Sprint2Vec: a deep characterization of sprints in iterative software development

Morakot Choetkiertikul, Peerachai Banyongrakkul, Chaiyong Ragkhitwetsagul, Suppawong Tuarob, Hoa Khanh Dam, and Thanwadee Sunetnanta

Abstract—Iterative approaches like Agile Scrum are commonly adopted to enhance the software development process. However, challenges such as schedule and budget overruns still persist in many software projects. Several approaches employ machine learning techniques, particularly classification, to facilitate decision-making in iterative software development. Existing approaches often concentrate on characterizing a sprint to predict solely productivity. We introduce Sprint2Vec, which leverages three aspects of sprint information – sprint attributes, issue attributes, and the developers involved in a sprint, to comprehensively characterize it for predicting both productivity and quality outcomes of the sprints. Our approach combines traditional feature extraction techniques with automated deep learning-based unsupervised feature learning techniques. We utilize methods like Long Short-Term Memory (LSTM) to enhance our feature learning process. This enables us to learn features from unstructured data, such as textual descriptions of issues and sequences of developer activities. We conducted an evaluation of our approach on two regression tasks: predicting the deliverability (i.e., the amount of work delivered from a sprint) and quality of a sprint (i.e., the amount of delivered work that requires rework). The evaluation results on five well-known open-source projects (Apache, Atlassian, Jenkins, Spring, and Talendforge) demonstrate our approach's superior performance compared to baseline and alternative approaches.

Index Terms—Iterative software development, Deep learning, Mining software repositories

1 Introduction

In modern software development, iterative approaches such as Agile Scrum are commonly adopted by software developer teams to enhance the software development process and improve productivity [1]. Despite the adoption of iterative software development approaches, many software teams still face challenges such as schedule and budget overruns [2]. The dynamic nature of software development poses difficulties in project and risk management. These are crucial aspects of iterative development, which emphasizes rapid deployment and continuous delivery [3], [4]. Previous studies have indicated that while productivity may increase in iterative development, there is a risk of lower quality products [4], [5]. Rapid delivery can also contribute to technical, quality, and maintainability debts in software projects [6].

Artificial Intelligence (AI) and Machine Learning (ML) approaches have been widely explored to support iterative software development. These approaches have been applied to various tasks, such as predicting the amount of work delivered at the end of iterations (i.e., sprints) [7], [8], predicting the risk of delays in software development tasks and bug fixing time [9], [10], [11], and recommending a set of tasks in sprint planning [12]. These AI and ML techniques offer valuable insights and decision support to enhance the effectiveness and efficiency of iterative software development processes. These methods involve the process of finding a representation vector (i.e., features) of a given entity and utilizing it in the training of classification models. However, finding suitable features is a non-trivial task. The effectiveness of these approaches heavily relies on the quality of features obtained from the feature extraction process (i.e., feature engineering). In order to support a wide range of prediction tasks in iterative software development, it is crucial to have an approach that can automatically derive a comprehensive set of features representing iterations (i.e., sprints).

In this paper, we introduce Sprint2Vec, an approach that develops vector representations of sprints in Agile Scrum software development. Specifically, Sprint2Vec is an approach that derives a feature vector representation of an iteration (i.e., sprint). It combines features extracted from sprints, tasks, and developer teams' attributes with learned features from textual descriptions of tasks and developer teams' activities using automated feature learning based on deep learning techniques. Moreover, while existing works in the field primarily focus on predicting the amount of work that can be delivered at the end of iterations (i.e., productivity) [7], [13], it is equally important to consider the quality of the delivered work. Both productivity and quality outcomes in iterative development heavily depend on the performance of the software development team [5]. The features derived from Sprint2Vec have been evaluated in two predictive (regression) tasks, aiming to address these gaps. The evaluation results show that the predictive models trained using the features derived from Sprint2Vec have the capability to predict both the amount of work, represented by the number of resolved issues in an iteration, and assess the risk of quality impact by identifying reopened issues.

The main contribution of this paper is the proposal of

- *M. Choetkiertikul, P. Banyongrakkul, C. Ragkhitwetsagul (corresponding author), S. Tuarab, and T. Sunetnanta are with the Faculty of Information and Communication Technology, Mahidol University, Thailand E-mail: {morakot.cho, chaiyong.rag, suppawong.tua, thanwadee.sun}@mahidol.ac.th, peerachai.ban@student.mahidol.ac.th*
- *H.K. Dam is with the University of Wollongong, Australia. E-mail: hoa@uow.edu.au*

a novel approach that leverages both work tasks and team information using an automated feature learning approach combined with essential extracted features. This approach can be applied to the two predictive tasks, encompassing both qualitative and quantitative outcomes of sprints. To facilitate future research and reproducibility, we have published the dataset, scripts, and experimental results (including raw predictions) on a GitHub repository [14].

The remainder of this paper is organized as follows. Section 2 provides background and related work on the iterative software development approach, the challenges faced in sprint planning and prediction tasks, and automatic feature learning. We present the details of our proposed approach, Sprint2Vec, including the data sources, feature extraction techniques, and the deep learning-based feature learning process in Section 3. Section 4 describes the experimental setup, dataset used, and evaluation metrics. The results of the evaluation are presented and analyzed in this section. We then conclude our work in Section 5.

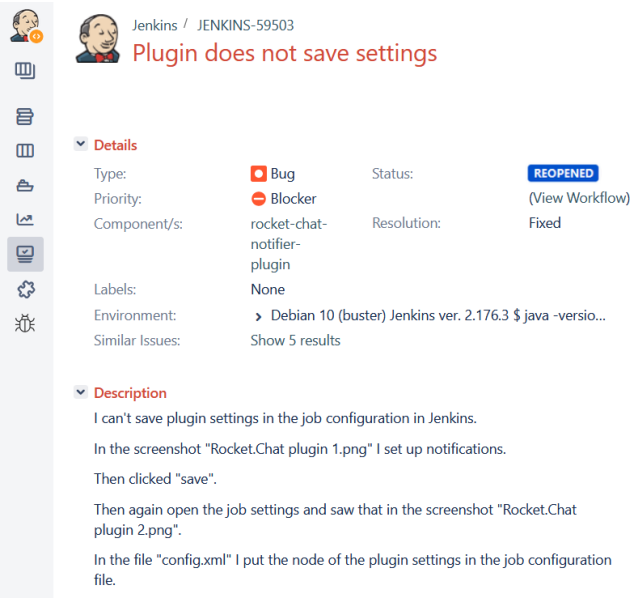


Fig. 1: An example of a reopened issue[2]

2 Background and Related Work

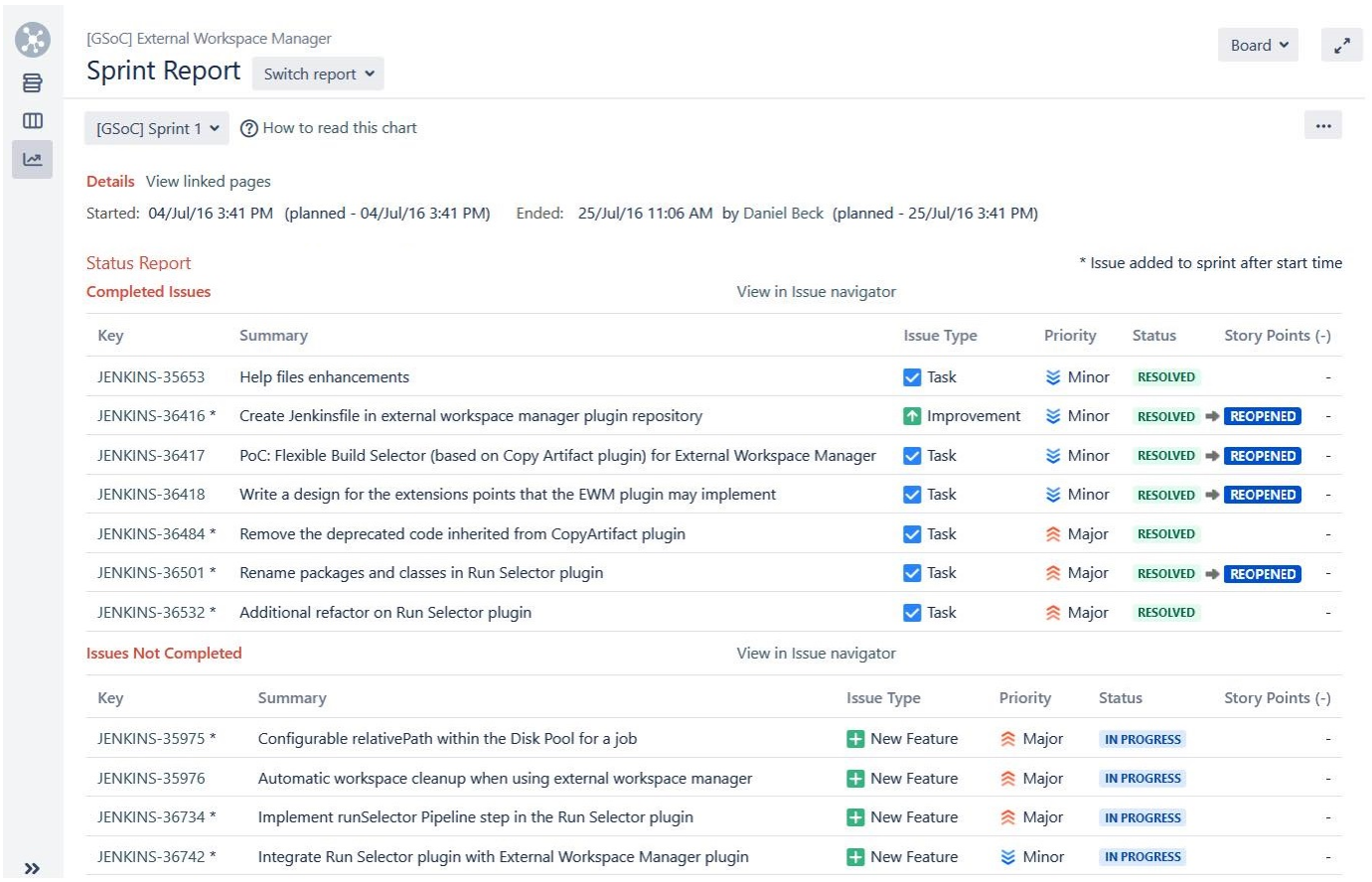
2.1 Iterative Software Development

In iterative software development, projects are executed in short time-boxed periods called iterations or sprints. These iterations allow for adaptability to changing requirements throughout the project's lifecycle. A typical sprint lasts for about two to four weeks and involves the design, development, testing, and delivery of a specific software component. Effective planning and monitoring are essential during each sprint to ensure the completion of tasks, known as user stories. The team's productivity is measured by the number of tasks completed at the end of a sprint. Software teams commonly utilize issue tracking systems like Jira to manage their tasks or user stories, which are represented as "issues" [15], [16]. Each issue goes through a lifecycle, starting from creation and progressing through various stages such as triaging and assignment to developers. At the end of its lifecycle, an issue is closed or resolved. It is important to note that after closing a sprint, previously completed issues can be reopened if they fail to meet the expected criteria. This can have a negative impact on the sprint's quality, requiring additional effort and resources to address these reopened issues [17], [18].

Figure 1 illustrates an example of an issue that has been reopened in the Jenkins project. The issue ID *JENKINS-59503* highlights a situation where a plugin delivered by the team fails to meet the expected requirements as it does not save settings. Despite an initial resolution attempt, the problem persists on certain systems, necessitating a reopening of the issue and additional work to address it in a future sprint. Another instance of an issue reopening with a negative impact on sprint quality is shown in Figure 2. The sprint report for *[GSoC] Sprint 1* displays a commitment to deliver four issues within the sprint. However, seven additional issues are added during the sprint. While the team achieves productivity goals by completing seven issues in total, it fails to maintain quality, with four issues being reopened out of the seven completed ones. This example underscores the significance of considering both productivity and quality aspects in sprint outcomes, as productivity alone does not guarantee high-quality deliverables. Achieving a balance between productivity and quality is crucial for successful sprint execution.

Since Agile methodologies prioritize the ability to adapt to changes (e.g., in requirements), Scrum employs various strategies to incorporate these changes into its processes. For instance, backlog refinement enables teams to continuously revisit and reprioritize user stories, while employing short iteration time-boxes allows teams to plan for brief periods and then replan for subsequent iterations. Consequently, effective sprint planning significantly influences the value derived from Agile practices. Several works have aimed to support practitioners in performing these Scrum activities. For example, DeliveredStoryPoint [19] developed a multi-dimensional regression model to estimate team maturity and the number of delivered points. Additionally, Michael et al. [20] and Phan et al. [21] proposed approaches to predict story points at the issue level using Transformer-Based models and Heterogeneous Graph Neural Networks, respectively. Kortum et al. [8] created a tool called ProDynamics, a Jira plugin, to visualize team performance and display trends in velocity using neural network models for prediction. While these works primarily focus on supporting estimation at the issue level, Ramessur et al. [13] and Choetkiertikul et al. [7] employed machine learning techniques to estimate the delivery capability of a sprint. However, Ramessur et al.'s approach [13] relies on human-justified features such as the quality of requirements, staff experience, and technical ability, with the dataset being simulated for evaluation. Choetkiertikul et al. [7] utilized sprint and issue data using feature aggregation techniques to construct sprint-level features for estimating delivery capability. In our evaluation, we compare our proposed approach with Choetkiertikul et al.'s [7] because both approaches focus on

2. https://issues.jenkins-ci.org/browse/JENKINS-59503
3. https://issues.jenkins.io/secure/RapidBoard.jspa?rapidView=181&projectKey=JENKINS&view=reporting&chart=sprintRetrospective&sprint=81



[GSoC] External Workspace Manager

Sprint Report Switch report

Board

[GSoC] Sprint 1 How to read this chart

Details View linked pages

Started: 04/Jul/16 3:41 PM (planned - 04/Jul/16 3:41 PM) Ended: 25/Jul/16 11:06 AM by Daniel Beck (planned - 25/Jul/16 3:41 PM)

Status Report

* Issue added to sprint after start time

Completed Issues

View in Issue navigator

Key	Summary	Issue Type	Priority	Status	Story Points (-)
JENKINS-35653	Help files enhancements	Task	Minor	RESOLVED	-
JENKINS-36416 *	Create Jenkinsfile in external workspace manager plugin repository	Improvement	Minor	RESOLVED → REOPENED	-
JENKINS-36417	PoC: Flexible Build Selector (based on Copy Artifact plugin) for External Workspace Manager	Task	Minor	RESOLVED → REOPENED	-
JENKINS-36418	Write a design for the extensions points that the EWM plugin may implement	Task	Minor	RESOLVED → REOPENED	-
JENKINS-36484 *	Remove the deprecated code inherited from CopyArtifact plugin	Task	Major	RESOLVED	-
JENKINS-36501 *	Rename packages and classes in Run Selector plugin	Task	Major	RESOLVED → REOPENED	-
JENKINS-36532 *	Additional refactor on Run Selector plugin	Task	Major	RESOLVED	-

Issues Not Completed

View in Issue navigator

Key	Summary	Issue Type	Priority	Status	Story Points (-)
JENKINS-35975 *	Configurable relativePath within the Disk Pool for a job	New Feature	Major	IN PROGRESS	-
JENKINS-35976	Automatic workspace cleanup when using external workspace manager	New Feature	Major	IN PROGRESS	-
JENKINS-36734 *	Implement runSelector Pipeline step in the Run Selector plugin	New Feature	Major	IN PROGRESS	-
JENKINS-36742 *	Integrate Run Selector plugin with External Workspace Manager plugin	New Feature	Minor	IN PROGRESS	-

Fig. 2: An example of a closed sprint report containing completed, incompleted, and reopened issues[3]

predicting at the sprint level and leverage information from issue tracking systems.

2.2 Unsupervised feature learning with deep learning models

Unsupervised feature learning, a powerful technique, leverages deep learning models to automatically learn informative features from data without the need for labeled training examples [22]. This approach has demonstrated promising results in capturing meaningful representations of sequential data, including text. In our approach, we utilize unsupervised feature learning methods to handle two types of sequential data: text descriptions and sequences of developer's activities. By leveraging these techniques, we can extract rich and informative features that capture the underlying patterns and dependencies within the data, enabling us to effectively characterize and analyze sprints in iterative software development.

To convert text data into a numerical representation, various approaches have been developed. These include Bag-of-Words (BoW) [23], which represents a text as a collection of word frequencies; the Skip-gram model [24], which learns word embeddings capturing semantic relationships between words; Term Frequency-Inverse Document Frequency (TF-IDF) [25], [26], [27], which weights the importance of words based on their frequency in a document and their rarity across the corpus; and Latent Semantic Analysis (LSA) [28], which identifies latent topics in a text by performing a matrix factorization. These approaches allow us to transform text into vector representations, enabling the application of machine learning algorithms for text-based tasks. Word2Vec is a popular deep learning model used for unsupervised feature learning on text data. It employs a shallow neural network architecture to learn word embeddings, which are dense vector representations that capture the semantic meaning of words. Another approach is the use of recurrent neural networks (RNNs) [29], [30], such as Long Short-Term Memory (LSTM) networks, which excel in handling sequential data. LSTM models can learn to encode the sequential structure of text, capturing dependencies between words and phrases. This enables them to capture contextual information and generate expressive representations for text data [31]. In addition, transformer-based models, such as the popular BERT (Bidirectional Encoder Representations from Transformers), have revolutionized the field of natural language processing (NLP). Unlike traditional RNNs, which process text sequentially, transformer models rely on a self-attention mechanism to capture contextual relationships between words in a more parallelizable manner. The transformer model, consisting of an encoder-decoder architecture, has the encoder responsible for encoding the input text into a fixed-length representation. It employs attention mechanisms to weigh the importance of different words within the input sequence. By considering both the left and right context, the model can capture contextual dependencies more effectively.

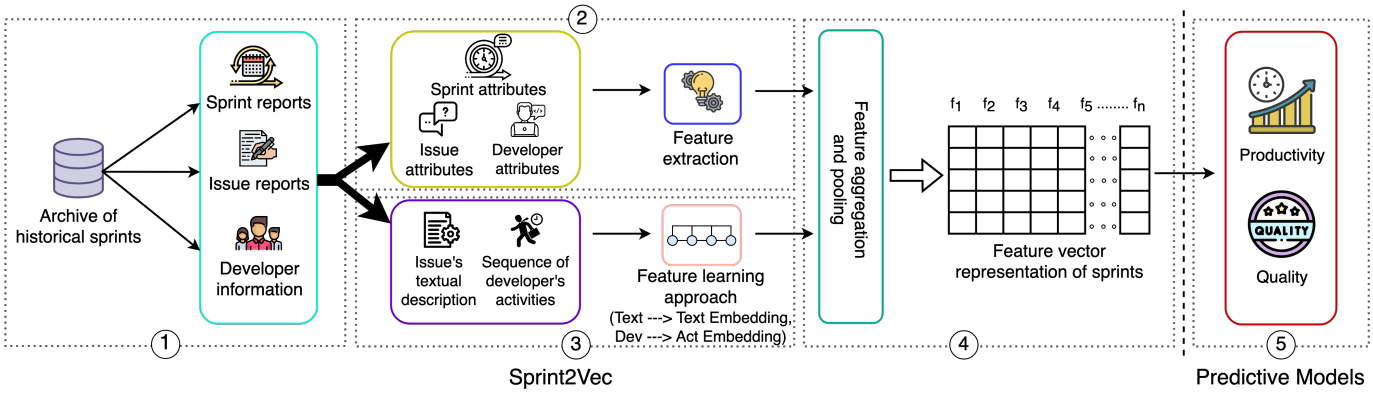


Fig. 3: An overview framework of Sprint2Vec

Moreover, RNNs like LSTM, can be extended to multiple layers, resulting in what is known as a Stacked LSTM. This allows for the addition of multiple hidden layers in the LSTM architecture [32]. The use of Stacked LSTM enables increased model complexity [33]. The output of each intermediate layer can be interpreted as a representation of the original input data. Each layer takes the representation generated by the preceding layer as input, generates new representations as output, and subsequently passes them to the next layer [33]. In various studies, the effectiveness of stacked LSTM models has been demonstrated. For instance, in machine translation tasks, stacked LSTMs achieved improved performance [34]. Stacked LSTMs were also successful in addressing the reconstruction problem in compressed sensing [35], and outperformed single-layer LSTMs in weather forecasting [36].

3 SPRINT2VEC

In this section, we discuss our approach in detail, including feature extraction, feature learning approach, and the construction of vector representation of sprints.

3.1 Overview

Sprint2Vec is an approach for extracting vector representation of sprints in iterative software development. It combines intuitive features with deep learning techniques to learn informative representations. These representations can be used in the training of predictive models. Figure 3 shows the overview of the Sprint2Vec approach.

In (1), Sprint2Vec makes use of three types of information from project repositories containing historical records of sprints and tasks (e.g., issues). A sprint report is a record of sprint's outcomes where we can identify sprint planning data (e.g., committed to delivered user stories, sprint duration), sprint execution data (e.g., task assignments), and sprint outcome data (e.g., number of delivered user stories, number of tasks completed). A work task detail contains information about a work task in the form of an issue report (e.g., issue type, issue priority, issue status, issue resolution, issue assignee, issue reporter, issue description, issue comments, and issue history). These issue reports are usually assigned to be completed in a sprint. Team information is also included in the issue report in the form of assignees involved in resolving an issue (e.g., developer, tester, and integrator). Note that these three types of information can be extracted from project repositories hosted by several well-known project management supporting tools such as Jira Agile and OpenProject. Specifically, in our approach, the data is divided into two distinct groups, each corresponding to different feature extraction methods. The criteria for this division are based on the nature of the data itself. Specifically, we classify data attributes as either structured or unstructured. Structured data, such as attributes from sprint reports, issue reports, and developers, are fed into the feature extraction process. These structured attributes typically include numerical or categorical information that can be readily processed using traditional feature extraction techniques. On the other hand, unstructured data consists of textual descriptions of issues, including issue descriptions (e.g., user stories), as well as developer's activities. These unstructured data elements are inherently more complex and require a different approach. Hence, they are fed into deep learning-based feature learning methods. This segmentation of data into structured and unstructured categories is driven by the distinct nature of these data types. Structured data lends itself well to conventional feature extraction techniques, while unstructured data, due to its complexity, benefits from the capabilities of deep learning-based feature learning.

Thus, in the next step, the attributes (i.e., structured data) of sprint reports, issue reports, and developers are fed to the feature extraction process in (2) (see Section 3.2). Those unstructured data consisted of textual descriptions of issues (e.g., user stories, tasks description) and developer's activities are fed into deep learning-based feature learning methods in (3) (see Section 3.3). Specifically, we employ two deep architectures to learn feature vector representation of sprints from the issue description (i.e., sequence of words) and developer's activities (i.e., sequence of actions) along with types of the corresponding issue. The features extracted from the feature engineering methods and deep learning approaches are then aggregated. Subsequently, they are concatenated into a single vector representation of sprints at (4) (see Section 3.4).

A sprint usually involves a number of issues, and a number of developers contributed to a sprint. Thus, this step aggregates features from each issue and each developer to

derive a vector that can be concatenated with the features at a sprint level. The feature vector representation of sprints can then be used in training predictive models at (5) such as predicting the outcome of ongoing sprints. We, however, note that these features are extracted at the beginning of a sprint (i.e., sprint start time) which we use as a reference point to characterize sprints and make a prediction.

3.2 Feature extraction

There are three groups of data: sprint reports, issue reports, and developers' information that can be extracted as features. In this section, we discuss the extracted features as shown in Table 1. These attributes were chosen because many of them are mandatory and provide essential information for characterizing an iteration. For instance, attributes like sprint length, issue type, and priority are crucial for understanding the nature of a sprint. Additionally, the issues assigned to an iteration play a significant role in characterizing it. Furthermore, the selected features cover various aspects of an issue. An issue's description text, for example, serves as a valuable feature since it elucidates the nature and complexity of the issue. A well-documented description aids participants in understanding the issue better. These pieces of information are commonly available in sprint management tools, making them practical choices for our feature extraction.

3.2.1 The features extracted from a sprint

A sprint is a timeboxing window where a team works on delivering committed issues. A sprint report usually provides the following attributes: start date, end date, and a list of issue reports assigned to be completed within a sprint. We can identify issues and assignees involved in a sprint which can be used to characterize a sprint. Thus, there are three features extracted from a sprint: the planned duration of a sprint, the number of issues planned to be completed within a sprint, and the number of developers that are found as an assignee.

3.2.2 The features extracted from an issue report

An issue report represents software development tasks, such as new feature implementation and bug fixing. It is usually planned to be resolved within a sprint. Thus, the characteristics of issues assigned to a sprint can be used to characterize the sprint. It usually contains an issue type (e.g., new feature, bug) reflecting the nature of the issue, issue priority reflecting the issue's importance relative to other issues, an issue component describing the grouping of similar issues, and fix versions showing versions that the issue is targeted to be released for. In addition, the team's discussion on an issue is also recorded as comments. We extract features from these issue attributes to characterize issue reports assigned to a sprint. Moreover, an issue report also has an issue description in which we employ techniques for textual feature extraction to characterize the issue description, which we discuss in detail in Section 3.3.

3.2.3 The features extracted from an assignee

A developer who is assigned to resolve an issue can be identified from the assignee recorded on an issue report.

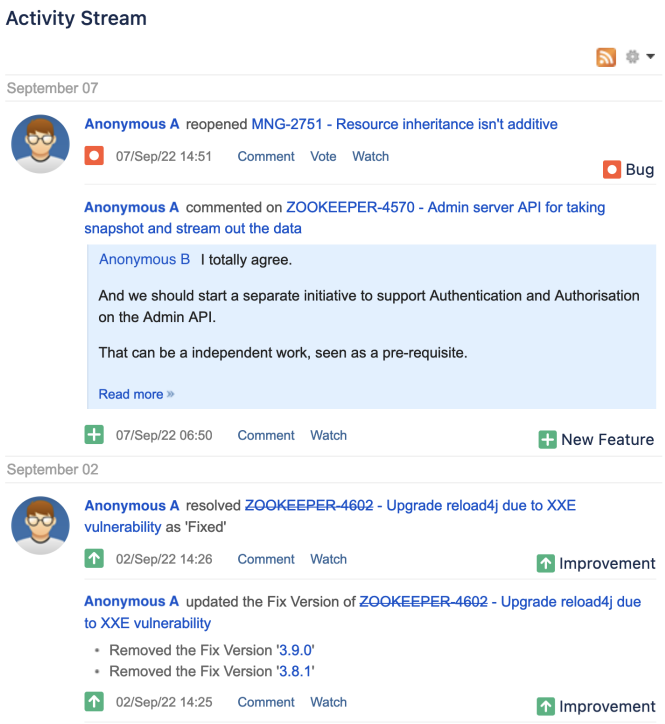


Fig. 4: An example of activity streams on the Jira platform

From the identified developer, we can extract the record of the developer's actions (i.e., activity streams) that the developer contributed to a project in the past. Figure 4 shows an example of the developer's actions recorded in the Jira issue tracking system. A record of the developer's actions usually presents issue-resolving activities such as commenting, implementing, and triaging. Our approach leverages this information to characterize a sprint by two methods which are extracting intuitive features and feeding the action streams to a deep learning architecture to learn feature representation automatically.

We extract four features related to a developer, including the number of unique action types that were performed by a developer in past issues, the number of issues involved with a developer, the number of comments made by a developer, and the issue type that a developer mostly involved with such as bug, new feature request, and change request. Note that these features were extracted before the sprint start time.

3.3 Automatic feature learning

The textual description of an issue and the sequence of developer actions are handled by different methods of automatic feature learning. This section explains how vector features representation of texts and action sequences are derived.

3.3.1 The features learned from an issue description

An issue report regularly contains two pieces of textual information, a summary and a description. For example, the issue *JENKINS-59503* in Figure 1 has a summary as "Plugin does not save settings" and a description says "I can't save plugin settings in the job configuration in Jenkins...". Textual

TABLE 1: List of the extracted features

Feature	Source	Description	Rationale
Planned duration	Sprint	The duration of a sprint, measured in hours, is calculated as the difference between the start date and the planned completion date	This group of features reflects the fundamental characteristics of sprints, including duration, workload, and team size. Longer sprints may lead to fatigue and lower productivity, while shorter ones could result in rushed work and lower quality. A higher number of issues might indicate a heavier workload, impacting productivity and quality. Larger teams may benefit from collaboration but could face coordination challenges.
No. of issues	Sprint	The number of committed issues in a sprint refers to the total number of tasks or user stories that the development team has agreed to deliver within that specific sprint	
No. of team members	Sprint	The total count of developers who are actively working on tasks or user stories within that specific sprint. It represents the size of the development team dedicated to completing the sprint's goals.	
Type	Issue	The category of an issue in a project. It represents the specific classification or label assigned to an issue based on its nature or purpose	These features cover different aspects of issues. For instance, the type and priority of an issue provide insights into its significance and urgency, which can impact the allocation of resources. The number of comments on an issue reflects the level of discussion and collaboration about it, potentially influencing the quality of solutions created. Additionally, metrics such as the number of components, changes in fix versions and priority, and readability score reflect the complexity of the issue, all of which can affect the efficiency and effectiveness of addressing it. Finally, the involvement of developers in specific types of issues prior to the sprint may indicate their familiarity and expertise in handling similar tasks, potentially influencing their performance and contributions during the sprint.
Priority	Issue	Priority refers to the relative importance or urgency assigned to an issue in a project	
No. of comments	Issue	The count of comments that have been posted under an issue in a project. It indicates the level of discussion, collaboration, and engagement around the particular issue	
No. of components	Issue	The number of components indicates how many components an issue is assigned to in a software project	
Changing of fix versions	Issue	The number of times a fix version is changed for an issue indicates the level of uncertainty or instability in determining the appropriate release for fixing the issue	
Changing of priority	Issue	The number of times the priority of an issue is changed reflects the dynamic nature of issue prioritization. It may occur when there are shifts in project requirements, changes in stakeholder priorities, or re-evaluation of the impact and severity of the issue	
Changing of description	Issue	The number of times the description of an issue is changed can indicate the level of refinement or updates made to the understanding and documentation of the issue over time	
Readability score	Issue	By calculating the Gunning Fog index [37] for issue descriptions, we can assess the level of readability and understandability of the text	
No. of unique actions	Developer	Refers to the number of distinct actions performed by a developer before the start of a sprint	The rationale for extracting developer features lies in understanding how their involvement and contributions affect productivity and quality. Active engagement in tasks can drive productivity, indicating efficiency. However, excessive activity may suggest inefficiency, potentially impacting quality. Developers working on multiple issues may have a broader understanding of the project, but handling too many tasks may impact the quality of work.
No. of issues involved with a developer	Developer	Refer to the number of issues in which a developer has performed any actions within three months before the start of a sprint	
No. of comments by dev	Developer	Refers to the number of comments that a developer has posted on issues before the start of a sprint	
Issue type of dev	Developer	Refers to the type of issues that a developer has primarily contributed to before the start of a sprint.	

description (i.e., sequence of words) is considered a rich source of information as it can describe the nature of an issue. We then concatenate the summary and the description and feed it into textual feature extraction techniques.

The BoW represents text as a fixed-length vector, typically equal to the size of the vocabulary, containing word counts. However, this method completely disregards the sequential nature of text [38]. In addition, a problem with the BoW technique is that the words with higher frequency become dominant in the data [39]. To solve this problem, TF-IDF considers how frequently the words occur in the corpus. In our context, a document can be represented by an issue textual description while a corpus (i.e., all documents) can refer to all issue descriptions. Therefore, the importance of frequent words, which are also frequent among all the issue descriptions, is reduced by TF-IDF. The TF-IDF representation consists of two components: (1) Term Frequency (TF) which is the frequency of the word in the description of the current issue, and (2) Inverse Document Frequency (IDF) which is the score of the words among all the descriptions. A word is then defined by a weighted score (W) computing from the product of its TF and its IDF scores [26] (see Equation 1). A single textual issue description is, thus, represented by a single vector of W. However, the TF-IDF representation still cannot capture the order dependencies of words. Moreover, it is context-independent and has the sparsity problem as same as that of BoW.

$$W_{i,j} = tf_{i,j} \times \log(\frac{N}{df_i}) \quad (1)$$

where $W_{i,j}$ is the weight of the word i in the description of the issue j, $tf_{i,j}$ is the frequency of the word i in the issue

description j (Term Frequency), df_i is the number of issue descriptions containing the word i (Document Frequency), and N is the total number of issue descriptions. BoW and TF-IDF are two common techniques used in natural language processing (NLP) for learning features from text data. BoW is simple and easy to understand for text representation. It is also computationally efficient. On the other hand, TF-IDF assigns weights to words based on their frequency within a document and across the entire corpus, reducing the impact of noisy and irrelevant terms in the text.

In contrast, neural network-based embedding techniques, such as Word2Vec [24] developed by Google, aim to overcome the sparsity issue of traditional methods. Word2Vec models are shallow neural networks that input text and output vectors for each unique word in the corpus. Words with similar meanings are situated close to each other in the vector space, enabling Word2Vec embeddings to capture both semantic and syntactic word qualities. However, Word2Vec embeddings remain context-independent, as they maintain a lookup table mapping a single vector representation for each word. Consequently, they cannot discern differences in word meanings based on varying sentence contexts. For our study, we utilized SO_Word2Vec [40] to represent the textual descriptions of issues, as it was specifically pre-trained on StackOverflow data tailored for the software engineering domain.

BERT is a pre-trained transformer-based technique for Natural Language Processing (NLP) developed by Google. It was trained to predict the next word in a sentence. BERT achieves state-of-the-art performance on a number of recent natural language understanding tasks (e.g., GLUE [41], SQuAD [42], and SWAG [43]). Unlike the others, BERT converts a sequence of text input into a sequence of context-aware embedding dense vectors. Specifically, it considers a word as a function of the entire sentence, therefore, a word can have different vectors based on the context. We employed $\text{BERT}_{\text{BASE\_UNCASED}}$ [44], the most common architecture, to represent the textual description of issues. Additionally, we incorporated two other BERT models specifically pre-trained for the software engineering domain: BERTOverflow [45] ($\text{BERT}_{\text{BASE\_CASED}}$-based model) and seBERT [46] ($\text{BERT}_{\text{LARGE\_UNCASED}}$-based model). Although there are other models based on SE data, they are not publicly available. The BERT structure that we used consists of two parts as follows:

(1) The input layer aims to prepare an input sequence for the model by undergoing the WordPiece tokenization process [47] with a predefined token vocabulary. The tokenization is used for segmenting the input sequence, and the split word pieces are denoted with ##. The token of each word piece then yielded the input representations, facilitating subsequent processing.

(2) The BERT encoder consists of multiple transformer encoders (12 for the base model or 24 for the large model) with multiple bidirectional self-attention heads (12 for the base model or 16 for the large model). This layer takes the token sequence and then outputs hidden states with a predefined size (768 for the base model or 1,024 for the large model). Finally, the hidden states are averaged across all hidden sizes to obtain a final vector representation of the text sequence.

We are required to preprocess the textual description of issues to prepare the input for each method. For BoW and TF-IDF, the text is first converted to lowercase, HTML tags, URLs, file references, code blocks, hashes, and special Jira format references are removed. Furthermore, control characters (e.g., newline, carriage return, and tab) are eliminated. Finally, punctuations are cleaned and stop words are filtered out. It is important to note that stemming and lemmatization are intentionally omitted to preserve the nuanced context of word usage. For instance, consider the distinction between "assign" (verb) and "assignee" (noun). Applying these procedures would collapse these distinct forms into a single root, potentially obscuring their intended meanings within the text [48]. For the other methods, we preprocess the text input by following their specific procedures outlined in the original work, with the inclusion of truncation and padding to ensure a consistent input length across all instances. In total, our study considers six methods: BoW, TF-IDF, the pre-trained word2vec-based technique (SO_Word2Vec), and three pre-trained transformer-based techniques (i.e., $\text{BERT}_{\text{BASE\_UNCASED}}$, BERTOverflow, and seBERT.)

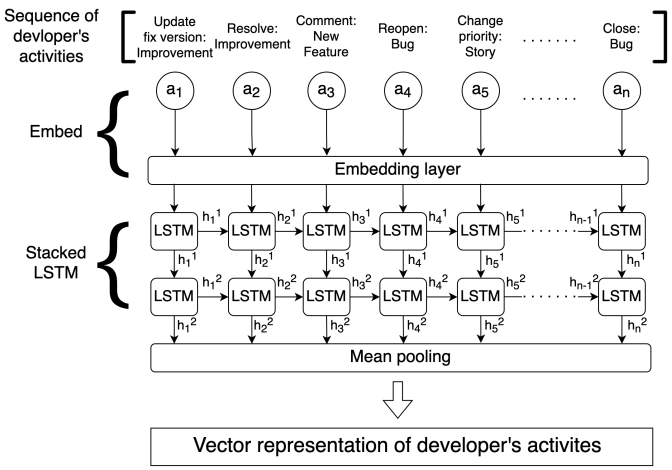


Fig. 5: The stacked LSTM architecture for a developer action

3.3.2 The features learned from developer's actions

A set of developers who are assigned to resolve issues in a sprint can be identified. Their activities performed on issues are recorded. We employ the LSTM to learn features representation of the developer's actions. In a sequence of actions, one action (i.e., token) is constructed from 2 elements. The first is an action type (e.g., create, comment, change priority, close, and reopen). The second element is an issue type (e.g., bug, story, epic, and improvement). Thus, a token can reflect an activity that the developer performs on a specific type of issue. For example, according to the developer activity in Figure 4, we can represent those actions as input sequence tokens $a_1, a_2, ..., a_n$ shows in Figure 5. The token a_1 is that a developer updated fixed versions of an improvement issue report, token a_2 shows that the developer resolved an improvement issue, token a_3 shows that the developer made a comment on a new feature issue, and token a_n shows that the developer closed a bug issue. Note that we use averaging and padding to maintain their fixed length (n).

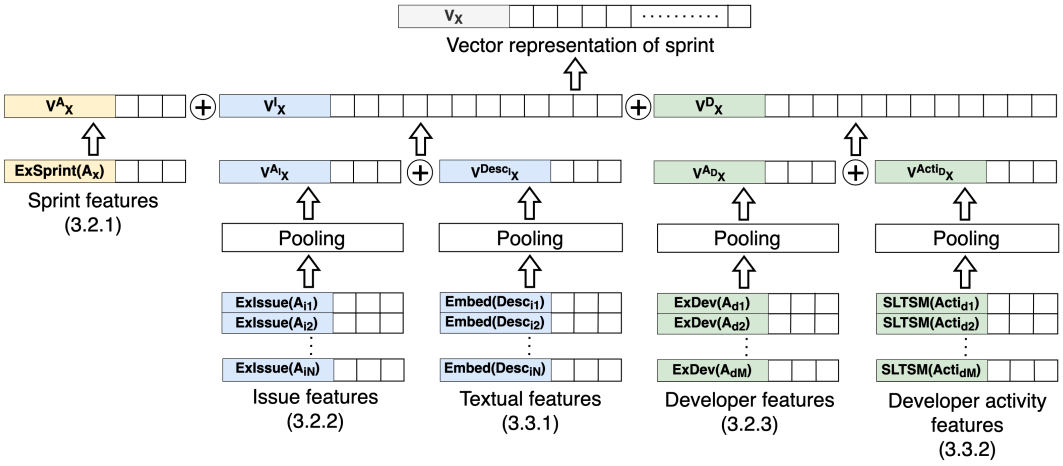


Fig. 6: Constructing a feature representation of a sprint

Each action in the input sequence is represented as a real-valued vector in a low-dimensional semantic space derived from the distribution of activity neighbors by feeding into the embedding layer (*activity embedding layer*). Stacked LSTM is a variation of the standard LSTM model, characterized by multiple LSTM layers stacked on top of each other. The number of layers in a stacked LSTM can be adjusted to enhance the model's ability to capture complex patterns in data. In the stacked LSTM layers, at time step t, a first-layered LSTM cell reads the input a_t and the previous state h^1_{t-1} to compute the output state h^1_t. While a second-layered LSTM cell reads the output from the first layer h^1_t as an input and the previous state h^2_{t-1} to compute the output state h^2_t. With the output states of the last LSTM layer for every activity in the input sequence computed, the *average pooling* technique is employed to derive a final vector representation for a whole sequence of the developer activities.

3.4 Constructing a feature representation of a sprint

A feature representation of a sprint is formed from five sets of features across two different feature extraction methods, including a set of sprint features (Section 3.2.1), a set of issue report features (Section 3.2.2), and a set of developer (Section 3.2.3) features extracted by common feature extraction method; and a set of issue description features (Section 3.3.1) and a set of developer's activity feature (Section 3.3.2) learned by automatic feature learning.

Since a sprint is associated with multiple issues and involves a number of developers, we thus need techniques to consolidate the set of features extracted from issues and developers by both conventional feature extraction and automatic feature learning methods. These techniques aim to derive a new set of features at a parent level (i.e., sprint level) by aggregating the features at a child level (i.e., issue and developer levels). Specifically, the feature aggregation refers to the process of combining or summarizing multiple individual features or attributes into a single, higher-level feature or representation. This aggregation is typically done to simplify data, reduce dimensionality, or create more informative and compact representations using mathematical operations (e.g., averaging) to generate a consolidated representation [49]. As a sprint involves multiple issues and developers, we require methods to consolidate the sets of features extracted from issues and developers using both traditional feature extraction and automatic feature learning techniques. These feature aggregation methods are employed to generate a new set of features at a higher level, specifically at the sprint level, by aggregating the features obtained at the lower levels i.e., the issue and developer levels, as also employed in the previous work [7]. However, in our approach, we introduce flexibility by treating the feature aggregation process as a configurable pooling layer within the deep learning component. This flexibility allows us to use various methods such as averaging, minimum, and maximum pooling, which can be defined during training time. This approach provides adaptability to different scenarios, allowing us to identify the optimal pooling techniques during training based on the specific characteristics of each project. Once the features are aggregated using pooling methods, the aggregated features from each method are then merged into one long representation through feature concatenation. Feature concatenation refers to the process of joining multiple feature vectors or representations together to create a single, extended feature vector. In our context, it involves taking the feature vectors obtained from sprints, aggregated features of issues, and aggregated features of developers, and combining them into the representation vector of the sprint. This process has been visualized in Figure 6.

To do so, we employ the pooling techniques, which allow us to represent a matrix of child-level features with a single vector of parent-level features. Our experiment includes three pooling techniques, namely min pooling, max pooling, and mean pooling. Specifically, minimum and maximum pooling select the minimum and maximum values, respectively, from each feature at the issue and developer levels to represent features at the sprint level. Mean pooling calculates the average value for each feature. It's important to note that no single pooling method universally outperforms others; the choice depends on the specific dataset. Note that a pooling method is determined during the training phase. Each pooling technique has its advantages: mean pooling, for instance, helps reduce variance and outliers in feature values, while maximum and minimum pooling

excel at identifying strong and weak values, respectively, thus handling variations in the input effectively. Finally, we concatenate the pooled vector with the sprint-level feature vector. The output for this step, thus, is the final feature vector representation of a sprint that can be directly used in predictive model training.

To formulate how a vector representation of sprint is constructed from Sprint2Vec, given a software system, let X, I, and D be the set of sprints, issues, and developers, respectively. A sprint $x = \langle A_x, I_x, D_x \rangle \in X$ is a tuple of sprint attribute set A_x, issue set $I_x = \{i_1, i_2, ..., i_N\} \in I$, and developer set $D_x = \{d_1, d_2, ..., d_M\} \in D$, associated with x. An issue $i = \langle A_i, \text{Desc}_i \rangle \in I$ is a tuple of an associated attribute set A_i and textual description Desc_i. A developer $d = \langle A_d, \text{Acti}_d \rangle \in D$ is defined as a tuple of an associated attribute set A_d and a sequence of activities Acti_d.

The objective of Sprint2Vec is to learn a vector representation of the sprint x, symbolized as v_x, i.e., $v_x = f(x)$, where $f(\cdot)$ is a feature extractor. v_x is formed by concatenating three sprint-level feature vectors v_x^A, v_x^I, and v_x^D, representing the vector representations of the sprint attributes, associated issues, and associated developers in the sprint, respectively, as defined in Equation 2. The $\oplus$ symbol represents the concatenation operator.

$$v_x = f(x) = v_x^A \oplus v_x^I \oplus v_x^D \tag{2}$$

The feature vector of the sprint attributes, v_x^A, can be represented as follows:

$$v_x^A = \text{ExSprint}(A_x)$$

where $\text{ExSprint}(\cdot)$ is the common feature extraction for a sprint. The feature vector representation of the associated issues, v_x^I, can be represented as follows:

$$\begin{aligned} v_x^I &= v_x^{A_I} \oplus v_x^{\text{Desc}_I} \\ v_x^{A_I} &= \text{Pooling}(\{\forall i \in I_x : \text{ExIssue}(A_i)\}) \\ v_x^{\text{Desc}_I} &= \text{Pooling}(\{\forall i \in I_x : \text{Embed}(\text{Desc}_i)\}) \end{aligned}$$

where $v_x^{A_I}$ and $v_x^{\text{Desc}_I}$ denote the pooled attribute and description vectors, respectively, of all the issues in the sprint, while $\text{ExIssue}(\cdot)$ is the common feature extraction for an issue, $\text{Embed}(\cdot)$ is one of the automatic feature learning functions for text (i.e., BoW, TF-IDF, and BERT), and $\text{Pooling}(\cdot)$ represents one of the pooling techniques (i.e., minimum, maximum, and mean). The feature vector representation of the developers, v_x^D, can be represented as follows:

$$\begin{aligned} v_x^D &= v_x^{A_D} \oplus v_x^{\text{Acti}_D} \\ v_x^{A_D} &= \text{Pooling}(\{\forall d \in D_x : \text{ExDev}(A_d)\}) \\ v_x^{\text{Acti}_D} &= \text{Pooling}(\{\forall d \in D_x : \text{Embed}(\text{Acti}_d)\}) \end{aligned}$$

where $v_x^{A_D}$ and $v_x^{\text{Acti}_D}$ denote the pooled attribute and activity feature vectors of all the developers in the sprint, respectively, while $\text{ExDev}(\cdot)$ is the common feature extraction for a developer and $\text{SLSTM}(\cdot)$ is automatic feature learning function for a sequence of developer activities which is our LSTM.

With the above definition of v_x, each $x \in X$ can be represented in a latent space with $|v_x|$ dimensions. However, assessing the efficacy of the proposed sprint representation learning method needs validation at the predictive tasks. In this research, predictive models are trained on the sprint representation to predict sprint performance indicators or target outcomes $Y = \{y_1(\cdot), y_2(\cdot), ...\}$, where $y_1(x)$ is the corresponding actual target value of the sprint x. For instance, Y could represent the actual target values corresponding to a sprint, such as productivity and quality. Mathematically, the training set $T = \langle X_T, Y_T \rangle$ contains a mapping of sprints $X_T \subset X$ and ground-truth actual target variables Y_T. We use T to build a set of predictors $H = \{h_1(\cdot), h_2(\cdot), ...\}$ that takes the low-level representation of the sprint x (i.e., $v_x = f(x)$) and approximates Y_T. That is, $h_i(f(x)) \approx y_i(x)$.

4 Evaluation

4.1 Predictive tasks

In our evaluation, we mainly assess Sprint2Vec-derived features in two regression tasks: predicting sprint productivity and quality, both quantifiable outcomes at the end of sprints. The details of these two prediction tasks are as follows.

4.1.1 Sprint productivity

Firstly, the amount of work delivered at the end of the sprint can reflect the team's productivity (i.e., the delivery capability). This standard measurement is used in sprint progress monitoring and reflects the quantitative aspect of a sprint outcome. We quantify it as the completion ratio of committed issues, defined as the quotient of completed issues to committed issues in a sprint. Specifically, completed issues are tasks assigned within a sprint that the development team finishes when the sprint ended, while committed issues are tasks the team agrees to complete during the sprint.

4.1.2 Sprint quality

Secondly, we also consider the quality aspect of the sprint outcome. The completed issues delivered from a sprint can be reopened for several reasons, such as errors being identified after issue resolution and a solution not satisfying a quality checking practice (e.g., testing, code review). These reopened issues affect the performance of a scrum team since it requires extra effort to rework those issues [17], [18]. We thus make use of the issue reopen status to quantify the quality of sprints. From the historical record of an issue (i.e., issue changelog), we can identify whether issues are reopened after resolving them in a sprint. Therefore, sprint quality is determined by the rate of issues reopened, calculated as *the number of issues reopened after the sprint ends divided by the total number of committed issues* (with a lower rate indicating better quality).

Evaluating our methodology through predictions of sprint productivity and quality is essential, as they are key to agile team performance. Sprint productivity is an indicator of a team's effectiveness in achieving its objectives. Meanwhile, sprint quality underscores the necessity of completing tasks to a high standard, ensuring robust solutions that

meet quality benchmarks and reduce the need for further adjustments. These predictive tasks thus offer a thorough assessment of our approach, capturing the essence of an agile team's performance and ensuring that both efficiency and quality are enhanced in project delivery.

4.2 Research questions

In our evaluation of the approach, we formulate five distinct research questions to guide our investigation and assess its efficacy.

RQ1: Sanity check - How does our approach compare to baseline methods?

As part of our sanity checks, we compared the predictive performance achieved from the prediction model using Sprint2Vec to three common baseline benchmarks, which are random guessing, mean estimation, and simple linear regression, on the predictive tasks mentioned in Section 4.1. These benchmarks are frequently used in the software engineering research [7], [50], [51]. Specifically, the inclusion of random guessing, mean estimation, and simple linear regression as naive baselines serves as a sanity check to ensure that our model's performance is reasonable and not worse than basic methods. Firstly, random guessing involves making predictions by randomly selecting a target variable of one sprint from a training set and assign it as an estimate to a new sprint (test set). Mean estimation, on the other hand, predicts by assigning the mean of the past sprints' target variables as an estimate for a new sprint. It is important to note that we add or subtract the mean by random noises sampled within the range of one standard deviation of the past sprint's target variables to introduces an element of randomness to account for potential variations. Both random guessing and mean estimation lack the utilization of any specific information associated with the target sprint. Consequently, any effective estimation model should surpass the predictive performance of these methods. Simple linear regression utilizes Ordinary Least Squares (OLS) to make predictions for a new sprint. We specifically use only sprint features to avoid overfitting due to high dimensionality in model training [52]. We expect our approach to surpass the performance of these baselines. Note that the samples used for all the baselines were from the training set.

RQ2: Does the incorporation of features derived from Sprint2Vec lead to improved predictive performance compared to a specific existing approach?

The objective of this research question is to compare the predictive performance between the existing approach and our Sprint2Vec approach. The existing approach utilizes statistical feature aggregation techniques (e.g., mean, maximum, minimum, standard deviation, and variance) along with issue information to characterize the sprints proposed by Choetkiertikul et al. [7]. Our objective is to demonstrate the usefulness of textual description features and the sequence of the developer's activities derived from automatic feature learning in characterizing the sprint, thereby showcasing that our approach outperforms the existing approach. Note that to ensure a fair comparison, both the existing and the proposed approach use t_0 as the prediction time, which corresponds to the sprint start time.

RQ3: Do the features derived from Sprint2Vec enhance the predictive performance compared to alternative combinations of feature sets?

We aim to evaluate the predictive performance of Sprint2Vec by comparing it to alternative approaches. Specifically, our approach combines five sets of features, including sprint features, common features extracted from issues, and three novel sets of information: text features extracted from issues, common features of developers, and activity features of developers. By constructing a comprehensive representation of sprints using these feature sets, we aim to demonstrate their complementary nature. To achieve this, we conduct a comparative analysis of Sprint2Vec against five alternative methods that incorporate different combinations of feature sets during model training: (1) the model trained from only the features extracted from sprints, (2) the model trained from only the features extracted from sprint, issues, and developers, (3) the model trained from a set of features that excludes the features related to developers, (4) the model trained from a set of features that excludes the features related to issues, and (5) the model trained from a set of features that excludes the features from textual description of issues. Specifically, this RQ explores the use of these new features and their potential impact on performance. It serves as an investigation into the synergy of combining different feature sets, treating them as configurable components within our approach.

RQ4: Do the features derived from Sprint2Vec improve predictive performance compared to the state-of-the-art software engineering feature embedding technique?

To address this research question, we have integrated a state-of-the-art technique for characterizing information within sprints by employing seBERT [46] in the feature learning process. seBERT is an advanced method that builds on the well-known BERT embedding model, specifically tailoring it for software engineering tasks. seBERT has shown exceptional performance in understanding domain-specific vocabulary, identifying missing words, and excelling in classification tasks, outperforming other recent transformer-based approaches. Moreover, it has demonstrated high effectiveness in downstream tasks, such as predicting issue types [53], making it a robust benchmark for comparison. The model was originally trained on a comprehensive corpus of SE textual data, including StackOverflow posts, GitHub issue descriptions, Jira issue descriptions, and GitHub commit messages. To represent its approach in the study, we thus utilize seBERT to embed the issue textual description, providing a baseline for comparison. Our objective is to evaluate effectiveness of Sprint2Vec-derived features against these seBERT embeddings, aiming to establish whether Sprint2Vec can provide superior predictive capabilities over state-of-the-art, software engineering-specific feature learning models.

RQ5: Does the Sprint2Vec approach outperform existing baseline methods in predicting the number of completed issues and reopened issues at the issue level?

This question focuses on applying the Sprint2Vec approach to two additional predictive tasks at the issue level: predicting the number of completed issues at the end of a sprint and predicting the number of issues that are reopened after the end of a sprint. These predictive tasks thus

complement the proposed predictive tasks by providing a clearer reflection of base productivity and quality of sprint outcomes without conflating them with sprint commitment. Note that while our proposed approach aims to provide predictions at the sprint level, these two additional tasks allow us to incorporate state-of-the-art methods for issue-level prediction. Thus, we carefully considered baselines capable of performing similar predictive tasks at the issue level. We have added the issue resolution prediction model proposed by Mora et al. [54] as our baseline for predicting the number of completed issues at the end of a sprint. For predicting the number of issues reopened after the end of a sprint, we added the issue reopen prediction model proposed by Shihab et al. [55]. To the best of our knowledge, there is no existing method that aims to predict at the sprint level except the approach proposed by Choetkiertikul et al. [7], which is already included in our baseline. The selected baseline methods perform predictions at the issue level, and their outputs are then aggregated to the sprint level, for example, if Issue A and Issue B are both predicted to be completed and belong to Sprint 1, the final prediction for the number of completed issues in Sprint 1 would be two. To ensure the validity and reliability of our comparisons, we have carefully implemented the approaches described in the baseline papers, adhering closely to the methodologies and explanations provided by the authors. Furthermore, to maintain fairness, we ensured that both our approach and the new baseline approaches employ only the features available at the sprint start date, which is our prediction time. We also selected the best classification model with the optimal configuration as stated in their work. Specifically:

- In predicting issue completion, the baseline utilized the textual descriptions and comments of issues by employing TF-IDF with an n-gram range of two to ten. The prediction model employed Multinomial Naive Bayes (MNB) to train the classifier.
- In predicting issue reopens, the baseline extracted ten available features, including component, priority, description size, Bayesian score derived from description text, number of comments, comment size, Bayesian score derived from comment text, priority changes, reporter name, and reporter experience. The baseline employed a Decision Tree with the C4.5 algorithm as their classifier.

By faithfully replicating these baseline approaches and using comparable features, we aim to minimize potential threats to the accuracy and fairness of our evaluation, thereby ensuring a robust comparison.

4.3 Dataset

To evaluate our approach based on the two predictive tasks, i.e., sprint productivity and quality, we develop the datasets for our study from five well-known open source projects: Apache, Atlassian, Jenkins, Spring, and Talendforge, that use the Jira issue tracking system as their sprint and issue management tool. We collected sprint reports, issue reports, and developer activities using the Jira REST APIs and Jira Agile APIs. We collected the sprints that were created from May 2012 to April 2022. During data preprocessing, we implemented data checking procedures by comparing the criteria used in our data collection with the data available on the Jira website, such as the number of sprints and issues. This cross-validation helped us verify the accuracy and consistency of the collected data. Subsequently, during the data preprocessing stage, we applied rigorous filters to enhance data quality. We excluded ongoing sprints and those with zero assigned issues, as well as removed sprints with an excessive number of assigned issues, treating them as potential outliers. As we can identify developers assigned to those issues, for each developer, we collected the activity recorded three months before the sprint start date to derive the developer's activity features. This is also to avoid the information leakage issue in our evaluation, as our prediction uses only the past sprint data, e.g., the past activity before the sprint start date. The collected data are process to extract the features by examining JSON files of sprints, issues, issue changelog, and developer activities. For example, the list of complete/incomplete issues of a sprint can be extracted from Sprint's JSON files. The status of issues can be determined from issues' changelog JSON files. Developer activities are retrieved from the Jira API as a list of actions in JSON format. A detailed description of the Jira JSON file structure can be found in Jira's API description.[1] In addition, we then examined the distribution of issue assignment and conducted thorough validation of extracted features based on issue changelogs, which contain comprehensive transaction records related to issues. This examination, particularly in handling issue change logs, was essential to prevent any data leakage during feature extraction. For instance, we ensured that the number of issues assigned to a sprint at the sprint start time (prediction time) was processed from the issue changelog, while the number of completed issues in the sprint found in the sprint report might differ. This meticulous approach guarantees the accuracy of our feature extraction process. Additionally, for textual data and developer activities, we conducted validation by examining the distribution of data characteristics, such as the length of textual descriptions and the number of developer activities. These analyses enabled us to detect any anomalies in our data extraction process and maintain data quality. To manage incomplete records (i.e., missing data), we implemented specific strategies to maintain data integrity. For missing categorical data, such as issue priority, we filled in the gaps with the most frequent value, ensuring no valuable information was lost. For developers with less than three months of activity, we used zero padding to keep our dataset consistent.

In total, our study was performed on 7,987 sprints, 118,346 issues, 1,037 developers, and over three million records of developer activities. Note that we have included all projects available in these five repositories (Apache, Atlassian, Jenkins, Spring, and Talendforge) in our experiment. This inclusive approach was chosen because it is common for teams within these repositories to work across multiple projects. Furthermore, projects within a single repository often share common development flows. Table 2 shows

1. Jira REST API:https://docs.atlassian.com/software/jira/docs/api/REST/7.6.1/ and Jira Agile API: https://docs.atlassian.com/jira-software/REST/7.0.4/

TABLE 2: Statistical information of the sprints, issues, developers, and developer's activities

Project	# Sprints	# Issues	# Developers	# Activities	# Issues per Sprint		# Developers per Sprint		# Activities per Developer	
					Min/Max	Mean/SD	Min/Max	Mean/SD	Min/Max	Mean/SD
Apache (AP)	904	8,776	418	667,852	1/90	9.71/10.10	0/30	5.54/4.07	0/2004	133.41/187.73
Atlassian (AT)	3,018	11,441	280	241,779	1/34	3.79/3.63	0/13	2.67/1.68	0/674	29.97/54.07
Jenkins (JE)	1,420	60,227	66	1,110,574	1/92	42.41/23.78	0/19	10.60/3.95	0/1000	73.78/187.38
Spring (SP)	835	22,954	56	1,068,159	1/191	27.49/33.71	1/16	6.51/4.04	0/998	196.39/206.88
Talendforge (TA)	1,810	14,948	217	611,617	1/75	8.26/9.94	0/14	3.40/2.66	0/969	99.39/127.50
Total	**7,987**	**118,346**	**1,037**	**3,699,981**						

Sprints = Number of sprints, # Issues = Number of issues, # Developers = Number of developers, # Activities = Number of developers' activities

TABLE 3: Statistical information of the sprint productivity and quality outcomes

Project	Sprint productivity		Sprint quality	
	Min/Max	Mean/SD	Min/Max	Mean/SD
Apache (AP)	0.06/49.40	1.52/3.44	0.00/14.00	0.80/1.53
Atlassian (AT)	0.06/36.00	1.07/1.42	0.00/36.00	0.92/1.35
Jenkins (JE)	0.07/10.00	0.96/1.28	0.00/2.60	0.46/0.45
Spring (SP)	0.03/29.00	1.78/2.57	0.00/14.50	0.81/1.48
Talendforge (TA)	0.03/65.00	1.75/3.28	0.00/65.00	1.22/2.96

the number of sprints, issues, developers, and developers' activities for each project in our datasets. It also reports the number of issues per sprint, the number of developers per sprint, and the number of activities per developer in terms of minimum, maximum, mean, and standard deviation to show the descriptive statistics of our dataset. These five projects exhibit distinct characteristics. For instance, Apache involved 418 developers across 904 sprints, while Jenkins had only 66 developers but conducted more than one thousand sprints. Conversely, Apache averaged five developers per sprint, whereas Jenkins had an average of ten developers per sprint.

As outlined in Section 4.1, we evaluate two key sprint outcomes: sprint productivity and sprint quality. Table 3 presents the outcomes' descriptive statistics, including minimum, maximum, mean, and standard deviation for each project. The diversity of these five projects enriches our dataset. Specifically, sprint productivity varies significantly among projects, with a low of 0.03 in Spring and Talendforge and a high of 65 in Talendforge. Sprint quality ranges from 0.00 (consistent across all projects) to 65.00 in Talendforge. Sprint productivity's mean values span from 0.96 in Jenkins to 1.78 in Spring, with standard deviations ranging from 1.28 in Jenkins to 3.44 in Apache, indicating Apache's higher mean productivity alongside greater fluctuation. For sprint quality, mean values vary from 0.46 in Jenkins to 1.22 in Talendforge, with standard deviations from 0.45 in Jenkins to 2.96 in Talendforge, reflecting the range of quality outcomes.

4.4 Experimental setting

We utilized a train-validation-test split to partition our dataset based on a 60:20:20 ratio. The sprints were sorted by their start dates and then allocated to the training, validation, and testing sets accordingly. This technique mirrors a real deployment scenario in an Agile software development project, where the model is trained on data from past sprints to predict the metrics or factors for the current sprint. Employing this technique ensures that the predictions are grounded in historical data, enhancing their reliability for deployment. As shown in Figure 7, the sprints in the training and validation sets were created before those in the testing set. Additionally, the sprints in the training set preceded those in the validation set. To prevent information leakage, we confined our information usage to data available at the prediction time, such as sprint reports, issue reports, and developer activities. This strategy ensured our predictions were grounded in data realistically accessible at deployment, thereby reducing potential biases or overfitting. Our model's configuration and training were customized to each project's specific traits.

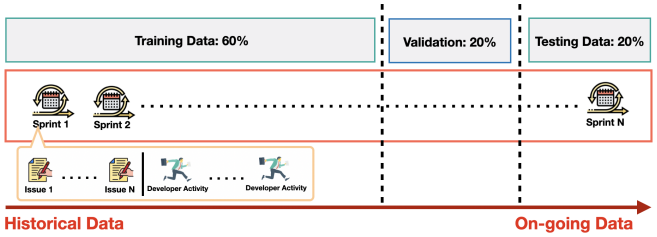


Fig. 7: The splitting of training set, validation set, and testing set

4.5 Performance measures

In our analysis, we used the Mean Absolute Error (MAE) to assess performance across two predictive tasks. The choice of MAE was critical for accurately capturing prediction error in both contexts. MAE provides a straightforward quantification of the model's prediction accuracy, free from underlying assumptions. It treats all errors equally, avoiding bias towards either overestimation or underestimation. This fairness ensures that the error metric genuinely represents the model's performance without distortion. Additionally, MAE's assumption-free nature renders it adaptable to a wide range of data types and distributions, a crucial feature given the potential variance in data characteristics between the two tasks. Moreover, MAE stands as a common metric in regression problems, consistently employed as the evalu-

ation criterion for several research works within the field of software engineering [7], [20], [50].

We also applied statistical methods to confirm the effectiveness of our approach. Specifically, we used the Wilcoxon Signed-Rank Test [56], a non-parametric statistical technique, to test the significance of the performance comparison between the two predictive models by analyzing the absolute errors generated by each model. We formulated the following null hypothesis (H_0): the absolute errors provided by model A are larger or equal than those achieved by model B, while the alternative hypothesis (H_a): the absolute errors provided by model A are less than those achieved by model B. A significance level of 0.05 ($p < 0.05$) was applied to determine statistical significance. However, it is insufficient to rely solely on demonstrating statistical significance; we must also assess whether the observed effect size is substantively meaningful and of practical interest [50]. To address this aspect, we employed the Vargha and Delaney's A^{12} statistic [57], a non-parametric effect size measure. For a given performance measure (i.e., the absolute error), denoted as M, the $\hat{A}^{12}$ statistic gauges the probability that model A produces superior M-values compared to another model B. This probability is determined by the following formula:

$$\hat{A}^{12} = \frac{r_1/m - \frac{(m+1)}{2}}{n}$$

where r_1 represents the rank sum of observations where M achieves better results than N, and m and n are respectively the number of observations in the samples derived from M and N. If the two models are equivalent, then $\hat{A}^{12}$ = 0.5. In cases where M outperforms N, $\hat{A}^{12} > 0.5$, and vice versa [58]. Both the Wilcoxon Signed-Rank Test and Vargha and Delaney's A^{12} statistic are a robust test to apply because there is no assumptions about underlying data distribution. Furthermore, these tests have been suggested and widely utilized in numerous research endeavors within the field of software engineering [7], [50], [59], [60].

While our primary statistical comparison is based on Mean Absolute Error (MAE), we also report performance using two additional metrics: Mean Squared Error (MSE) and Root Mean Squared Error (RMSE). MSE emphasizes larger errors by squaring them, capturing error variance and revealing significant prediction deviations. RMSE, with its square root adjustment, keeps the error metric in the same unit as the dependent variable and is also sensitive to outliers. This sensitivity improves the understanding of predictive accuracy when outliers are present. The coefficient of determination (R^2) is less effective for non-linear relationships [61], which deep learning often addresses better. Thus, MSE and RMSE provide more useful insights for models with possible non-linear patterns.

4.6 Model training

In our study, we trained our model using a designated training dataset, and a separate validation dataset was instrumental in fine-tuning the model's configuration. We developed the Sprint2Vec model with Keras in Python, concentrating on two key aspects: an automated feature learning component for analyzing developer actions, and a regression task predictor. Our experimentation thus involved exploring various model configurations (e.g., model architecture, number of layers, hidden units, activation functions, and batch normalization) and varying training hyperparameters (e.g., optimizer, learning rate, and batch size).

Regarding the extraction of features from textual descriptions of issues and developers' activities, both types of data display sequential patterns that are essential for conveying the semantic core of the information, highlighting shared characteristics. Therefore, it is essential to utilize techniques that can effectively capture and retain this sequential data to maintain its semantic integrity within the representation of sprints. Regarding language models, there is a wide array of popular applications and pre-trained language models available such as $\text{BERT}_{\text{BASE\_UNCASED}}$, $\text{BERT}_{\text{CASED}}$, and $\text{BERT}_{\text{LARGE}}$. We acknowledge and leverage for learning features from text. Thus, we address the variability among existing pre-trained models as configuration components in our approach, enhancing flexibility in feature extraction from text, particularly in scenarios with constraints such as limited computational power. On the other hand, capturing developer activities necessitates techniques capable of handling long sequences with varied elements. These techniques must also have the ability to ignore information from overly long sequences, mimicking scenarios where past actions may not significantly influence the current situation. The longer the time that has passed, the less relevance it holds to the current scenario. This led us to employ LSTM. However, LSTM comes with variations, including the number of stacked layers and directions. As a result, we improved transparency in model training and reduced selection bias in certain elements of our methodology. We focused on the pre-trained language model and the quantity of LSTM layers, in addition to hyperparameter adjustments such as learning rate and dropout rate, as key configuration components in our approach.

For the developer action feature learning model, we selected the best model based on the perplexity metric during validation [62]. The perplexity value is a common evaluation method for a language probabilistic model. As it is defined as the inverse of aggregated probabilities of each activity, the lower perplexity of a model shows that it is good in predicting the next activity of a sample. Additionally, we use the average length of sequences in textual and developer activities from the training set as the length of input tokens for training. This method enables us to accommodate the variations inherent in different project's natures.

For training the regression task predictors, we used MAE as the loss function. Before training, we normalized the input data to ensure consistent scaling across features. Furthermore, we incorporated the early stopping technique to continuously monitor the model's performance and terminate the training when the metric in the validation phase showed consecutive signs of degradation. During the training of the regression layer, we employ an automatic regression model that utilizes neural networks (NN) with dropout regularization. This technique helps prevent overfitting by randomly turning off some neurons during training, improving the model's performance on new, unseen

data. Finally, the testing dataset was used to evaluate the performance of the trained regression models. The selected configurations of our approach from the model training are provided in our replication package.

Furthermore, after we derived the trained model of all projects, we have conducted an analysis of various model configurations, including hyperparameter settings, feature selection methods, and model architectures. Our investigation reveals the first observation regarding the relationship between the complexity of automated feature learning models and the associated data. Specifically, we found that as the average length of issue textual descriptions increases, the complexity of the models also tends to increase. For instance, as shown in Figure 8 (a), the Spring project, which has the shortest average description length, which has the shortest average description length, the simplest feature learning technique i.e., Bag of Words (BoW) offers the best performance. In contrast, the Atlassian project, which has the longest average description length, uses the most complex technique, seBERT, with an embedding dimension of 1024. This suggests that projects with more detailed issue descriptions may benefit from more sophisticated feature learning models that can capture the nuances and intricacies of the text.

A similar pattern is observed in the developer activity learning. As shown in Figure 8 (b), the Spring project has the longest average sequence length for developer activities. To manage this complexity, it uses a more advanced RNN model configuration with separate embedding layers for the encoder and decoder. In contrast, other projects with shorter sequences use shared embedding layers. This suggests that the more complex sequences in Spring's developer activities require separate embedding layers to effectively capture the details and patterns within the data.

Next, we identified that Jenkins is the only project that utilizes the same regressor model configuration for both predictive tasks. Notably, Jenkins also displays a low-variance distribution for both sprint outcomes, as shown in Table 3. This may suggest a relationship between the choice of model configuration and the distribution of sprint outcomes. The consistent use of the same regressor configuration could contribute to the stable outcomes observed in Jenkins' sprint performance, implying that uniform model configurations might help reduce variability in sprint results.

4.7 Results

Table 4 reports the evaluation outcomes for research questions RQs 1-3, summarizing the performance of our Sprint2Vec approach against other methods. We compare predictive accuracy and statistical significance through MAE, MSE, RMSE, percentage improvement, Wilcoxon p-value, significance level, and A^{12} effect size across five projects: Apache (AP), Atlassian (AT), Jenkins (JE), Spring (SP), and Talendforge (TA), covering ten cases in total.

RQ1: Sanity check - How does our approach compare to baseline methods?

The results demonstrate that Sprint2Vec consistently outperforms the baseline methods across all scenarios for both predictive tasks, with significant reductions in Mean Absolute Error (MAE), Mean Squared Error (MSE), and Root Mean Squared Error (RMSE). For example, in the productivity task for the Spring project, Sprint2Vec achieved superior performance with an MAE of 0.72, an MSE of 1.55, and an RMSE of 1.25. In comparison, the mean estimation method resulted in an MAE of 3.06, an MSE of 14.13, and an RMSE of 3.76. Random guessing yielded an MAE of 1.77, an MSE of 13.11, and an RMSE of 3.62, while linear regression produced an MAE of 2.84, an MSE of 10.15, and an RMSE of 3.19. Similarly, in the Spring project's quality task, Sprint2Vec achieved an MAE of 0.16, an MSE of 0.12, and an RMSE of 0.35, showing substantial improvement over the mean estimation method, which had an MAE of 1.77, an MSE of 4.83, and an RMSE of 2.20. Sprint2Vec also greatly outperformed random guessing, which had an MAE of 1.11, an MSE of 5.89, and an RMSE of 2.43, and linear regression, which had an MAE of 1.58, an MSE of 3.13, and an RMSE of 1.77. These results further highlight Sprint2Vec's ability to deliver more accurate predictions than the baseline methods across multiple metrics. Moreover, Sprint2Vec shows statistically significant improvements over mean estimation and random guessing in all instances, as indicated by p-values below 0.05 ($p < 0.05$) and effect sizes greater than 0.5 ($A^{12} > 0.5$). However, we acknowledge that in two of the ten cases, our approach did not show significant statistical improvement compared to the linear regression method. These instances exhibited low variability in their outcomes, which may inherently favor simpler models, leading to less differentiation in performance metrics (MAE).

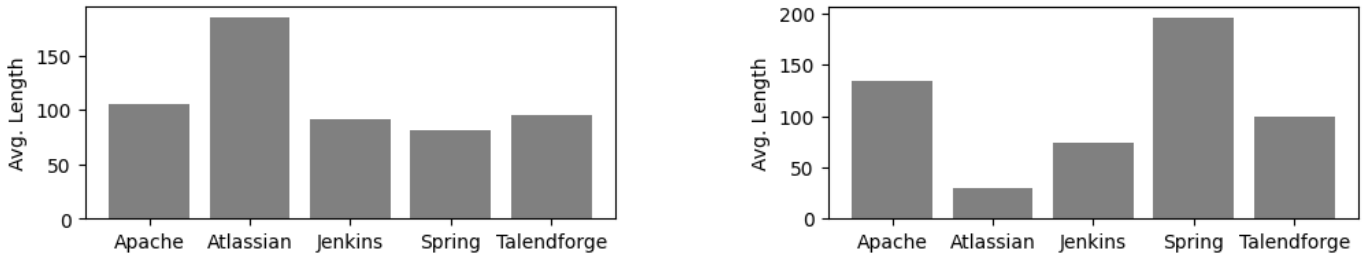
RQ2: Does the incorporation of features derived from Sprint2Vec lead to improved predictive performance compared to a specific existing approach?

The results consistently indicate that Sprint2Vec outperforms the existing approaches across all metrics in both productivity and quality tasks. For example, in the Apache project's productivity task, Sprint2Vec achieved an MAE of 1.04 compared to the existing approach's MAE of 1.58, representing an improvement of 33.96%. Sprint2Vec also demonstrated better performance in MSE and RMSE, with values of 3.38 and 1.84, respectively, compared to the existing approach's 6.23 (MSE) and 2.50 (RMSE). This highlights Sprint2Vec's ability to more accurately predict sprint productivity. Similarly, in the quality task for the Apache project, Sprint2Vec outperformed the existing approach with an MAE of 1.15 versus 1.27, representing an improvement of 9.78%. Sprint2Vec also showed lower MSE and RMSE values of 4.88 and 2.21, respectively, compared to 5.46 (MSE) and 2.34 (RMSE) for the existing approach. Moreover, Sprint2Vec demonstrates statistically significant improvements over the existing approach in all cases ($p < 0.05$) and with effect sizes exceeding 0.5 ($A^{12} > 0.5$). These results underscore the benefits of integrating Sprint2Vec-derived features in improving prediction accuracy over traditional methods across diverse software development projects.

RQ3: Do the features derived from Sprint2Vec enhance the predictive performance compared to alternative combinations of feature sets?

We compared the predictive performance achieved by Sprint2Vec with five alternative combinations: Alternative (1) involved using only traditional features extracted from sprints, omitting all issues features and developer's features, Alternative (2) included all traditional features extracted

Fig. 8: Average length of issue textual descriptions and sequences of developer activities across projects



(a) Issue textual descriptions

(b) Sequences of developer activities

from sprints, issues, and developers, while omitting textual issue features and developer activities' features, Alternative (3) utilized traditional features extracted from only sprints and issues, along with textual features, omitting all developer features, Alternative (4) employed traditional features extracted from sprints and developers, including developer activities' features, while omitting all issue features, and Alternative (5) used all traditional features extracted from sprints, issues, and developers, while omitting only text issue features.

In most scenarios for both tasks, our approach consistently outperforms the alternatives in terms of MAE, MSE, and RMSE, as well as in statistical significance ($p < 0.05$ and $A^{12} > 0.5$). Specifically, our approach shows superior performance to Alternative (1) in nine out of ten cases. Against Alternative (2), it excels in eight cases, and similarly, it exceeds the performance of Alternative (3) and Alternative (4) in eight cases each. Against Alternative (5), our approach shows better performance in all cases. This exploration into combining different feature sets underscores the substantial impact these new features have on enhancing performance. The findings confirm that incorporating Sprint2Vec features significantly boosts predictive accuracy compared to using other feature set combinations. Additionally, the results suggest a synergistic effect, where combining various feature sets within our approach enhances Sprint2Vec's performance. In conclusion, these outcomes highlight the critical role of feature set combinations as configurable elements in maximizing predictive accuracy for each project's unique context.

RQ4: Do the features derived from Sprint2Vec improve predictive performance compared to the state-of-the-art software engineering feature embedding technique?

Table 5 and 6 show the evaluation results for our proposed predictive tasks: sprint productivity and quality. These results are quantified in terms of Mean Absolute Error (MAE), Mean Squared Error (MSE), and Root Mean Squared Error (RMSE) for both Sprint2Vec (S2V) and seBERT. Additionally, the results include the percentage improvement of MAE (PI), p-values with corresponding significance codes (Sig. Code) from the Wilcoxon tests, and A^{12} effect size. In predicting sprint productivity, the evaluation results show that Sprint2Vec (S2V) consistently outperforms seBERT in most projects across the key metrics of MAE, MSE, and RMSE. In the AP project, S2V demonstrates better performance, with lower MAE (1.041 vs. 1.277), MSE (3.379 vs. 5.066), and RMSE (1.838 vs. 2.251) compared to seBERT, alongside a statistically significant p-value ($<$0.001) and a moderate effect size (A^{12} = 0.543). In the AT project, S2V also shows slightly better MAE (0.467 vs. 0.497), although seBERT marginally outperforms S2V in MSE (0.883 vs. 0.894) and RMSE (0.940 vs. 0.946). The p-value (0.002) indicates a statistically significant difference, with a small effect size (A^{12} = 0.526). In addition, we observed that this case exhibited the lowest variability in outcomes compared to other projects, which may have contributed to the reduced effectiveness of our method. The results for the predicting sprint quality show that Sprint2Vec (S2V) generally outperforms the seBERT approach across most scenarios, particularly when evaluating the Mean Absolute Error (MAE). Specifically, S2V shows superior performance in all five projects (AP, AT, JE, SP, TA) based on MAE. For example, in the JE project, S2V achieves a notably lower MAE of 0.300 compared to seBERT's 0.644, which achieves a significant performance improvement.

RQ5: Does the Sprint2Vec approach outperform existing baseline methods in predicting the number of completed issues and reopened issues at the issue level?

Table 7 and Table 8 show the evaluation results achieved by Sprint2Vec (S2V) and the baselines (Mora [54] and SH [55]) on the two additional predictive tasks: (1) the number of completed issues and (2) the number of reopened issues. We employed Mean Absolute Error (MAE), Mean Squared Error (MSE), and Root Mean Squared Error (RMSE) as the measurements. Additionally, we reported the percentage improvement of MAE (PI), p-values with corresponding significance codes (Sig. Code) from the Wilcoxon tests, and A^{12} effect size.

Table 7 shows the evaluation results for the task of predicting the number of completed issues, comparing Sprint2Vec (S2V) with the approach by Mora et al. [54] across five projects (AP, AT, JE, SP, TA). The results indicate that Sprint2Vec (S2V) generally outperforms the approach proposed by Mora et al. in predicting the number of completed issues across most projects. Specifically, in the AP, AT, and JE projects, S2V shows statistically significant improvements with moderate effect sizes. For instance, in the AP project, S2V achieves a 30.12% improvement in MAE compared to Mora's approach, with a p-value of less than 0.001 and an effect size (A^{12}) of 0.64. Similarly, in the AT and JE projects, S2V demonstrates 36.18% and 38.78%

TABLE 4: Evaluation results of MAE, MSE, and RMSE achieved by Sprint2Vec (S2V), naive baselines (Mean, Rand., LR), existing approach (Ex.), and alternative approaches (ALT (1), ALT (2), ALT (3), ALT (4), ALT (5)), reported with percentage improvement (PI), p-values with corresponding significant codes (Sig. Code) from the Wilcoxon tests, and A^{12} Effect size

Pj.	Appr.	Sprint productivity						Sprint quality					
		MAE	MSE	RMSE	PI	P-value	A^{12}	MAE	MSE	RMSE	PI	P-value	A^{12}
AP	S2V	1.04	3.38	1.84	-	-	-	1.15	4.88	2.21	-	-	-
	Mean	2.03	7.02	2.65	48.64	<0.001 [***]	0.75	1.63	5.43	2.33	29.56	<0.001 [***]	0.64
	Rand.	1.42	5.57	2.36	26.54	<0.001 [***]	0.60	1.22	4.99	2.24	5.77	0.003 [**]	0.54
	LR	1.14	4.03	2.01	8.52	0.024 [*]	0.53	1.24	4.93	2.22	7.13	0.386 [ ]	0.56
	Ex.	1.58	6.23	2.50	33.96	<0.001 [***]	0.60	1.27	5.46	2.34	9.78	<0.001 [***]	0.53
	ALT (1)	1.97	46.74	6.84	47.05	0.023 [*]	0.56	1.26	5.48	2.34	8.77	<0.001 [***]	0.51
	ALT (2)	>100.00	>100.00	>100.00	99.99	<0.001 [***]	0.60	>100.00	>100.00	>100.00	99.81	<0.001 [***]	0.53
	ALT (3)	1.29	5.37	2.32	19.15	<0.001 [***]	0.52	1.28	5.26	2.29	10.13	<0.001 [***]	0.55
	ALT (4)	>100.00	>100.00	>100.00	100.00	<0.001 [***]	0.63	>100.00	>100.00	>100.00	99.98	<0.001 [***]	0.54
	ALT (5)	>100.00	>100.00	>100.00	99.99	<0.001 [***]	0.57	>100.00	>100.00	>100.00	99.97	<0.001 [***]	0.52
AT	S2V	0.47	0.89	0.95	-	-	-	0.44	0.55	0.74	-	-	-
	Mean	1.41	3.36	1.83	66.80	<0.001 [***]	0.82	1.39	3.03	1.74	68.56	<0.001 [***]	0.84
	Rand.	0.95	4.03	2.01	50.68	<0.001 [***]	0.65	0.98	3.88	1.97	55.21	<0.001 [***]	0.67
	LR	0.56	0.97	0.98	16.78	<0.001 [***]	0.61	0.67	0.78	0.89	35.16	<0.001 [***]	0.69
	Ex.	0.48	0.89	0.94	3.09	0.033 [*]	0.52	0.45	0.53	0.73	2.51	0.021 [*]	0.51
	ALT (1)	0.74	1.59	1.26	37.14	<0.001 [***]	0.64	0.48	0.56	0.75	9.04	<0.001 [***]	0.55
	ALT (2)	0.51	0.98	0.99	8.97	<0.001 [***]	0.54	0.51	0.58	0.76	13.50	<0.001 [***]	0.58
	ALT (3)	0.58	0.98	0.99	19.18	<0.001 [***]	0.61	0.45	0.54	0.74	2.14	0.030 [*]	0.51
	ALT (4)	0.52	0.94	0.97	10.77	<0.001 [***]	0.57	0.49	0.58	0.76	10.46	<0.001 [***]	0.56
	ALT (5)	0.51	0.99	0.99	8.75	<0.001 [***]	0.53	0.47	0.55	0.74	6.92	<0.001 [***]	0.54
JE	S2V	1.29	5.54	2.35	-	-	-	0.30	0.27	0.52	-	-	-
	Mean	1.59	7.94	2.82	19.06	<0.001 [***]	0.58	0.51	0.43	0.66	41.03	<0.001 [***]	0.70
	Rand.	1.60	7.87	2.81	19.45	<0.001 [***]	0.60	0.48	0.47	0.69	38.16	<0.001 [***]	0.66
	LR	1.34	6.60	2.57	3.56	0.267 [ ]	0.54	0.47	0.29	0.54	35.54	<0.001 [***]	0.73
	Ex.	>100.00	>100.00	>100.00	100.00	<0.001 [***]	0.61	>100.00	>100.00	>100.00	100.00	<0.001 [***]	0.80
	ALT (1)	1.33	6.51	2.55	3.40	0.926 [ ]	0.47	0.47	0.30	0.55	36.56	<0.001 [***]	0.75
	ALT (2)	1.39	7.41	2.72	7.16	0.330 [ ]	0.49	0.32	0.23	0.48	5.71	<0.001 [***]	0.61
	ALT (3)	>100.00	>100.00	>100.00	99.98	<0.001 [***]	0.56	>100.00	>100.00	>100.00	100.00	<0.001 [***]	0.75
	ALT (4)	1.38	6.20	2.49	6.82	<0.001 [***]	0.54	0.40	0.26	0.51	25.86	<0.001 [***]	0.68
	ALT (5)	1.52	8.08	2.84	15.10	<0.001 [***]	0.53	0.34	0.31	0.56	13.00	0.006 [**]	0.56
SP	S2V	0.72	1.55	1.25	-	-	-	0.16	0.12	0.35	-	-	-
	Mean	3.06	14.13	3.76	76.57	<0.001 [***]	0.85	1.77	4.83	2.20	90.86	<0.001 [***]	0.94
	Rand.	1.77	13.11	3.62	59.62	<0.001 [***]	0.68	1.11	5.89	2.43	85.46	<0.001 [***]	0.81
	LR	2.84	10.15	3.19	74.76	<0.001 [***]	0.93	1.58	3.13	1.77	89.75	<0.001 [***]	0.97
	Ex.	1.42	6.51	2.55	49.52	<0.001 [***]	0.60	0.19	0.14	0.37	16.03	<0.001 [***]	0.57
	ALT (1)	0.86	1.84	1.36	16.44	<0.001 [***]	0.58	0.21	0.14	0.37	22.58	<0.001 [***]	0.64
	ALT (2)	0.75	1.86	1.36	4.68	0.181 [ ]	0.50	0.20	0.13	0.37	17.84	<0.001 [***]	0.56
	ALT (3)	0.77	1.57	1.25	7.22	<0.001 [***]	0.54	0.22	0.15	0.39	25.00	<0.001 [***]	0.58
	ALT (4)	0.85	1.95	1.40	15.67	<0.001 [***]	0.58	0.16	0.13	0.36	-1.72	0.504 [ ]	0.46
	ALT (5)	1.17	2.95	1.72	38.85	<0.001 [***]	0.66	0.19	0.14	0.37	14.63	<0.001 [***]	0.54
TA	S2V	0.38	0.39	0.63	-	-	-	0.35	0.21	0.46	-	-	-
	Mean	3.39	18.08	4.25	88.67	<0.001 [***]	0.92	3.08	15.13	3.89	88.52	<0.001 [***]	0.92
	Rand.	2.39	69.52	8.34	83.94	<0.001 [***]	0.73	2.28	66.34	8.15	84.51	<0.001 [***]	0.75
	LR	1.90	4.17	2.04	79.80	<0.001 [***]	0.94	1.69	3.31	1.82	79.09	<0.001 [***]	0.94
	Ex.	0.78	0.79	0.89	50.61	<0.001 [***]	0.80	0.37	0.21	0.46	3.93	<0.001 [***]	0.54
	ALT (1)	0.40	0.38	0.62	3.40	0.003 [**]	0.55	0.37	0.22	0.46	5.06	0.011 [*]	0.54
	ALT (2)	0.48	0.43	0.66	20.49	<0.001 [***]	0.62	0.38	0.23	0.47	7.73	<0.001 [***]	0.56
	ALT (3)	0.39	0.39	0.63	0.41	0.009 [**]	0.45	0.35	0.21	0.45	-0.80	0.948 [ ]	0.50
	ALT (4)	0.38	0.39	0.63	0.13	0.009 [**]	0.45	0.37	0.21	0.46	3.33	0.004 [**]	0.53
	ALT (5)	0.40	0.40	0.63	3.38	0.022 [*]	0.55	0.37	0.20	0.44	3.24	0.025 [*]	0.55

Sig. codes: < 0.001 (***) 0.01 (**) 0.05 (*) 0.1 (.) 1 ( )

TABLE 5: Evaluation results of MAE, MSE, and RMSE achieved by Sprint2Vec (S2V) and seBERT on the predictive task of the sprint productivity, reported with percentage improvement of MAE (PI), p-values with corresponding significant codes (Sig. Code) from the Wilcoxon tests, and A^{12} Effect size

Pj.	Appr.	Sprint productivity					
		MAE	MSE	RMSE	PI	P-value	A^{12}
AP	S2V	1.04	3.38	1.84	-	-	-
	seBERT	1.28	5.07	2.25	18.47	<0.001 [***]	0.54
AT	S2V	0.47	0.89	0.95	-	-	-
	seBERT	0.50	0.88	0.94	5.94	0.002 [**]	0.53
JE	S2V	1.29	5.54	2.35	-	-	-
	seBERT	1.14	5.17	2.27	-13.32	0.387 [ ]	0.49
SP	S2V	0.72	1.55	1.25	-	-	-
	seBERT	1.11	2.79	1.67	35.68	<0.001 [***]	0.63
TA	S2V	0.38	0.39	0.63	-	-	-
	seBERT	0.48	0.43	0.66	19.77	<0.001 [***]	0.62

Sig. codes: < 0.001 (***) 0.01 (**) 0.05 (*) 0.1 (.) 1 ( )

TABLE 6: Evaluation results of MAE, MSE, and RMSE achieved by Sprint2Vec (S2V) and seBERT on the predictive task of the sprint quality, reported with percentage improvement of MAE (PI), p-values with corresponding significant codes (Sig. Code) from the Wilcoxon tests, and A^{12} Effect size

Pj.	Appr.	Sprint quality					
		MAE	MSE	RMSE	PI	P-value	A^{12}
AP	S2V	1.15	4.88	2.21	-	-	-
	seBERT	1.25	5.60	2.37	7.85	<0.001 [***]	0.49
AT	S2V	0.44	0.55	0.74	-	-	-
	seBERT	0.48	0.54	0.73	8.32	<0.001 [***]	0.56
JE	S2V	0.30	0.27	0.52	-	-	-
	seBERT	0.64	0.60	0.77	53.52	<0.001 [***]	0.80
SP	S2V	0.16	0.12	0.35	-	-	-
	seBERT	0.19	0.14	0.37	12.69	<0.001 [***]	0.57
TA	S2V	0.35	0.21	0.46	-	-	-
	seBERT	0.36	0.21	0.45	1.75	0.155 [ ]	0.52

Sig. codes: < 0.001 (***) 0.01 (**) 0.05 (*) 0.1 (.) 1 ( )

improvements in MAE, respectively, both with statistically significant p-values and moderate effect sizes. However, the SP project is an exception, where Mora's approach slightly outperforms S2V, though the difference is not statistically significant, as indicated by a p-value of 0.999 and an effect size of 0.45. We found that the lack of improvement in the SP project is likely due to its unique data characteristics, which include having the fewest sprints and the highest variability in issues per sprint. In the TA project, the differences between the two approaches are minimal, with S2V showing a small improvement of 6.09% in MAE, but this is not statistically significant, as reflected by a p-value of 0.394 and an effect size of 0.50. Overall, these results suggest that Sprint2Vec (S2V) is a more effective approach for predicting the number of completed issues in most projects.

For the task of predicting the number of reopened issues, the results show that Sprint2Vec (S2V) generally outperforms the approach by Shihab et al. [55]. Specifically, S2V excels over the baseline in four out of five cases across the MAE metric and in all cases in terms of MSE and RMSE. For instance, in the TA project, S2V achieved an MAE of 2.377, an MSE of 14.295, and an RMSE of 3.781, while the baseline recorded values of 3.110, 22.586, and 4.752, respectively. Furthermore, the improvements are statistically significant in three out of five cases based on the Wilcoxon tests, and S2V shows a larger effect size in four out of five cases. However, we observed no improvement in the JE project, where the baseline slightly outperformed S2V. This anomaly can be attributed to the presence of a normal distribution in the outcome for this project, which may make simpler methods more effective. Despite this exception, the overall results indicate that Sprint2Vec offers significant advantages over the baseline approach, especially in terms of reducing prediction errors as reflected in the MAE, MSE, and RMSE metrics.

TABLE 7: Evaluation results of MAE, MSE, and RMSE achieved by Sprint2Vec (S2V) and Mora et al. approach (Mora) on the predictive task of the number of completed issues, reported with percentage improvement of MAE (PI), p-values with corresponding significant codes (Sig. Code) from the Wilcoxon tests, and A^{12} Effect size

Pj.	Appr.	# of completed issues					
		MAE	MSE	RMSE	PI	P-value	A^{12}
AP	S2V	11.02	>100.00	17.42	-	-	-
	Mora	15.77	>100.00	22.06	30.12	<0.001 [***]	0.64
AT	S2V	1.60	5.12	2.26	-	-	-
	Mora	2.50	18.23	4.27	36.18	<0.001 [***]	0.55
JE	S2V	12.66	>100.00	16.30	-	-	-
	Mora	20.68	>100.00	28.60	38.78	<0.001 [***]	0.59
SP	S2V	30.34	>100.00	49.30	-	-	-
	Mora	24.88	>100.00	45.77	-21.96	0.999 [ ]	0.45
TA	S2V	2.28	13.89	3.73	-	-	-
	Mora	2.43	18.19	4.27	6.09	0.394 [ ]	0.50

Sig. codes: < 0.001 (***) 0.01 (**) 0.05 (*) 0.1 (.) 1 ( )

4.8 Feature analysis

We aimed to predict sprint outcomes in terms of productivity and quality using a predictive model based on five feature groups: sprint characteristics, issue details, developer attributes, issue textual descriptions, and developer activity sequences. To understand how features available before the sprint start influence predictions, we applied SHAP (SHapley Additive exPlanations). SHAP's strength provides interpretable insights into each feature's impact on the model's predictions [63]. Specifically, SHAP values are used to explain the output of machine learning models. They show how much each feature contributes to the

TABLE 8: Evaluation results of MAE, MSE, and RMSE achieved by Sprint2Vec (S2V) and Shihab et al. approach (SH) on the predictive task of the number of reopened issues, reported with percentage improvement of MAE (PI), p-values with corresponding significant codes (Sig. Code) from the Wilcoxon tests, and A^{12} Effect size

Pj.	Appr.	# of reopened issues					
		MAE	MSE	RMSE	PI	P-value	A^{12}
AP	S2V	10.42	>100.00	14.56	-	-	-
	SH	11.29	>100.00	16.59	7.75	0.690 [ ]	0.52
AT	S2V	1.29	2.69	1.64	-	-	-
	SH	3.93	36.08	6.01	67.26	<0.001 [***]	0.71
JE	S2V	11.40	>100.00	12.43	-	-	-
	SH	9.99	180.57	13.44	-14.12	0.999 [ ]	0.38
SP	S2V	8.55	>100.00	12.22	-	-	-
	SH	36.40	>100.00	56.63	76.51	<0.001 [***]	0.67
TA	S2V	2.38	14.30	3.78	-	-	-
	SH	3.11	22.59	4.75	23.57	<0.001 [***]	0.57

Sig. codes: < 0.001 (***) 0.01 (**) 0.05 (*) 0.1 (.) 1 ( )

model's final prediction. A positive SHAP value means the feature increases the prediction, while a negative SHAP value means the feature decreases it. The size of the SHAP value indicates how strong the feature's impact is. The larger the value, the more influence it has. SHAP also takes into account how features interact with each other, ensuring a fair distribution of contributions [63], [64]. This approach is well known in not only clarifying the role of features in predicting outcomes but also emphasizing the need for integrating diverse data types to refine predictive accuracy and model transparency [64].

Table 9 shows the average SHAP values for five groups of features. The sprint feature consistently has a strong positive effect on both productivity and quality in most groups (e.g., Apache). The developer feature group also appears across all groups, highlighting its key role in affecting both productivity and quality. However, the impact of textual descriptions and issue attributes varies significantly between projects. Textual descriptions are especially important in Jenkins for quality and in Apache for productivity, showing that the importance of these features changes based on the project. Despite these differences, the collective contribution of these five feature groups enhances the predictive performance of our models, suggesting that different projects may exhibit distinct characteristics, but these features together provide a comprehensive factor correlate with sprint outcomes. We further investigated feature importance by observing the correlational polarity of the features based on SHAP values. Additionally, we applied SHAP to the existing approaches. The results suggest that the existing approach places excessive emphasis on certain features (e.g., strong positive correlation), which may lead to overfitting or a less generalizable model. In contrast, our approach incorporates a combination of both positive and negative contributions from a diverse set of features, leading to more balanced predictions. Notably, our inclusion of issue textual features and developer-related features, which are absent in the existing approach, provides a more comprehensive understanding of the factors influencing issue reopening. This broader feature set and more balanced approach may explain why our model outperforms others. Note that the additional SHAP results are provided in the repository.

TABLE 9: The five groups of features with their normalized average SHAP values

Sprint productivity		Sprint quality	
Apache			
sprint	1.00	sprint	1.00
text	0.91	text	0.45
activity	0.76	activity	0.42
developer	0.25	issue	0.17
issue	0.22	developer	0.12
Atlassian			
sprint	1.00	developer	1.00
developer	0.98	activity	0.93
activity	0.86	text	0.62
text	0.80	sprint	0.61
issue	0.31	issue	0.55
Jenkins			
text	1.00	text	1.00
sprint	0.91	sprint	0.74
activity	0.47	activity	0.40
issue	0.43	issue	0.22
developer	0.20	developer	0.18
Spring			
sprint	1.00	sprint	1.00
activity	0.30	activity	0.33
developer	0.18	developer	0.11
text	0.14	issue	0.05
issue	0.13	text	<0.00
Talendforge			
activity	1.00	activity	1.00
sprint	0.95	text	0.96
text	0.94	sprint	0.75
developer	0.90	developer	0.63
issue	0.78	issue	0.46

Table 10 shows the top ten discriminative features that are highly correlated with sprint outcomes, including their direction of correlation. For clarity and interpretation, we have limited our analysis to only those features that were manually extracted, allowing us to analyze and interpret their correlations with sprint outcomes. The initial observation shows that sprint duration (*s_plan_duration*) has a significant correlation on both productivity and quality, with effects varying positively and negatively. In Apache, longer sprint durations lead to lower productivity but higher quality. In contrast, Talendforge experiences higher productivity but lower quality with extended sprint duration. This may seem surprising since longer sprints are often expected to improve both productivity and quality. However, this effect

TABLE 10: Top 10 most important features with their normalized SHAP values and direction of correlation

Sprint productivity				Sprint quality			
Positive		Negative		Positive		Negative	
Apache							
d_no_distinct_action	0.35	s_plan_duration	1.00	i_priority_Critical	0.25	s_plan_duration	1.00
i_priority_Critical	0.29	i_type_New Feature	0.74	d_no_comment	0.22	i_type_New Feature	0.48
d_developer_activeness	0.20	i_type_Improvement	0.63	d_developer_activeness	0.10	i_priority_Major	0.29
d_most_prefer_type_Bug	0.18	s_no_issue	0.27	s_no_teammember	0.08	i_fog_index	0.16
d_most_prefer_type_New Feature	0.17	i_priority_Blocker	0.18	i_no_change_priority	0.07	i_type_Improvement	0.10
Atlassian							
d_most_prefer_type_Na	1.00	s_no_teammember	0.42	d_most_prefer_type_Na	0.98	i_priority_High	1.00
d_most_prefer_type_Bug	0.35	i_priority_Medium	0.10	i_priority_Low	0.80	i_no_change_priority	0.43
d_developer_activeness	0.28	i_type_Bug	0.09	d_most_prefer_type_Bug	0.66	d_no_comment	0.38
s_no_issue	0.25	s_plan_duration	0.02	d_no_distinct_action	0.39	i_priority_Medium	0.35
d_no_distinct_action	0.18			i_priority_Highest	0.22	i_no_change_description	0.33
Jenkins							
s_no_teammember	0.21	i_type_Improvement	1.00	s_no_issue	0.53	i_type_Improvement	1.00
d_most_prefer_type_Bug	0.04	i_priority_Minor	0.41	i_priority_Minor	0.47	i_priority_Major	0.78
s_no_issue	0.04	i_priority_Major	0.23	i_fog_index	0.41	s_plan_duration	0.74
d_no_distinct_action	0.02	s_plan_duration	0.22	i_no_change_description	0.08	i_no_component	0.18
d_developer_activeness	0.01	i_fog_index	0.03	d_no_distinct_action	0.07	d_most_prefer_type_Improvement	0.16
Spring							
d_most_prefer_type_Task	0.51	s_no_teammember	1.00	s_plan_duration	1.00	i_type_Story	0.06
i_type_Task	0.48	s_no_issue	0.60	s_no_teammember	0.28	i_priority_Minor	0.05
s_plan_duration	0.44	d_most_prefer_type_Bug	0.14	d_no_comment	0.16	d_no_distinct_action	0.05
d_developer_activeness	0.22	i_priority_Trivial	0.11	i_fog_index	0.13	i_no_comments	0.02
d_most_prefer_type_New Feature	0.15	i_priority_Critical	0.11	s_no_issue	0.10		
Talendforge							
s_plan_duration	1.00	i_fog_index	0.57	i_type_Work Item	1.00	d_most_prefer_type_Work Item	0.41
i_no_component	0.45	d_most_prefer_type_Work Item	0.53	s_plan_duration	0.43	i_no_change_fix	0.41
d_most_prefer_type_Bug	0.43	d_most_prefer_type_Na	0.31	i_priority_Minor	0.29	d_most_prefer_type_Backlog Task	0.28
d_most_prefer_type_Backlog Task	0.36	i_no_change_fix	0.27	d_most_prefer_type_Bug	0.23	i_no_component	0.25
i_no_change_description	0.34	d_most_prefer_type_Epic	0.27	d_most_prefer_type_Na	0.12	i_no_comments	0.13

is strongly associated with tasks labeled as New Features or Improvements and those with Major and Critical priorities. It suggests that teams might plan longer sprints for complex, effort-intensive, and critical tasks to ensure quality. It results in fewer tasks completed within a sprint. Thus, balancing sprint duration and committed deliverables is crucial. For example, consider Sprint *Hudi 0.10.1 - 2021/01/03* and Sprint *Apache DataLab release 2.5.2* of Apache, both sprints started with the same number of assigned issues. Sprint *Hudi 0.10.1 - 2021/01/03* had a shorter duration of seven days and achieved high productivity with 33 issues completed. However, it showed low quality, as evidenced by a high number of reopened issues (19). In contrast, Sprint *Apache DataLab release 2.5.2* had a much longer duration of 18 weeks and achieved both high productivity, with 84 issues completed, and high quality, with only six issues reopened. The extended duration of Sprint *Apache DataLab release 2.5.2* likely allowed for more thorough issue resolution, resulting in fewer reopened issues. This emphasizes the need for strategic sprint planning, taking into account the fixed nature of sprint duration as well as the characteristics and criticality of tasks.

The second observation focuses on features representing developer involvement in sprints (*d_no_distinct_action* and *d_developer_activeness*). These features show a positive correlation with productivity in projects such as Apache, Atlassian, and Jenkins. This suggests that higher developer engagement and a variety of actions lead to improved sprint performance. On the other hand, the feature related to the number of team members (*s_no_teammember*) often correlates negatively with sprint outcomes in projects such as Atlassian and Spring. This may indicate that having too many team members can complicate project management, possibly due to increased collaboration overhead.

We also observe that adding new tasks during a sprint may impact the overall productivity and quality outcomes. For example, in Sprint *2021-09* of the Apache project, the sprint began with only three issues but ended with 35

issues completed, indicating high productivity. However, all 35 completed issues were reopened, suggesting very poor quality. Similarly, in Sprint *Lovelace SR1* of the Spring project, the sprint started with two issues and finished with 20 issues completed, but seven of these were reopened, reflecting a notable decline in quality.

Additionally, changes in issue attributes, such as modifications to the issue description or fix version, may reflect uncertainty and unclear task requirements, potentially impacting the productivity and quality of a sprint. For example, in Sprint *DQ21 CN 8* from the Talendforge project, there were 15 issues at the start of the sprint. During the sprint, the descriptions of ten issues were changed, with some undergoing frequent revisions. Notably, Issue *TDQ-19354*[6] and Issue *TDQ-19536*[7] had their descriptions modified up to seven times. Furthermore, some issues experienced multiple changes to their fix version, such as Issue *TDQ-19202*[8], which had its fix version altered ten times. By the end of the sprint, only one issue was completed, indicating very low productivity, and this issue was also reopened later. This suggests that the changes in issue attributes during the sprint likely contributed to poor issue resolution. However, we acknowledge that an extensive qualitative study on the practical adoption of this approach is required.

Overall, the influence of features on projects varies significantly, highlighting the context-dependent nature of what drives productivity and quality in agile projects. This variation underlines the need for the approach that can handle the unique characteristics and requirements of each project. By understanding the specific dynamics of each team and project, more effective sprint planning, improved resource allocation, and enhanced project outcomes can be achieved. To facilitate this, our approach leverages diverse and comprehensive information on sprints, tasks, and developers. We employ both manual feature extraction and automatic feature learning to accurately characterize sprints, enabling predictions that accommodate the wide range of project characteristics. By applying the Sprint2Vec approach in such scenarios, the development team can make informed, data-driven decisions to enhance their Agile processes. For instance, if the productivity prediction indicates that they often fall short of completing committed work, they may need to adjust their planning or workload allocation. On the other hand, if the quality prediction suggests a high reopening issue rate, they can focus on improving their development and testing practices.

Moreover, regarding the adoption of Sprint2Vec in practice, it is important to note that the most resource-intensive aspect of the approach lies in training the developer activity embedding layer. While the text embedding can utilize existing pre-trained models, the activity embedding requires practitioners to collect data and train the model specifically for their context. This step can be computationally expensive and time-consuming. Sprint2Vec is designed to support practitioners by providing insights based on their team's past performance, rather than replacing expert judgment. By adopting this approach, teams can gain valuable information that aids in sprint planning and decision-making, enhancing the effectiveness of their strategies. We also outline a study on the practical adoption of this approach, which aims to provide deeper insights into the real-world challenges and benefits of implementing our method in our future work.

6. https://jira.talendforge.org/browse/TDQ-19354
7. https://jira.talendforge.org/browse/TDQ-19536
8. https://jira.talendforge.org/browse/TDQ-19202

4.9 Threats to validity

External validity: It is crucial to recognize that our findings may not fully represent all types of software projects, particularly those in commercial settings. Factors such as the nature of contributors, developers, and project stakeholders can differ between open-source and commercial projects, potentially impacting the predicted sprint outcomes. While we acknowledge the importance of validating our approach in commercial settings, we must also acknowledge the inherent challenges in accessing commercial project datasets, which often require collaboration with industrial partners. Thus, addressing this limitation is a priority for our future work. In the meantime, we mitigated this concern by conducting our analysis on five repositories that offer a range of sizes, complexities, developer teams, and attributes, thereby providing diverse perspectives.

Internal validity: We utilized real-world data from popular open-source software project repositories, namely Apache, Atlassian, Jenkins, Spring, and Talendforge. This data was obtained from sprint reports, issue reports, and developer activities using Jira APIs. We enhanced transparency by including the data collection script in our replication package. Cross-validation against data available on the Jira website ensured accuracy and consistency. In addition, we utilized information available at the beginning of each sprint, extracting details from the issue change logs. This approach mitigated any potential information leakage or bias that could impact the accuracy and reliability of our results. In data preprocessing, rigorous filters were applied to enhance data quality, including excluding ongoing sprints, those with zero assigned issues, and outliers with excessive issue assignments. Additionally, for textual data and developer activities, we conducted validation by examining the distribution of data characteristics, such as the length of textual descriptions and the number of developer activities. These analyses enabled us to detect any anomalies in our data extraction process and maintain data quality.

Construct validity: It refers to how well our experiment accurately measures the underlying construct it aims to assess. In our study, this involves ensuring that our methods effectively capture the predictive tasks we are investigating. We validate our approach at various stages, including data collection, preprocessing, and cleansing, to construct a reliable dataset for analysis. This involves removing outliers, handling missing values, and ensuring data consistency to minimize the risk of measurement errors or biases. Furthermore, to prevent information leakage and maintain the integrity of our analysis, we take careful steps in data processing. Specifically, we avoid using future information in the prediction process, ensuring that our model's performance is evaluated solely on data available at the time of prediction. This helps uphold the integrity of our experimental design and ensures that the observed effects

are not confounded by temporal dependencies or predictive biases. To assess the quality and reliability of our predictive models, we employ Mean Absolute Error (MAE), which are selected for their ability to capture various aspects of prediction quality and their resilience to data imbalance. By utilizing robust evaluation metrics, we ensure that our models' performance is rigorously assessed and accurately reflects their predictive capabilities. Furthermore, we conduct a Wilcoxon test to validate the robustness of our results and determine the statistical significance of any performance improvements observed. In addition, we apply effect size analysis to complement our evaluation metrics. Effect size provides valuable insight into the practical significance of observed differences, helping us assess the magnitude of the effects beyond statistical significance.

5 Conclusions and Future Work

Sprint planning is a critical component of the Scrum framework as it establishes the groundwork for a successful sprint iteration. Effective planning and risk management in the early phases are crucial for decision-makers to minimize potential problems. In this study, we aim to find the suitable approach for each specific feature category. For instance, unstructured data, such as textual descriptions, are handled by automatic feature learning approaches. However, during the model training layer, these diverse features are consolidated and merged to construct a comprehensive sprint representation. This diverse set of features is then learned together in the neural network model, which is parameterized and trainable, automatically adapting to the diversity among features to predict the sprint outcomes. Our evaluation results have shown that these feature sets complement each other, leading to superior predictive performance in both of our predictive tasks. Additionally, We employ dropout techniques to enhance our model by mitigating overfitting and improving generalization, which is well-supported in existing literature [65], [66]. This allows our methodology to adapt to different team and project dynamics, ensuring that it remains robust and versatile when applied to a wide range of real-world tasks.

To evaluate our approach, we conducted extensive experiments on five well-known open-source software project repositories: Apache, Atlassian, Jenkins, Spring, and Talendforge, utilizing the Jira issue tracking system. The performance of the predictive models trained using the features derived by Sprint2Vec was evaluated. Our experimental results demonstrate that our approach consistently outperforms the baseline, existing approaches, and alternative methods.

For future work, we plan to expand our study to larger repositories and commercial software projects to further validate and enhance the robustness of our approach. We will address the cold start problem by employing transfer learning and cross-project training techniques. Additionally, we aim to explore the impact of time on sprint characteristics by evaluating our approach in a sliding window setting, allowing us to uncover trends and patterns that may not be evident when considering the data as a whole. Regarding feature understanding, while our study has highlighted the varying importance of different features (sprint, text, activity, developer, issue) in characterizing a sprint, there is considerable potential to enhance these findings by exploring more sophisticated methods of combining features. Future research could explore the use of attention mechanisms or dynamic weighting strategies to better capture the interactions between features. Furthermore, we intend to develop our approach as a tool for project management platforms, such as Jira, to support user decision-making during sprint planning processes.

Acknowledgment

This work was financially supported by the Office of the Permanent Secretary, Ministry of Higher Education, Science, Research and Innovation (Grant No. RGNS 64-164).

References

[1] P. Viechnicki and M. Kelkar, "Agile by the numbers," in *Agile in Government: A playbook from the Deloitte Center for Government Insights*. The Deloitte Center, 2017, pp. 42–47.

[2] B. Michael and Others, "Delivering large-scale IT projects on time, on budget, and on value," *McKinsey Company*, no. October 2012, pp. 1–11, 2012.

[3] E. Altameem, "Impact of Agile Methodology on Software Development," *Computer and Information Science*, vol. 8, no. 2, pp. 9–14, 2015.

[4] J. M. Sanchez, "Challenges of Traditional and Agile Software Processes," in *Proceedings of the 12th Americas Conference on Information Systems (AMCIS)*, no. March. AIS Electronic Library, Nov 2006, pp. 4–8.

[5] A. Nugroho and M. R. Chaudron, "A Survey of the practice of design - Code correspondence amongst professional software engineers," in *Proceedings of the 1st International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE, 2007, pp. 467–469.

[6] P. Kruchten, R. L. Nord, and I. Ozkaya, "Technical Debt : From Metaphor," *IEEE Software*, vol. 29, no. 6, pp. 18–22, 2012.

[7] M. Choetkiertikul, H. K. Dam, T. Tran, A. Ghose, and J. Grundy, "Predicting Delivery Capability in Iterative Software Development," *IEEE Transactions on Software Engineering*, vol. 44, no. 6, pp. 551–573, Jun 2018.

[8] F. Kortum, J. Klünder, W. Brunotte, and K. Schneider, "Sprint Performance Forecasts in Agile Software Development - The Effect of Futurespectives on Team-Driven Dynamics," in *Proceedings of the International Conference on Software Engineering and Knowledge Engineering (SEKE)*, vol. 2019-July, no. July. ACM, 2019, pp. 94–101.

[9] M. Choetkiertikul, H. K. Dam, T. Tran, and A. Ghose, "Characterization and Prediction of Issue-Related Risks in Software Projects," in *Proceedings of the IEEE/ACM 12th Working Conference on Mining Software Repositories (MSR)*, vol. 2015-Augus. IEEE, May 2015, pp. 280–291.

[10] ——, "Predicting the delay of issues with due dates in software projects," *Empirical Software Engineering*, vol. 22, no. 3, pp. 1223–1263, 2017.

[11] Y. Lee, S. Lee, C.-G. Lee, I. Yeom, and H. Woo, "Continual Prediction of Bug-Fix Time Using Deep Learning-Based Activity Stream Embedding," *IEEE Access*, vol. 8, no. 6, pp. 10 503–10 515, 2020.

[12] W. H. A. Al-Zubaidi, H. K. Dam, M. Choetkiertikul, and A. Ghose, "Multi-Objective Iteration Planning in Agile Development," in *Proceedings of the 25th Asia-Pacific Software Engineering Conference (APSEC)*, vol. 2018-Decem. IEEE, 2018, pp. 484–493.

[13] M. A. Ramessur and S. D. Nagowah, "A predictive model to estimate effort in a sprint using machine learning techniques," *International Journal of Information Technology*, vol. 13, no. 3, pp. 1101–1110, 2021.

[14] [Online]. Available: https://github.com/MUICT-SERU/sprint2vec_revision

[15] P. L. Braga, A. L. I. Oliveira, and S. R. L. Meira, "Software Effort Estimation using Machine Learning Techniques with Robust Confidence Intervals," in *Proceedings of the 7th International Conference on Hybrid Intelligent Systems (HIS)*. IEEE, Sep 2008, pp. 352–357.

[16] G. Azizyan, M. K. Magarian, and M. Kajko-Mattson, "Survey of agile tool usage and needs," in *Proceedings of the Agile Conference (Agile)*. IEEE, Nov 2011, pp. 29–38.

[17] Q. Mi and J. Keung, "An empirical analysis of reopened bugs based on open source projects," in *Proceedings of the ACM International Conference Proceeding Series (ICPS)*, vol. 01-03-June. ACM, Jun 2016.

[18] T. Zimmermann, N. Nagappan, P. J. Guo, and B. Murphy, "Characterizing and predicting which bugs get reopened," in *Proceedings of the 34th International Conference on Software Engineering (ICSE)*, vol. 02-Jun. IEEE, Jun 2012, pp. 1074–1083.

[19] C. Mathieu and P. Jose, "Agile Team Maturity and Delivered Story Points Estimation Model: a Case Study," in *Proceedings of the International Conference on Evaluation and Assessment in Software Engineering (EASE)*. ACM, 2016.

[20] M. Fu and C. Tantithamthavorn, "GPT2SP: A Transformer-Based Agile Story Point Estimation Approach," *IEEE Transactions on Software Engineering*, vol. 49, no. 2, pp. 611–625, 2023.

[21] H. Phan and A. Jannesari, "Heterogeneous graph neural networks for software effort estimation," in *Proceedings of the 16th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, ser. ESEM '22. New York, NY, USA: ACM, 2022, pp. 103–113.

[22] Y. Bengio, A. Courville, and P. Vincent, "Representation Learning: A Review and New Perspectives," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1798–1828, Aug 2013.

[23] Z. S. Harris, "Distributional Structure," *WORD*, vol. 10, no. 2-3, pp. 146–162, 1954.

[24] T. Mikolov and Others, "Distributed representations of words and phrases and their compositionality," *Advances in Neural Information Processing Systems*, vol. 26, pp. 1–9, 2013.

[25] K. S. Jones, "A Statistical Interpretation of Term Specificity and Its Application in Retrieval," *Journal of Documentation*, vol. 28, no. 1, pp. 11–21, 1972.

[26] S. Robertson, "Understanding Inverse Document Frequency: On Theoretical Arguments for IDF," *Journal of Documentation (J DOC)*, vol. 60, no. 5, pp. 503–520, 2004.

[27] J. Suntoro, "Data Mining," in *Mining of Massive Datasets*, 2nd ed. Cambridge University Press, 2005, vol. 2, no. January 2013, pp. 5–20.

[28] E. Altszyler, S. Ribeiro, M. Sigman, and D. Fernández Slezak, "The interpretation of dream meaning: Resolving ambiguity using Latent Semantic Analysis in a small corpus of text," *Consciousness and Cognition*, vol. 56, pp. 178–187, Nov 2017.

[29] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[30] M. Browne and S. S. Ghidary, "Convolutional neural networks for image processing: An application in robot vision," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 2903, 2003, pp. 641–652.

[31] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov 1997.

[32] U. Singh, S. Chauhan, A. Krishnamachari, and L. Vig, "Ensemble of deep long short term memory networks for labelling origin of replication sequences," in *Proceedings of the IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, vol. Oct-2015. IEEE, Oct 2015, pp. 1–7.

[33] R. Pascanu, C. Gulcehre, K. Cho, and Y. Bengio, "How to construct deep recurrent neural networks," in *Proceedings of the 2nd International Conference on Learning Representations (ICLR)*. ICLR Press, 2014.

[34] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *Advances in Neural Information Processing Systems*, vol. 4, no. January, pp. 3104–3112, 2014.

[35] T. Han, K. Hao, Y. Ding, and X. Tang, "A new multilayer LSTM method of reconstruction for compressed sensing in acquiring human pressure data," in *Proceedings of the Asian Control Conference (ASCC)*, vol. 2018-Janua. IEEE, Dec 2018, pp. 2001–2006.

[36] A. G. Salman, Y. Heryadi, E. Abdurahman, and W. Suparta, "Single Layer and Multi-layer Long Short-Term Memory (LSTM) Model with Intermediate Variables for Weather Forecasting," *Procedia Computer Science*, vol. 135, pp. 89–98, 2018.

[37] W. DuBay, "The principles of readability," in *Costa Mesa: Impact Information*. ERIC Clearinghouse, 2008, no. 949, p. 77.

[38] M. Choetkiertikul, H. K. Dam, T. Tran, T. Pham, A. Ghose, and T. Menzies, "A Deep Learning Model for Estimating Story Points," *IEEE Transactions on Software Engineering*, vol. 45, no. 7, pp. 637–656, Jul 2019.

[39] R. N. Waykole and A. D. Thakare, "a Review of Feature Extraction Methods for Text Classification," *International Journal of Advance Engineering and Research Development*, vol. 5, no. 04, pp. 351–354, 2018.

[40] V. Efstathiou, C. Chatzilenas, and D. Spinellis, "Word embeddings for the software engineering domain," in *Proceedings of the 15th International Conference on Mining Software Repositories (MSR)*. New York, NY, USA: ACM, May 2018, pp. 38–41.

[41] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. Bowman, "GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding," in *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, no. 11. Brussels, Belgium: ACL, Nov 2018, pp. 353–355.

[42] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, "SQuAD: 100,000+ Questions for Machine Comprehension of Text," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, no. 11. Austin, Texas: ACL, Nov 2016, pp. 2383–2392.

[43] R. Zellers, Y. Bisk, R. Schwartz, and Y. Choi, "SWAG: A Large-Scale Adversarial Dataset for Grounded Commonsense Inference," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Brussels, Belgium: ACL, 2018, pp. 93–104.

[44] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, vol. 1. Minneapolis, Minnesota: ACL, Jun 2019, pp. 4171–4186.

[45] J. Tabassum, M. Maddela, W. Xu, and A. Ritter, "Code and Named Entity Recognition in StackOverflow," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*. Stroudsburg, PA, USA: ACL, May 2020, pp. 4913–4926.

[46] J. von der Mosel, A. Trautsch, and S. Herbold, "On the validity of pre-trained transformers for natural language processing in the software engineering domain," *Lecture Notes in Informatics (LNI), Proceedings - Series of the Gesellschaft fur Informatik (GI)*, vol. P-332, no. 8, pp. 93–94, 2023.

[47] Y. Wu *et al.*, "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation," *CoRR*, Sep 2016.

[48] E. M. De Bortoli Fávero, D. Casanova, and A. R. Pimentel, "EmbSE: A Word Embeddings Model Oriented Towards Software Engineering Domain," in *Proceedings of the XXXIII Brazilian Symposium on Software Engineering*, no. July. New York, NY, USA: ACM, Sep 2019, pp. 172–180.

[49] R. Salman, A. Alzaatreh, and H. Sulieman, "The stability of different aggregation techniques in ensemble feature selection," *Journal of Big Data*, vol. 9, pp. 1–23, 12 2022.

[50] F. Sarro, A. Petrozziello, and M. Harman, "Multi-objective software effort estimation," in *Proceedings of the 38th International Conference on Software Engineering (ICSE)*, vol. 14-22-May-2016. New York, New York, USA: ACM Press, 2016, pp. 619–630.

[51] Ijlemr, "Prediction of Software Defects Using Zero R," *International Journal of Latest Engineering and Management Research (IJLEMR)*, vol. 02, no. 12, 2017.

[52] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning*, 1st ed., ser. Springer Texts in Statistics. New York, NY: Springer New York, 2013, vol. 103.

[53] A. Trautsch and S. Herbold, *Predicting Issue Types with seBERT*. Association for Computing Machinery, 2022, vol. 1, no. 1.

[54] S. L. Ramírez-Mora, H. Oktaba, and H. Gómez-Adorno, "Descriptions of issues and comments for predicting issue success in software projects," *Journal of Systems and Software*, vol. 168, p. 110663, 2020.

[55] E. Shihab, A. Ihara, Y. Kamei, W. M. Ibrahim, M. Ohira, B. Adams, A. E. Hassan, and K. I. Matsumoto, "Studying re-opened bugs in open source software," *Empirical Software Engineering*, vol. 18, no. 5, pp. 1005–1042, 2013.

[56] P. J. Wozniak, "Applied Nonparametric Statistics (2nd ed.)," *Technometrics*, vol. 33, no. 3, pp. 364–365, 1991.

[57] A. Vargha and H. D. Delaney, "A critique and improvement of the CL common language effect size statistics of McGraw and Wong,"

Journal of Educational and Behavioral Statistics, vol. 25, no. 2, pp. 101–132, 2000.

[58] G. Neumann, M. Harman, and S. Poulding, "Transformed vargha-delaney effect size," in *Proceedings of the Search-Based Software Engineering (SSBSE)*, M. Barros and Y. Labiche, Eds. Cham: Springer International Publishing, 2015, pp. 318–324.

[59] M. Shepperd and S. MacDonell, "Evaluating prediction systems in software project estimation," *Information and Software Technology*, vol. 54, no. 8, pp. 820–827, 2012.

[60] F. Palma, T. Abdou, A. Bener, J. Maidens, and S. Liu, "An improvement to test case failure prediction in the context of test case prioritization," in *Proceedings of the 14th International Conference on Predictive Models and Data Analytics in Software Engineering (PROMISE)*, ser. PROMISE'18, vol. 10-Oct. New York, NY, USA: ACM, 2018, p. 80–89.

[61] A. N. Spiess and N. Neumeyer, "An evaluation of R2as an inadequate measure for nonlinear models in pharmacological and biochemical research: A Monte Carlo approach," *BMC Pharmacology*, vol. 10, pp. 1–11, 2010.

[62] F. Jelinek, R. L. Mercer, L. R. Bahl, and J. K. Baker, "Perplexity—a measure of the difficulty of speech recognition tasks," *The Journal of the Acoustical Society of America*, vol. 62, no. S1, pp. S63–S63, Dec 1977.

[63] S. M. Lundberg and S. I. Lee, "A unified approach to interpreting model predictions," in *Proceedings of the Advances in Neural Information Processing Systems*, vol. 2017-December, no. December. MIT Press, 2017, pp. 4766–4775.

[64] Y. Al-Smadi, M. Eshtay, A. Al-Qerem, S. Nashwan, O. Ouda, and A. A. Abd El-Aziz, "Reliable prediction of software defects using Shapley interpretable machine learning models," *Egyptian Informatics Journal*, vol. 24, no. 3, p. 100386, 2023.

[65] P. Baldi and P. J. Sadowski, "Understanding dropout," *Advances in neural information processing systems*, vol. 26, 2013.

[66] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.