGBoost generalizes best during inference.

A. SHAP Analysis Plots

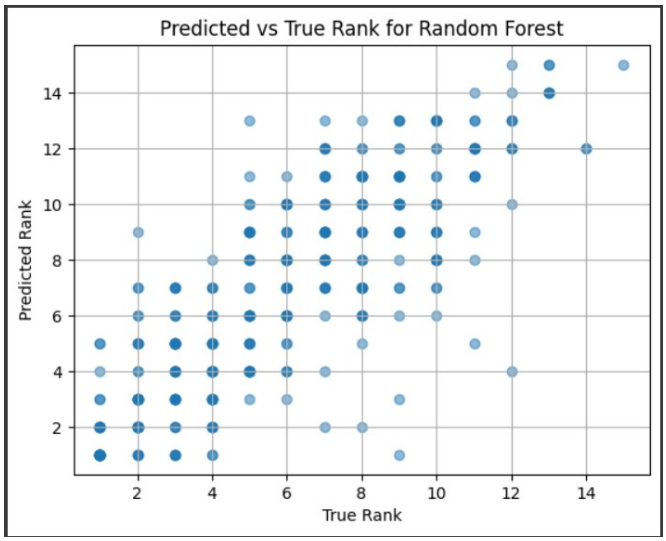
SHAP (Shapley Additive explanations) summary plots were generated for the Linear Regression, Random Forest, and XGBoost models using Roberta embeddings alongside categorical and numerical features to gain insights into how each model makes predictions. The SHAP plots reveal how each feature contributes to the predicted task rank. For the Linear Regression Model, the impact is distributed more smoothly across numerical features such as Urgency, Complexity, and Business Value, along with specific Roberta embedding dimensions like textfeature-588 and textfeature-85. This aligns with the nature of linear models, which tend to weigh multiple features moderately rather than focus sharply on a few. In contrast, the Random Forest Model exhibits more abrupt attributions, where a few features dominate the output. High-impact features include semantic embeddings textfeature-139, textfeature-566, and structured task properties such as Business Value, Urgency, Effort Estimation). This behavior reflects the decision-tree structure of Random Forests, which split on the most informative features. The XGBoost displays the most concentrated feature importance. Several embedding dimensions, such as textfeature-212, textfeature-742, and task-level descriptors, especially Business Value and Urgency, contribute significantly to the predictions. This aligns with XGBoost's gradient-boosting framework, which performs feature selection during optimization. The SHAP analysis confirms that all models effectively learn from semantic content via Roberta and structured task features, with each model showing its characteristic pattern of reliance and attribution.

B. LIME Analysis Plots

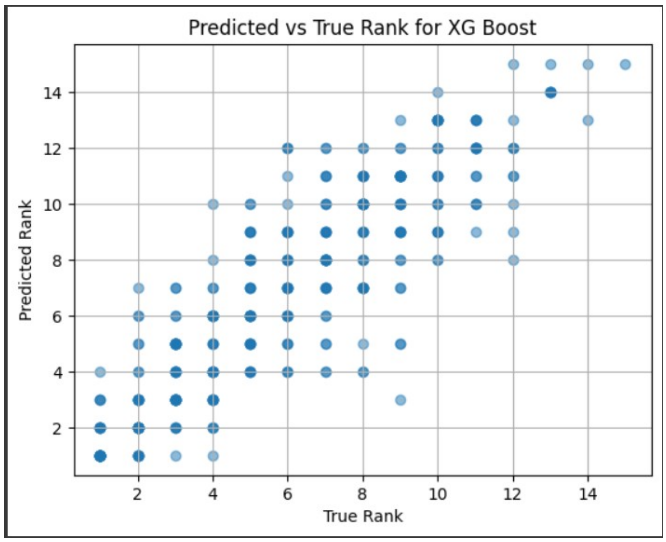
Figs. 14–16 illustrates the LIME explanations for individual task predictions generated by the Linear Regression, Random Forest, and XGBoost models, respectively. Each LIME



(a) Predicted vs True Rank (Linear Regression Model)



(b) Predicted vs True Rank (Random Forest Model)



(c) Predicted vs True Rank (XGBoost Model)

Fig. 10: Predicted vs True Rank During Inference is applied

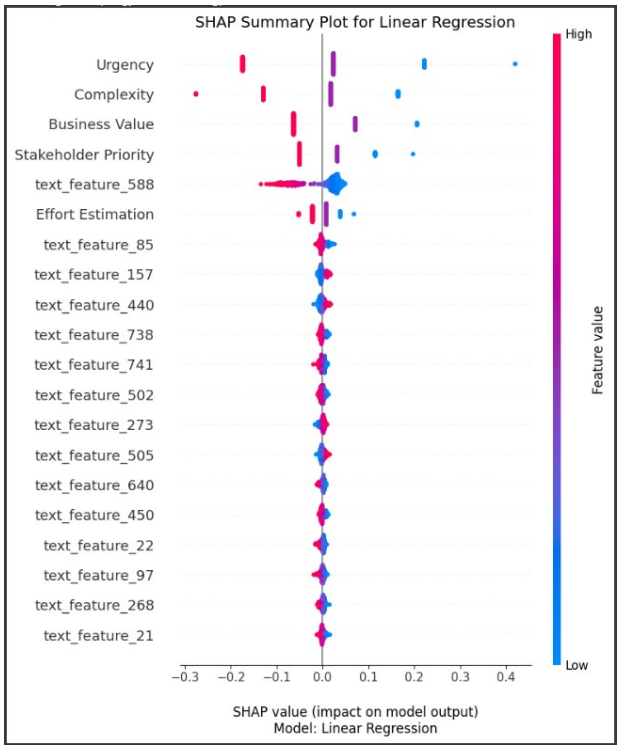


Fig. 11: SHAP Summary Plot for Linear Regression Model

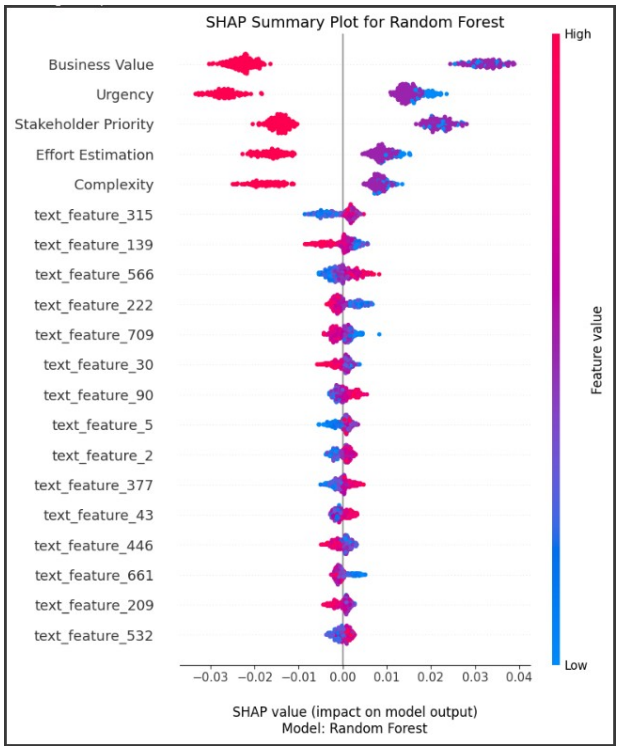


Fig. 12: SHAP Summary Plot for Random Forest Model

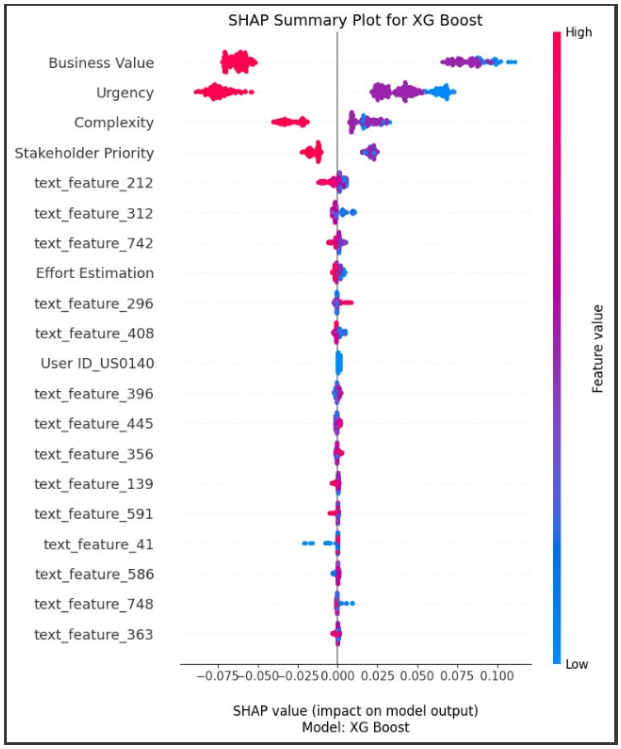


Fig. 13: SHAP Summary Plot for XGBoost Model

plot highlights the most influential features contributing to a model's prediction for a specific task instance. The bar chart on the left side of each figure displays the direction and magnitude of feature contributions. Features with orange bars positively influence the prediction, increasing the task priority score, while blue bars contribute negatively. The horizontal length of each bar indicates the relative strength of that feature's impact.

The actual feature values used in the prediction are shown on the right-hand side. For instance, in Fig. 14, features such as Urgency, Complexity, and Business Value contributed positively to the predicted score. In Fig. 16, the XGBoost model also highlights Business Value, Urgency, and Effect Estimation as key contributors, indicating consistency across models regarding important factors influencing task priority. Across all models, domain-driven features such as Business Value, Stakeholder Priority, and Effect Estimation appear repeatedly among the top-ranked variables. The XGBoost model demonstrates more sharply focused attributions, reflecting its capacity to capture complex, non-linear interactions between features. In the case of the linear Regression model, it distributes influence more evenly across multiple variables due to its linear nature.

These insights confirm that the machine learning models align well with Agile task prioritization logic and that LIME provides actionable transparency into model behavior.

VI. Discussion

This study leverages a real-world agile task dataset to evaluate the effectiveness of machine learning models in predicting the task priority. Among all the models, XGBoost delivered the highest performance evaluation metrics. The test also

Fig. 14: LIME summary plot for Linear Model

Fig. 15: LIME summary plot for the Random Forest Model

Fig. 16: LIME summary plot for the XGBoost model

confirmed its ability to generalize well. A weighted score logic is applied to aggregate the attributes into a single composite priority score. Two samples from the test data (User IDs: US0112 and US0113) were analyzed in detail to provide task-level insights. As shown in Table IX, XGBoost's predicted ranks were closely related to actual priorities. The linear regression model often under-performed on mid-range priorities, and Random Forest exhibits ranking inconsistencies; it shows that XGBoost excels in overall accuracy and maintains the domain logic when applied to real-world scenarios. This makes the agile environments support backlog prioritization and sprint planning processes.

To determine whether my models are overfitting by learned patterns by a machine learning algorithm, a synthetic dataset was constructed for validation. The dataset also deals with a limited number of samples that cannot be applied for validation. For User ID: US012, the predicted Task ranks were compared against the predefined set of actual ranks across the task set. Again, XGboost demonstrates the ability to effectively internalize the weighted scoring logic applied during the preprocessing as shown in Table X. Again, Linear Regression significantly varied in handling the nonlinear feature interactions. In contrast to the Random Forest, there is slight instability in the mid-ranked Tasks.XGboost offers superior performance in real-world and synthetic settings. Its predictions aligned more closely with the domain-derived actual ranks; it made flexible Agile planning tools aimed at automated task prioritization.

A. *Overview of the Model's Performance*

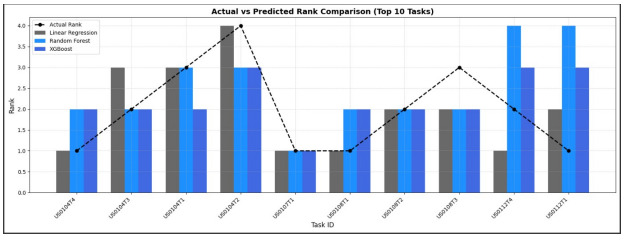


Fig. 17: Model's performance of Linear Regression, Random Forest, and XGBoost

Figure 17 compares the performance of three models, Linear Regression, Random Forest, and XGBoost, explicitly focusing on the top 10 tasks related to predicted rankings. The plot displays the rank predictions generated by each model as grouped bars, while a black dashed line with markers indicates the actual rankings (ground truth).

The visualization specifies that XGBoost predictions closely match the actual ranks, showing that it can capture the non-linear relationships and interactions among input features. Random Forest exhibits deviations in specific cases; it shows the variance introduced by its ensemble averaging mechanism. Linear Regression serves as a baseline model. It displays the

TABLE IX: COMPARISON OF ACTUAL AND PREDICTED TASK RANKS FOR SAMPLE USERS ON REAL WORLD DATASET

User ID	Task ID	Task Description	Actual Rank	Linear Regression	Random Forest	XGBoost
US0112	US0112T1	Track financial responsibility for each item	1	2	1	1
US0112	US0112T2	Run periodic report on storage consumed by financial contact	5	6	5	5
US0112	US0112T3	Specify remote replication policy for each collection	4	3	4	4
US0112	US0112T4	Elect to replicate remotely or not, and replicate beyond primary site	2	1	2	2
US0112	US0112T5	View authenticated, active users and anonymous users	8	8	8	8
US0112	US0112T6	Schedule message to users for notification of upcoming downtime	9	9	9	9
US0112	US0112T7	Display recent errors for easy comprehension	6	7	6	6
US0112	US0112T8	View dashboard statistics about collections and storage	3	4	3	3
US0112	US0112T9	View available versions of an object and restore a version	7	5	7	7
US0113	US0113T1	Add a date delimiter to a search string	1	1	1	1
US0113	US0113T2	Go to repository dashboard to manage members	3	3	2	3
US0113	US0113T3	Grant submit permissions by pasting in NetID list	2	2	3	2
US0113	US0113T4	See confirmation that DDR recognizes NetIDs	4	4	4	4

TABLE X: ACTUAL VS PREDICTED TASK RANKS BY ML MODELS ON SYNTHETIC DATASET (USER ID: US12)

User ID	Task ID	Task Description	Actual Rank	Linear Regression	Random Forest	XGBoost
US012	US012T1	Refactor the database schema	9	4	11	9
US012	US012T2	Improve the analytics module	8	10	5	7
US012	US012T3	Fix the user interface	6	8	6	6
US012	US012T4	Add the authentication module	12	12	10	12
US012	US012T5	Remove the login system	10	11	8	11
US012	US012T6	Create the error handler	2	3	1	1
US012	US012T7	Implement the API endpoint	11	9	12	10
US012	US012T8	Fix the authentication module	7	5	9	8
US012	US012T9	Remove the login system	5	7	3	5
US012	US012T10	Remove the analytics module	1	1	2	2
US012	US012T11	Optimize the user interface	4	6	4	3
US012	US012T12	Create the dashboard view	3	2	7	4

non-linear relationship at the highest level of discrepancy; this comparative analysis provides the superior performance of the gradient-boosted decision tree, XGBoost, as a solution for rank-based task prioritization in regression tasks.

VII. CONCLUSION

This research proposes a data-driven framework for task prioritization in Agile development by combining machine learning techniques with semantically enriched representations. This approach integrates ordinal scales to represent the Agile criteria such as Business Value, Urgency, Complexity, and Stakeholder Priority, enabling a structured and interpretable prioritization mechanism. Deep Learning, specifically the RoBERTa model, was employed to capture the semantic embeddings of the user stories' task descriptions. Among the evaluated models, XGboost handles the interactions of nonlinear relationship features as a superior performance aligning with the ground truth rankings. To ensure model transparency, such as SHAP and Lime were applied to contribute the Task rankings to model output. These explanations validate the trustworthiness. In real-world Agile Environments, teams often struggle with subjectivity in prioritization, large and rapidly growing backlogs, and misalignment in Business and technical goals. This study addresses the challenges by offering automation in task ranking, an explainable system that enhances decision consistency and incorporates structured criteria alongside a semantic understanding of user stories. For Future work, the framework can be integrated into the res time feedback from Agile teams, exploring the transformer-based ranking models end to end and evaluating the cross-domain generalization across the projects and organizations. It can also be Integrated with the help of Agile tool management like Jira, which could facilitate real-time decision support. This study highlights the effectiveness of interpretable machine learning for enhancing Agile task management.

REFERENCES

[1] M. A. Sami, Z. Rasheed, M. Waseem, Z. Zhang, T. Herda, and P. Abrahamsson, "Prioritizing software requirements using large language models," *arXiv preprint arXiv:2405.01564*, 2024.

[2] N. Tasneem, H. B. Zulzalil, and S. Hassan, "Enhancing agile software development: A systematic literature review of requirement prioritization and reprioritization techniques," *IEEE Access*, 2025.

[3] M. Mahboob, M. R. U. Ahmed, Z. Zia, M. S. Ali, and A. K. Ahmed, "Future of artificial intelligence in agile software development," *arXiv preprint arXiv:2408.00703*, 2024.

[4] B. Alsaadi and K. Saeedi, "Data-driven effort estimation techniques of agile user stories: A systematic literature review," *Artificial Intelligence Review*, vol. 55, no. 7, pp. 5485–5516, 2022.

[5] S. Sachdeva, A. Arya, P. Paygude, S. Chaudhary, and S. Idate, "Prioritizing user requirements for agile software development," in *2018 International Conference On Advances in Communication and Computing Technology (ICACCT)*, IEEE, 2018, pp. 495–498.

[6] G. S. Matharu, A. Mishra, H. Singh, and P. Upadhyay, "Empirical study of agile software development methodologies: A comparative analysis," *ACM SIGSOFT Software Engineering Notes*, vol. 40, no. 1, pp. 1–6, 2015.

[7] A. M. Radwan, M. A. Abdel-Fattah, and W. Mohamed, "Ai-driven prioritization techniques of requirements in agile methodologies: A systematic literature,"

[8] A. B. M. Moniruzzaman and S. A. Hossain, "Comparative study on agile software development methodologies," *CoRR*, vol. abs/1307.3356, 2013. arXiv: 1307.3356. [Online]. Available: http://arxiv.org/abs/1307.3356.

[9] V. Pendyala, "Evolution of integration, build, test, and release engineering into devops and to devsecops," in *Tools and techniques for software development in large organizations: emerging research and opportunities*, IGI Global, 2020, pp. 1–20.

[10] E. Alhenawi, S. Awawdeh, R. A. Khurma, M. García-Arenas, P. A. Castillo, and A. Hudaib, "Choosing a suitable requirement prioritization method: A survey," *arXiv preprint arXiv:2402.13149*, 2024.

[11] N. Saher, F. Baharom, and R. Romli, "A review of requirement prioritization techniques in agile software development," in *Knowledge management international conference (KMICe)*, 2018, pp. 25–27.

[12] N. A. TERIDI, Z. A. A. K. ADZHAR, N. M. RAHIM, *et al.*, "The approach using cumulative voting and spanning tree technique in implementing functional requirement prioritization: A case study of student's financial system development," *Journal of Theoretical and Applied Information Technology*, vol. 101, no. 3, 2023.

[13] L. Rojas, C. Olivares-Rodríguez, C. Alvarez, and P. G. Campos, "Ourrank: A software requirements prioritization method based on qualitative assessment and cost-benefit prediction," *IEEE Access*, vol. 10, pp. 131 772–131 787, 2022. DOI: 10.1109/ACCESS.2022.3230152.

[14] L. Karlsson, T. Thelin, B. Regnell, P. Berander, and C. Wohlin, "Pair-wise comparisons versus planning game partitioning—experiments on requirements prioritisation techniques," *Empirical Software Engineering*, vol. 12, pp. 3–33, 2007.

[15] R. Anwar and M. B. Bashir, "A systematic literature review of ai-based software requirements prioritization techniques," *IEEE Access*, vol. 11, pp. 143 815–143 860, 2023. DOI: 10.1109/ACCESS.2023.3343252.

[16] K. S. Ahmad, N. Ahmad, H. Tahir, and S. Khan, "Fuzzy$_m oscow$: A fuzzy based moscow method for the prioritization of software requirements," in *2017 International Conference on Intelligent Computing, Instrumentation and Control Technologies (ICICICT)*, 2017, pp. 433–437. DOI: 10.1109/ICICICT1.2017.8342602.

[17] M. Ramzan, M. A. Jaffar, and A. A. Shahid, "Value based intelligent requirement prioritization (virp): Expert driven fuzzy logic based prioritization technique," *International Journal Of Innovative Computing, Information And Control*, vol. 7, no. 3, pp. 1017–1038, 2011.

[18] P. Bajaj and V. Arora, "Multi-person decision-making for requirements prioritization using fuzzy ahp," *ACM SIGSOFT Software Engineering Notes*, vol. 38, no. 5, pp. 1–6, 2013.

[19] A. Fatima, A. Fernandes, D. Egan, and C. Luca, "Software requirements prioritisation using machine learning," 2023.

[20] N. R. Bollumpally, A. C. Evans, S. W. Gleave, A. R. Gromadzki, and G. Learmonth, "A machine learning approach to workflow prioritization," in *2019 Systems and Information Engineering Design Symposium (SIEDS)*, IEEE, 2019, pp. 1–5.

[21] P. Achimugu and A. Selamat, "A hybridized approach for prioritizing software requirements based on k-means and evolutionary algorithms," in *Computational intelligence applications in modeling and control*, Springer, 2014, pp. 73–93.

[22] P. Avesani, C. Bazzanella, A. Perini, and A. Susi, "Facing scalability issues in requirements prioritization with machine learning techniques," in *13th IEEE International Conference on Requirements Engineering (RE'05)*, 2005, pp. 297–305. DOI: 10.1109/RE.2005.30.

[23] P. Avesani, S. Ferrari, and A. Susi, "Case-based ranking for decision support systems," in *Case-Based Reasoning Research and Development: 5th International Conference on Case-Based Reasoning, ICCBR 2003 Trondheim, Norway, June 23–26, 2003 Proceedings 5*, Springer, 2003, pp. 35–49.

[24] J. T. de Souza, C. L. B. Maia, T. d. N. Ferreira, R. A. F. d. Carmo, and M. M. A. Brasil, "An ant colony optimization approach to the software release planning with dependent requirements," in *Search Based Software Engineering: Third International Symposium, SSBSE 2011, Szeged, Hungary, September 10-12, 2011. Proceedings 3*, Springer, 2011, pp. 142–157.

[25] H. Sheemar and G. Kour, "Enhancing user-stories prioritization process in agile environment," in *2017 International Conference on Innovations in Control, Communication and Information Systems (ICICCI)*, 2017, pp. 1–6. DOI: 10.1109/ICICCIS.2017.8660760.

[26] R. Tamrakar and M. Jørgensen, "Does the use of fibonacci numbers in planning poker affect effort estimates?" In *16th International Conference on Evaluation & Assessment in Software Engineering (EASE 2012)*, IET, 2012, pp. 228–232.