A Tale of Two Tasks: Automated Issue Priority Prediction with Deep Multi-task Learning

Yingling Li*
The Key Laboratory for Computer Systems of State Ethnic Affairs Commission, Southwest Minzu University, China
80300053@swun.edu.cn

Xing Che*
University of Chinese Academy of Sciences; Institute of Software Chinese Academy of Sciences; Sichuan University, China
24744873@qq.com

Yuekai Huang
University of Chinese Academy of Sciences; Institute of Software Chinese Academy of Sciences, China
huangyuekai18@mails.ucas.ac.cn

Junjie Wang†
Institute of Software Chinese Academy of Sciences, Beijing
junjie@iscas.ac.cn

Song Wang
York University, Canada
wangsong@yorku.ca

Yawen Wang
Institute of Software Chinese Academy of Sciences, China
yawen2018@iscas.ac.cn

Qing Wang†
Institute of Software Chinese Academy of Sciences, China
wq@iscas.ac.cn

ABSTRACT

Background. Issues are prevalent, and identifying the correct priority of the reported issues is crucial to reduce the maintenance effort and ensure higher software quality. There are several approaches for the automatic priority prediction, yet they do not fully utilize the related information that might influence the priority assignment. Our observation reveals that there are noticeable correlations between an issue's priority and its category, e.g., an issue of bug category tends to be assigned with higher priority than an issue of document category. This correlation motivates us to employ multi-task learning to share the knowledge about issue's category prediction and facilitating priority prediction.

Aims. This paper aims at providing an automatic approach for effective issue's priority prediction, to reduce the burden of the project members and better manage the issues.

Method. We propose issue priority prediction approach PRIMA with deep multi-task learning, which takes the issue category prediction as another task to facilitate the information sharing and learning. It consists of three main phases: 1) data preparation and augmentation phase, which allows data sharing beyond single task learning; 2) model construction phase, which designs shared layers to encode the semantics of textual descriptions, and task-specific layers to model two tasks in parallel; it also includes the indicative attributes to better capture an issue's inherent meaning; 3) model training phase, which enables eavesdropping by shared loss function between two tasks.

Results. Evaluations with four large-scale open-source projects show that PRIMA outperforms commonly-used and state-of-the-art baselines, with 32% -55% higher precision, and 28% - 56% higher recall. Compared with single task learning, the performance improvement reaches 18% in precision and 19% in recall. Results from our user study further prove its potential practical value.

Conclusions. The proposed approach provides a novel and effective way for issue priority prediction, and sheds light on jointly exploring other issue-management tasks.

CCS CONCEPTS

• **Software and its engineering** → **Software testing and debugging**.

ACM Reference Format:
Yingling Li, Xing Che, Yuekai Huang, Junjie Wang, Song Wang, Yawen Wang, and Qing Wang. 2022. A Tale of Two Tasks: Automated Issue Priority Prediction with Deep Multi-task Learning. In *ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM) (ESEM '22), September 19–23, 2022, Helsinki, Finland*. ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3544902.3546257

1 INTRODUCTION

Issues can emerge at any time during software development, while effectively managing and fixing issues are critical for ensuring higher software quality. Yet, the issues can vary considerably as some of them are critical enough to require immediate action, whereas others are minor and can be handled later. Identifying the correct priority of the reported issues is important in directing the software corrective maintenance effort and contributing to creating more stable software systems [1, 7, 27, 29, 38].

Assigning the issue priority is a time-consuming and non-trivial task, considering the large number of issues submitted in a software

*Co-first author
†Corresponding author

ESEM '22, September 19–23, 2022, Helsinki, Finland

ACM ISBN 978-1-4503-9427-7/22/09.
https://doi.org/10.1145/3544902.3546257

product, the complexity of the issue description, etc. Taking Kubernetes (an open-source system with 87.6k stars for managing containerized applications across multiple hosts) as an example, there are 8,480 issue reports submitted from 04/01/2021 to 04/01/2022, which means that this project can receive more than 23 new issue reports a day on average. In addition, each issue report has approximately 945 terms (for the Kubernetes project) mixed with code examples and stack traces in describing the encouraged defects. We also observe that for an issue report, it takes an average of 79 days to firstly assign the priority in an open-source project as Kubernetes. More than that, the priority of an issue report can also change in 35% of circumstances after the first assignment, due to the inappropriate priority assignment. Taken in this sense, automated priority prediction would be of great value.

Previous studies have proposed various approaches for the automated priority prediction [7, 26, 27, 29, 33]. Tian et al. [26, 27] extracted features from six dimensions, i.e., temporal, textual, author, related-report, severity, and product, and built a machine learning model for priority prediction. Umer et al. [29] utilized convolutional neural network to implement an automated priority assignment model, which can eliminate the efforts for feature definition. Fang et al. [7] proposed to utilize graph convolutional networks to better model the term's semantics for priority prediction.

Nevertheless, these aforementioned approaches either directly utilize the extracted features for machine learning, or only employ the textual information for deep learning, both of which do not fully utilize the related information that might influence the priority assignment. Our observations from real-world open-source projects reveal that there are some noticeable correlations between an issue's priority and its other attributes such as report category, e.g., an issue with *bug* category tends to be assigned with a higher priority than the issue with *documentation* category. Specifically, an issue report of *bug* category has a correlation of 0.34 with the priority *high*, while the correlation between *document* category and *high* priority is only 0.02 for project *amphtml* (a web component framework). These correlations could help take better advantage of the training data and boost the prediction performance.

Motivated by the correlation, this study proposes to adopt deep multi-task learning techniques, in which multiple learning tasks are solved at the same time, while exploiting both commonalities and differences across tasks, thereby increasing efficiency and prediction accuracy.

This paper proposes an effective issue PRIority prediction approach with deep Multi-tAsk learning named *PRIMA*, which takes the issue category prediction as the second task to facilitate the learning of priority. It consists of three main phrases, (1) data preparation and augmentation phase, for allowing data sharing beyond single task learning; (2) model construction phase, for constructing the multi-task model for priority prediction and category prediction tasks; (3) model training phase, for enabling eavesdropping by shared loss function between two related tasks. Furthermore, during model construction, we design word embedding and context embedding layers to better encode the issue textual descriptions, and include the indicative attributes extracted from historical issue reports to better model the issue's semantics and boost the performance.

Evaluation results with four large-scale open-source projects show that PRIMA outperforms two commonly-used and state-of-the-art baselines, with the precision of 73% (32% - 55% higher than baselines) and recall of 72% (28% - 56% higher than baselines). Specifically, by jointly learning the two tasks, the performance of PRIMA increases precision by 18% and recall by 19% compared with single task learning. The inclusion of indicative attributes in the model boosts the precision by 8% and recall by 7%. Furthermore, we conduct a field evaluation on the newly-reported issues on GitHub to evaluate its potential practicability in real-world practice. Specifically, we run our prediction model on 18 newly-reported issues, and then send the predicted priority label through issue's comment to developers for confirmation. Among the 8 responses, 6 issues are confirmed as correct, which further proves the potential practical value of this work.

The proposed approach provides an effective way to jointly learn the above two related tasks, and it can also be adopted/tailored to other issue report management tasks to better support the issue exploration and quality improvement.

The main contributions of this paper are as follows:

- The investigation of the correlation between issue priority prediction and category prediction tasks, which motivates the application of the deep multi-task learning in advancing issue priority prediction.
- PRIMA—a deep multi-task learning approach with shared layers for sharing information and task-specific layers for jointly learning priority prediction and category prediction tasks. PRIMA also incorporates indicative attributes together with textual descriptions in the model.
- The evaluation of PRIMA on 14,682 issue reports from four large-scale open-source projects and user study in real-world practice, with affirmative results.
- Publicly accessible dataset for replication and source code to facilitate applications in other scenarios[1].

2 BACKGROUND AND MOTIVATION

2.1 Background

Multi-task Learning (MTL) is an approach that can improve the generalization of the main task by sharing the domain-specific information contained in the training signals of related tasks [4]. Many recent deep learning approaches have used MTL either explicitly or implicitly as part of their models. For example, MTL has been applied successfully in natural language processing (NLP), speech recognition to computer vision [5, 21, 36].

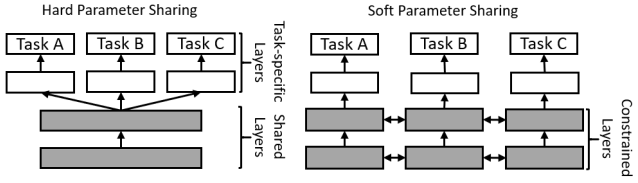


Figure 1: Two typical approaches of MTL.

[1]https://github.com/piexpe/Multi-task_priority

Table 1: Current status of priority assignment

Project	#days(first assign)	%changed	#days(final assign)
kubernetes	14.52	7.53	22.62
minikube	26.90	24.65	65.90
zephyr	48.36	7.17	51.70
amphtml	38.83	17.09	73.36

So far, there are two most typical ways to perform MTL in the context of deep learning: hard or soft parameter sharing of hidden layers [22], as shown in Figure 1. Hard parameter sharing [3] is the most commonly-used MTL paradigm in deep neural networks. This paradigm consists of two parts in constructing the network, i.e., shared layers and task-specific layers. Through the shared layers, the generally hidden vectors for input could be obtained. The hidden vectors are used as the input of the task-specific layers for many related tasks. After that, the shared information among tasks is learned by optimizing model parameters using the joint loss function. This structure can greatly reduce the risk of over-fitting. In the soft parameter sharing paradigm, each task has its own model with its own parameters, where the distance between the parameters of the model is regularized by regularization techniques [6, 35] to make the parameters similar. In our context, since these two tasks share the same inputs, (i.e., the review issue descriptions), they are naturally suitable for the hard parameter sharing, (i.e., sharing the same representation layer and processing multiple tasks in the task-specific layers). Therefore, we choose the hard parameter sharing for jointly learning.

2.2 Observations about Priority Assignment in Real-world OSS Projects

Based on the issue reports of four large-scale open-source projects (as shown in Section 4.2 and Table 4), we derive the following observations which motivate this study.

2.2.1 ***Dilemma of Priority Assignment.*** There are a large number of issue reports submitted in an open-source project each day, especially for the large projects. In real-world practice, assigning the priority for the emerging issue reports is a time-consuming and non-trivial task [7, 26, 27, 29].

As shown in Table 1, it takes an average of 26 days for a report to be assigned first priority in *minikube* project, while this number is 38 for issues in *amphtml* project. Furthermore, in 24% and 17% cases, an issue report's priority changes in these two projects, and it takes an average of 65 and 73 days for an issue report to be assigned the correct priority for these two projects respectively. This implies the inefficiency of current practice of priority allocation and indicates the need for the automatic support.

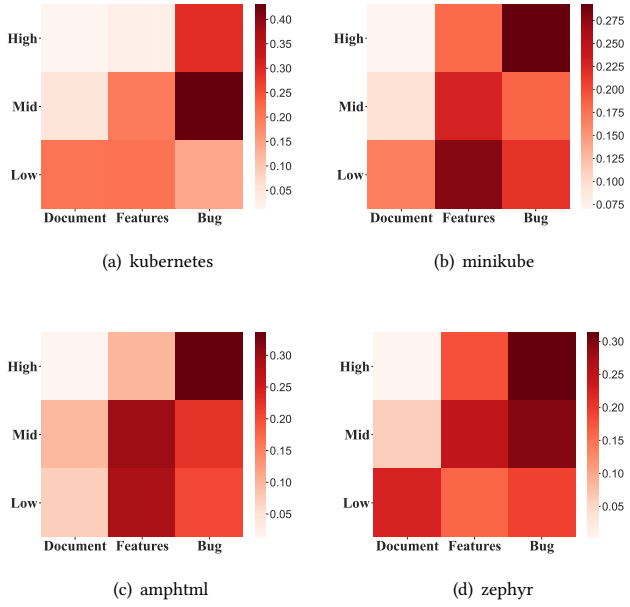
2.2.2 ***Challenges in Priority Assignment.*** To accurately and completely describe the encountered issues, an issue report is generally long and usually has code examples, stack traces, etc.

As shown in Table 2, each issue report is described with an average of 149 terms in *kubernetes* project, and the number of terms is 1,091 for *minikube* project. Meanwhile, a typical issue report would contain one or two code examples (i.e., we roughly count the code examples with *<code>* tag), described with an average of 29 to 600 terms.

This exerts great challenges for assigning correct priority for an issue, e.g., one needs to read through the whole description, speculate the possible influence scope, assess other reports' status, etc. Meanwhile, it also suggests the practical need for understanding the semantics of the descriptions for better priority assignment.

Table 2: Statistics of issue reports

Project	#terms	#<code>	#terms in <code>
kubernetes	149.48	1.89	60.51
minikube	1091.13	2.93	600.81
zephyr	260.26	1.58	138.85
amphtml	116.9	2.02	29.56



(a) kubernetes (b) minikube

(c) amphtml (d) zephyr

Figure 2: Correlation between issue's priority and its category

2.2.3 ***Correlation between An Issue's Priority and Its Category.*** Besides the textual descriptions of an issue report, we assume there might be other information that potentially implies the importance of the issue. For example, issue reports describing bugs might be prioritized higher than the issue reports for the feature requests. Taken in this sense, we investigate the correlation between issue's priority and its category, as shown in Figure 2. Note that, in this study, we focus on the top three most commonly used categories, i.e., bug, feature, and document. We can see that, in three of the

four projects, bug category is most correlated with the high priority. And in all four projects, document category is least correlated with the high priority, while has a relatively large correlation with the lowest priority. We believe the correlation information can serve as important clues for the priority assignment.

In summary, the first and second observations reveal that priority assignment is time-consuming and difficult, which motivates the need for the automatic approach for priority assignment. It also suggests the need for the effective semantic modeling and understanding so that the inherent meaning can be precisely captured. The third observation implies that the priority and issue category are correlated to some extent, which motivates us to employ multi-task learning for the priority prediction.

3 APPROACH

Motivated by the observations in Section 2.2, we propose an issue PRIority prediction approach with deep Multi-tAsk learning named **PRIMA**, which conducts the priority prediction and category prediction simultaneously.

As shown in Figure 3, the proposed approach, PRIMA, consists of three main phases: (1) data preparation and augmentation phase, for data preparation and augmentation to allow data sharing beyond single task learning; (2) model construction phase, for constructing the multi-task model for priority predication and category predication tasks; and (3) model training phase, enabling eavesdropping by shared loss function between the two related tasks. During model construction, we design word embedding and context embedding layers to better encode the issue textual descriptions, and include the indicative attributes extracted from historical issue reports to better model issue's semantics and boost the performance. We will present details on each of the three phases next.

3.1 Data Preparation and Augmentation

In this step, each issue report will be processed, and introduced with augmented information beyond the scope of single task learning.

3.1.1 ***Data Pre-processing.*** The raw issue reports are generally described with textual descriptions mixed with console messages, URLs, etc. Inappropriate processing may lead to the missing of useful description, or the introduction of the noise information. We first combine the title and description as the new descriptions. Then we use a regular expression to replace the URL with the token *<url>*, in order to help the model unify its understanding. Third, we tokenize the descriptions with ark-twokenize[2], which helps extract more useful information from the descriptions, e.g., splitting the method name following camel naming convention, matching the operator for better splitting the code statement. Fourth, we apply data cleaning for the special character removal and lowercase conversion.

3.1.2 ***Indicative Attributes Extraction.*** Since previous work revealed that certain factors such as the authors of an issue can indicate the issue's priority [27]. Following the existing study, we extract 17 features involving three dimensions, i.e., *temporal, author, and related-report* (as shown in Table 3), from the issue reports to add to the learning model.

[2]https://github.com/myleott/ark-twokenize-py

For the temporal dimension, the features measure the time-related tendency of priority assignment, i.e., the number of issue reports that are reported with the highest priority in the last x day(s). We vary the values of x, i.e., 1 day, 3 days, 7 days and 30 days, to get four features. Intuitively, there are usually a large number of critical issues submitted after the release of new versions, and during the period, a newly reported issue is more likely to be assigned with a higher priority.

For the author dimension, we capture the average and median priority of the issue reports, and the total number of issue reports reported by the author before this report. Intuitively, if an author always reports high priority issues, she/he might continue reporting high priority issues.

For the related-report dimension, we capture the average and median priority of the top-k most similar issue reports. We vary the value k to create features 8-17. Considering that the similar issue reports might be assigned the same priority, we employ them to help decide the issue's priority. To measure the similarity of two issue reports, we train a word2vec [20] model with all the experimental issue reports in training data. Then we represent the issue report as the average of all the individual word vectors of its contained terms, then the similarity between two reports is measured using the cosine distance of the two vectors.

3.1.3 ***Data Label Sharing.*** Designed based on supervised learning, each issue report comes with the ground-truth labels for the priority prediction and category classification tasks respectively. At the outset, PRIMA allows sharing labels across related tasks. Thus, the issue report is explicitly augmented in the following two aspects.

Augmented priority prediction data: by sharing labels, the data used to train the priority prediction task layers is automatically augmented by including additional information on issue's categories. The original priority prediction data are the issue reports themselves, and the augmented data are the reports with their issue categories.

Augmented category classification data: similarly, the data used to train category classification task layers is augmented by adding the priority of the issue report.

3.2 Model Construction

The model consists of two parts, i.e., the shared layers and two task-oriented layers, with details in Figure 4. The shared layers are to encode the issue reports, where we employ the word embedding layer and context embedding layer to better capture the semantics and contextual information of the issue. Then we solve two tasks with task-oriented layers respectively. Furthermore, we fuse the indicative attributes extracted from the past issue reports in our model to further boost the priority prediction.

3.2.1 ***Shared Layers Learning.*** To better capture the semantic meaning of the issue report, PRIMA designs two types of shared layers, i.e., word embedding layer, which encodes each token in the input into a semantic vector; context embedding layer, which introduces the contextual information of the input in the representation learning and produces the contextual semantic vector for each input token.

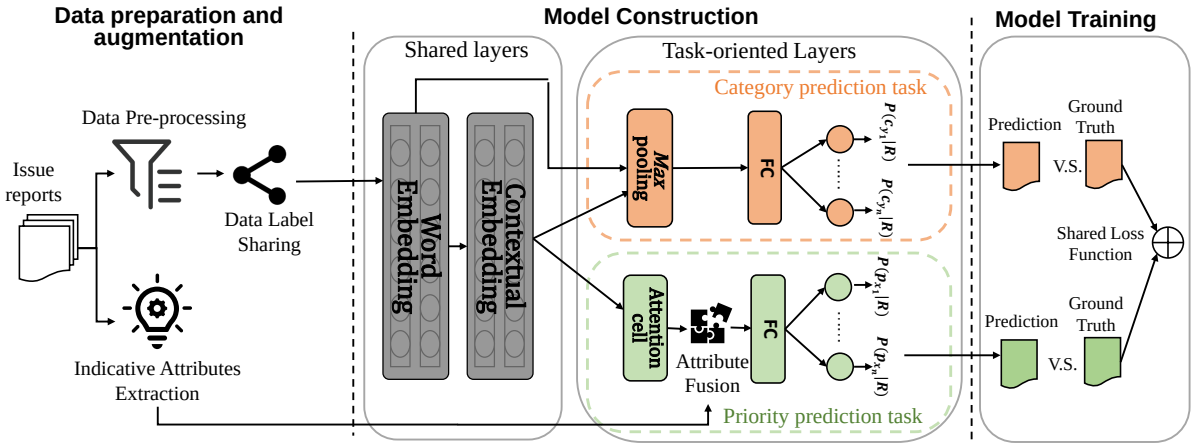


Figure 3: Overview of PRIMA

Table 3: Overview of features

Dimension	Feature	Description
Temporal	1 - 4	Number of issue reports assigned with the highest priority within 1 / 3 / 7 / 30 days before this report
Author	5, 6	Average / Median priority of issue reports reported by this author before this report
	7	Total number of issue reports reported by this author before this report
Related-Report	8, 9	Average / Median priority of the top-1 most similar issue reports with this report
	10, 11	Average / Median priority of the top-3 most similar issue reports with this report
	12, 13	Average / Median priority of the top-5 most similar issue reports with this report
	14, 15	Average / Median priority of the top-10 most similar issue reports with this report
	16, 17	Average / Median priority of the top-20 most similar issue reports with this report

Word embedding layer. The purpose of this layer is to encode each token in the issue report as a semantic vector. PRIMA employs a domain-specific pre-train BERT model [25] to encode the tokens in issue reports. It has 12 layers, 768 hidden dimensions and 12 attention heads. It is pre-trained on 152 million sentences from Stack Overflow, and has advantages in capturing the inherent meaning of terms in the software engineering domain. At the input, each report is represented by 512 word tokens with a special starting symbol [CLS]. For those that are not long enough, we use a special symbol [PAD] to align to the length of 512. For those who are longer, we truncate the excess sequence following common practice [9]. This layer will output the semantic vector (768 dimensions, by default in BERT) for each input token.

Context embedding layer. This layer is designed to capture contextual information between terms and represent each input token as the final semantic vector. We use the Bi-LSTM network structure which is better at capturing the long distance dependency and global semantic information. It converts the output of the semantic vector from the word embedding layer into corresponding contextual semantic vectors, and will be input into the subsequent task-oriented layers.

3.2.2 ***Task-oriented Layers Learning.*** As shown in Figure 4, PRIMA separately conducts *priority prediction* task and *category prediction* task learning.

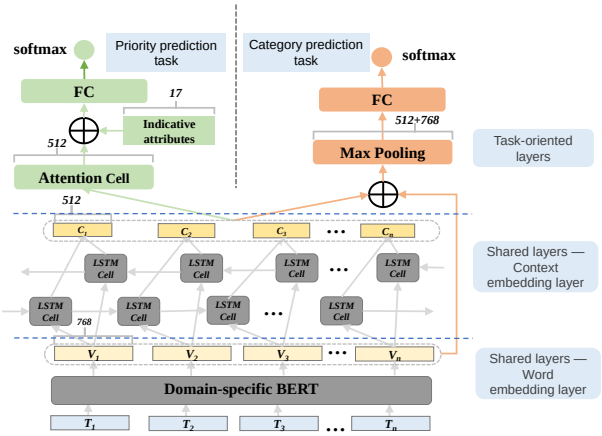


Figure 4: Architecture of our multi-task model

Priority prediction task layers. As we mentioned above, the contextual semantic vector obtained from shared layers is fed into priority prediction task layers. The dimension of the representation is $n \times 512$, where n is the length of the input, and 512 is the dimension of each contextual semantic vector (denoted as c_i) corresponding

to each input token i. In detail, the contextual semantic vectors are first fed into an attention cell to capture the most important information indicating the issue's priority. The attention cell maps the contextual semantic vectors into a 512-dimensional vector.

For the contextual semantic vectors $[c_1, ...c_i, ...c_n]$ (denoted as **C**), the attention cell will perform calculations using a trainable parameter matrix **W** to obtain the output. The formula is as follows:

$$\mathbf{M} = tanh(\mathbf{C}) \tag{1}$$

$$\alpha = softmax(\mathbf{M} * \mathbf{W}) \tag{2}$$

$$output = \sum \alpha \mathbf{C} \tag{3}$$

Specifically, we first fed contextual semantic vectors **C** into a *tanh* function to get the normalized representation **M**, and then calculate the global align weights α, which is the *softmax* value of the product of **M** and a trainable parameter matrix **W**. Finally, the attention output is the sum of all the contextual semantic vectors with attention weight.

To jointly capture other information for priority prediction, we fuse the 17 extracted indicative attributes into the vector. In other words, we concatenate the 512 dimensional vector with the 17 dimensional attribute vector to obtain a 529 dimensional new vector (denoted as a). The concatenated vector (a) is fed into a fully connected layer to obtain the prediction result. For priority prediction, we use *softmax* to output the probability of each priority, i.e., a X-dimensional (X is the number of priority types) vector $P(priority_{xi}|R)$ which represents the probability that reports R belong to a certain priority xi. The formula is as follows:

$$P(xi|a) = \frac{exp(a^\top w_{x_i})}{\sum_j exp(a^\top w_{x_j})} \tag{4}$$

Where the w stands for the trainable parameters in the fully connected layer.

Category prediction task layer. Different from the priority prediction, the category prediction is concerns more about the representative terms. For example, if "request" and "feature" appear in one issue report, the report is more likely to be a feature request. Taken in this sense, we not only utilize the contextual semantic vector, but also include the semantic vector produced by the word embedding layer which focuses more on the meaning of single tokens.

In detail, this network first concatenates the semantic vector and the contextual semantic vector. Then, we use max pooling to remove redundant information to improve the generalization of the model and reduce over-fitting.

Like the priority prediction, we adopt *softmax* to output the probability of each category, i.e., a Y-dimensional (Y is the number of category types) vector $P(category_{yi}|R)$ which represents the probability that report R belong to a certain category yi.

3.3 Model Training

3.3.1 ***Eavesdropping by Shared Loss Function.*** Before designing the loss function, we observe that the data instances for both priority prediction and category prediction tasks were imbalanced. Taking *kubernetes* project as an example, 42.7% of the issue reports are labeled as the *medium* priority, while only 25.9% are labeled as the *high* priority. For *category* labels, 63.3% of the issue reports are *bug*, while only 11.2% are *document*. To overcome the influence of imbalanced data, we adopt the *focal-loss* function [15] and tailor it for our multi-class prediction scenario. The formula is as follows:

$$Multi - FL = \sum_{i=1}^{n} -\alpha_i (1 - p_i)^2 log(p_i) \tag{5}$$

Where α_i stands for the weight of ith class, and p_i means the probability of predicting a data sample into ith class.

Then, following previous work, we employ a principle, shared loss function to take the weighted sum of the two individual losses, which is specified as:

$$Loss = \lambda Multi - FL_{priority} + (1 - \lambda) Multi - FL_{category} \tag{6}$$

where λ is the harmonic factor in the range (0,1). The shared loss function can allow the model to eavesdrop on information between two related tasks, and learn some knowledge that is difficult to learn from itself but easy to learn from the other task. The hyper-parameter λ is determined by a greedy strategy presented below.

3.3.2 ***Training Details.*** The hyper-parameters are tuned with a greedy strategy to obtain the best performance. Given a hyper-parameter P and its candidate values $\{v_1, v_2, ..., v_n\}$, we perform automated tuning for n iterations, and choose the values which lead to the best performance as the tuned value of P. After tuning, the harmonic factor λ is set as 0.5, and the learning rate is set as 5×10^{-5}. The optimizer is Adam algorithm [12]. We use the mini-batch technique for speeding up the training process with batch size 16. The drop rate is 0.1, which means 10% of neuron cells will be randomly masked to avoid over-fitting.

We implement PRIMA by using an open-source Pytorch library[3]. Our implementation and experimental data are available online[4].

4 EXPERIMENTAL DESIGN

4.1 Research Questions

We answer the following four research questions:

- **RQ1 (Effectiveness)**: What is the performance of PRIMA for priority prediction?
- **RQ2 (Multi-task Gain)**: What is the advantage of multi-task learning applied in PRIMA compared to single-task learning for priority prediction?
- **RQ3 (Information Necessity)**: What is the contribution of the indicative attributes for priority prediction?
- **RQ4 (Extra Benefit)**: For category prediction, what performance PRIMA can achieve compared with single-task learning?

4.2 Dataset

We utilize four open-source projects from various domains (e.g., operating system, web component framework) for the evaluation, as shown in Table 4. The selection criteria is that: more than 1k stars (popular), a public issue tracking system (traceable), more than three years of development history (trustworthy), more than 100 code commits (well maintained), and more than 1k closed issue

[3]https://github.com/huggingface/transformers
[4]https://github.com/piexpe/Multi-task_priority

reports. We first crawl all the closed issue reports for each project, and filter those reports that do not have the priority or category label. Note that, we need these two labels for model training and evaluation, and when applying the approach, we require none of this information. We further filter the reports which have less than 3 words for noise cleaning.

For priority types, project *zephyr* and *amphtml* utilize three labels, i.e., high, medium, and low. Project *kubernetes* and *minikube* utilize five labels, e.g., critical-urgent, important-longterm. To facilitate revealing the higher priority issues, we combine the lowest three priority types as low, and the highest two priority types still stand for high and medium respectively. This step is used to reorganize the five labels into three (i.e., high, medium and low). Note that, the proposed PRIMA can handle five labels of priority, yet in this evaluation, we use three labels to unify the demonstration of results across four projects.

For category types, we choose the three most common types, i.e., *bug*, *feature*, and *document*, for experiment. Note that again, PRIMA is scaleable to handle more category labels.

In total, there are 14,682 issue reports (#Issue) in the experimental dataset. We list the details of the priority distribution and category distribution in Table 4.

Table 4: Statistics of experimental dataset

Project	#Issue	Priority			Category		
		#High	#Medium	#Low	#Bug	#Feature	#Document
kubernetes	7017	1822	3001	2194	4440	1793	784
minikube	1094	364	334	396	469	449	176
zephyr	3097	840	1183	1074	1915	894	288
amphtml	3474	970	1207	1297	2015	1178	281
TOTAL	14682	3996	5725	4961	8839	4314	1529

4.3 Baselines

We employ two state-of-the-art approaches for priority prediction as the baselines to further demonstrate the advantages of PRIMA.

PPWGCN: this is the state-of-the-art approach using deep learning techniques by Fang et al.[7]. It builds a heterogeneous text graph for issue reports and applies graph convolutional networks based on weighted loss function to extract word's semantics in bug reports. We reuse the source code provided in the paper[5] and retrain the model with our training data to ensure the correctness.

DRONE: this is the state-of-the-art approach using machine learning techniques by Tian et al.[26, 27]. It extracts multiple factors from six dimensions, i.e., textual, temporal, author, related-report, severity, and product. Then it trains a machine learning model (e.g., SVM) and enhances the model with thresholding to handle imbalanced data. Since it does not provide the source code, we implement it strictly following the paper.

4.4 Experimental Setup

To simulate the usage of PRIMA in practice, we employ a commonly-used longitudinal data setup to potentially avoid the influence of using future data [28]. In detail, for all the issue reports in one project, we sort them in chronological order, and then divide them into 10 equally sized folds. We then employ the first 8 folds as the training dataset, the 9th fold as the validation dataset, and the last fold as the testing dataset.

To answer RQ1, we run PRIMA and each baseline to obtain their performance on the testing dataset. To answer RQ2, we configure PRIMA into the single task learning mode for the priority prediction, and compare the performance between PRIMA and its single-task learning mode for predicting the issue's priority. To answer RQ3, we design a variant of PRIMA excluding the indicative attributes from the model, and compare the performance between PRIMA and its variant for priority prediction. To answer RQ4, we configure PRIMA into the single task learning model as RQ2, yet for issue category prediction.

4.5 Evaluation Metrics

We use commonly-used metrics, i.e., precision, recall, and F1-Score, to evaluate the performance for priority prediction and category prediction. Precision is the ratio of the number of correct predictions to the total number of predictions. Recall is the ratio of the number of correct predictions to the total number of ground-truth samples. F1-Score is the harmonic mean of precision and recall.

Since the priority and category prediction are both involving multiple labels, we additionally obtain the average performance of precision, recall and F1 for each project across all the priority/-category labels. In addition, for all four experimental projects, we further obtain the average performance across all the priority/category labels to derive a full view.

5 RESULTS AND ANALYSIS

5.1 Answers to RQ1: Effectiveness of PRIMA

Table 5 demonstrates the performance of PRIMA and baselines on the priority prediction, i.e., high, medium, and low. PRIMA can achieve an average of 73.8% precision, 72.6% recall, and 72.3% F1, signifying the effectiveness of our approach in predicting the issue's priority.

We also observe that, among the three priorities, the approach achieves the highest performance (89% F1) for the high priority, while achieving the lowest performance (61.4% F1) for the low priority. For the high priority, the precision is 100% with 97% recall for *kubernetes* project, and the F1 is 90.5% for the *zyphyr* project. Since the issues with high priority are more important to the project's quality improvement, thus a higher performance of high priority prediction further signifies the practical value of this approach. For median and low priority, we find that there is usually a blurred boundary between them, which potentially causes a lower performance of these two priority types. Taken kubernetes #17561[6], whose ground truth priority is low while our prediction is median, as an example. The descriptive terms like "error", "unable" frequently occur in issues with median priority, which is the potential reason for our wrong prediction.

In addition, among the four projects, *minikube* achieves the lowest performance (78.6% F1 for high priority).The main reason

[5]https://github.com/TanYoushuai123/PPWGCN

[6]https://github.com/kubernetes/kubernetes/issues/17561

Table 5: Effectiveness of PRIMA (RQ1 and RQ2)

Priority	Project	Prima			Baseline 1: PPWGCN			Baseline 2: DRONE			Prima with single task		
		Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
High	kubernetes	**100**	**97.06**	**98.51**	63.15	72.23	67.38	36.26	33.7	34.93	93.38	74.71	83.01
	minkube	**79.55**	**77.78**	**78.65**	51.05	40.11	44.92	40.74	30.56	34.92	60.71	37.78	46.58
	zephyr	**90.53**	**90.53**	**90.53**	51.58	62.14	56.37	78.95	66.67	72.29	79.38	81.05	80.21
	amphtml	**98.8**	**80.39**	**88.65**	54.44	55.67	55.05	52.94	50.56	51.72	47.54	56.86	51.79
	Average	**92.22**	**86.44**	**89.09**	55.06	57.54	55.93	52.22	45.37	48.47	70.25	62.60	65.40
Medium	kubernetes	**72.24**	**76.08**	**74.11**	67.6	59.49	63.29	48	46.6	47.29	61.75	75.08	67.77
	minkube	**67.65**	**88.46**	**76.67**	42.58	39.52	40.99	30.43	41.18	35	48.28	53.85	50.91
	zephyr	58.17	**83.96**	**68.73**	**58.68**	54.22	56.37	50.76	59.82	54.92	58.52	74.53	65.56
	amphtml	**49.51**	44.74	47	45.47	41.56	43.43	43.24	**53.78**	**47.94**	42.07	53.51	47.1
	Average	**61.89**	**73.31**	**66.63**	53.58	48.70	51.02	43.11	50.35	46.29	52.66	64.24	57.84
Low	kubernetes	**66.97**	63.79	65.34	64.22	**67.09**	**65.63**	35.06	38.76	36.82	63.68	55.17	59.12
	minkube	**68.75**	56.41	61.97	51.81	65.15	57.72	40.54	37.5	38.96	56.6	**76.92**	**65.22**
	zephyr	**79.37**	45.45	57.8	63.91	**59.03**	**61.37**	58.82	55.56	57.14	70.89	50.91	59.26
	amphtml	54.94	**67.42**	**60.54**	56.38	59.94	58.1	53.04	43.57	47.84	**62.96**	38.64	47.89
	Average	**67.51**	58.27	**61.41**	59.08	**62.80**	60.71	46.87	43.85	45.19	63.53	55.41	57.87
Overall		**73.87**	**72.67**	**72.38**	55.91	56.35	55.89	47.40	46.52	46.65	62.15	60.75	60.37

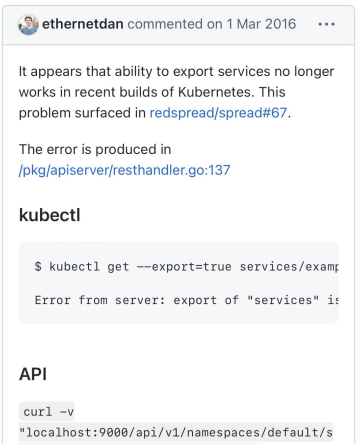


Figure 5: An example issue report of Kubernetes project

might be due to the small size of the experimental data of *minikube* project, which can hardly train an effective model.

Compared with the baselines, PRIMA is 32% higher in precision, and 28% higher in recall compared with PPWGCN, while PRIMA is 55% higher in precision, and 56% higher in recall compared with DRONE. This further demonstrates the advantages of our proposed PRIMA, i.e., capturing the semantics of issues and boosting the performance with multi-task learning. Only in predicting medium and low types of priority, the baselines occasionally outperforming our proposed approach slightly. Among the two baselines, the deep learning based approach PPWGCN is superior than the machine learning based approach DRONE, which indicates the advantages of deep learning approaches in capturing the semantics of the issue.

We use an issue[7] presented in Figure 5 to intuitively illustrate the superior performance of PRIMA compared with baselines. This issue is related with the not working state of the export services, and the manager marked it as priority/critical-urgent, i.e., the high priority, while PRIMA obtains the correct prediction, yet PPWGCN and DRONE predict it incorrectly, as low and median, respectively.

The first reason is that our proposed PRIMA uses multi-task learning which can gain the knowledge of prediction for category labels to better conduct the priority prediction. For this issue, the manager mentions that it is a bug in the comment, and tags it with both kind/bug and priority/critical-urgent labels at the same time. It indicates that the manager may assign the priority label in conjunction with assigning category label, which further demonstrates the design of PRIMA is reasonable. The second reason is that our proposed approach is better at handling the code related information, such as console message and API message, where the information (e.g., "error from server") can indicate the issue's priority. And the third reason is our utilized domain-specific BERT model and attention mechanism can better capture the key information for predicting the priority.

[7]https://github.com/kubernetes/kubernetes/issues/22247

Table 6: Priority prediction of PRIMA and its variant (RQ3)

Priority	Project	Prima			Prima Without Attributes		
		Precision	Recall	F1	Precision	Recall	F1
High	kubernetes	**100**	**97.06**	**98.51**	92.09	75.29	82.85
	minikube	**79.55**	**77.78**	**78.65**	74.42	71.11	72.73
	zephyr	**90.53**	**90.53**	**90.53**	87.5	88.42	87.96
	amphtml	**98.8**	80.39	**88.65**	84.16	**83.33**	83.74
	Average	**92.22**	**86.44**	**89.09**	84.54	79.54	81.82
Medium	kubernetes	**72.24**	**76.08**	**74.11**	59.49	62.46	60.94
	minikube	**67.65**	88.46	76.67	66.67	**92.31**	**77.42**
	zephyr	58.17	**83.96**	**68.73**	**60.87**	66.04	63.35
	amphtml	**49.51**	44.74	47	49.19	**53.51**	**51.26**
	Average	**61.89**	**73.31**	**66.63**	59.06	68.58	63.24
Low	kubernetes	**66.97**	**63.79**	**65.34**	54.44	58.19	56.25
	minikube	**68.75**	**56.41**	**61.97**	64.52	51.28	57.14
	zephyr	**79.37**	45.45	57.8	65	**59.09**	**61.9**
	amphtml	54.94	**67.42**	**60.54**	**58.54**	54.55	56.47
	Average	**67.51**	58.27	**61.41**	60.63	**55.78**	57.94
Overall		**73.87**	**72.67**	**72.38**	68.07	67.97	67.67

Table 7: Category prediction performance compared with single-task learning (RQ4)

Category	Project	Prima			Prima With Category task		
		Precision	Recall	F1	Precision	Recall	F1
Bug	kubernetes	94.64	**94.85**	**94.75**	**94.85**	90.6	92.68
	minikube	**82.98**	**82.98**	**82.98**	76.09	74.47	75.27
	zephyr	**89.09**	80.77	**84.73**	78.85	**90.11**	84.1
	amphtml	**89.53**	**81.04**	**85.07**	86.53	79.15	82.67
	Average	**89.06**	**84.91**	**86.88**	84.08	83.58	83.68
Feature	kubernetes	**88.2**	**87.22**	**87.71**	78.11	87.22	82.41
	minikube	**68.89**	**79.49**	**73.81**	61.22	76.92	68.18
	zephyr	67.48	**85.57**	**75.45**	**81.01**	65.98	72.73
	amphtml	**69.7**	**82.14**	**75.41**	63.04	77.68	69.6
	Average	**73.57**	**83.61**	**78.10**	70.85	76.95	73.23
Document	kubernetes	**97.4**	**98.68**	**98.04**	97.33	96.05	96.69
	minikube	83.33	**62.5**	**71.43**	**86.67**	54.17	66.67
	zephyr	**69.57**	**50**	**58.18**	62.5	46.88	53.57
	amphtml	**100**	**100**	**100**	88.24	60	71.43
	Average	**87.58**	**77.80**	**81.91**	83.69	64.28	72.09
Overall		**83.40**	**82.10**	**82.30**	79.54	74.94	76.33

5.2 Answers to RQ2: Multi-task Gain

Table 5 also presents the performance of single task learning mode of priority prediction. We can observe that PRIMA outperforms the single task learning mode, 18% higher in precision, and 19% higher in recall. This indicates the shared features learned from two related tasks indeed improve the priority prediction. The performance improvement is attributed to the internal shared learning mechanism of PRIMA (shared layers learning and shared loss function), which can share features between two related tasks.

In addition, PRIMA outperforms the single task learning mode in the high priority prediction, which is more beneficial considering the influence of the critical issues. The single task learning mode can occasionally outperform PRIMA in medium and low priority types slightly. This might be because the lower priority types are difficult to be predicted inherently.

5.3 Answers to RQ3: Information Necessity

Table 6 demonstrates the performance of PRIMA and its variant without indicative attributes. We can see that PRIMA achieves average 8.5% higher precision and 6.9% higher recall, compared with its variant. This suggests the necessity of these indicative attributes for predicting issue's priority, and the superiority of our design model architecture.

5.4 Answers to RQ4: Extra Benefit

Table 7 presents the performance of PRIMA for category prediction, and the category prediction with single-task learning. Generally speaking, with PRIMA, the category prediction achieves 83% average precision, and 82% average recall, outperforming its variant of single-task learning, i.e., 4.8% higher in precision and 9.5% higher in recall.

This indicates that PRIMA can effectively predict both issue's priority and issue's category, acting as an add-on for facilitating the project a step further towards the automatic issue management. When answering RQ2, we observe that for priority, PRIMA is average 18% higher in precision and 19% higher in recall compared with single-task learning mode. Meanwhile, as we mentioned above, for category prediction, PRIMA is 4.8% higher in precision and 9.5% higher in recall compared with the single-task learning mode. This might because, the performance of category prediction with single-task learning mode is already relatively high; and only little knowledge for predicting the issue's priority can be shared with the issue category prediction.

6 DISCUSSIONS AND THREATS TO VALIDITY

6.1 Field Evaluation

The performance in Section 5 is obtained based on the historical issues with the marked priority labels. This section conducts an experiment on the newly reported issues to further examine whether our proposed approach can accurately predict the priority labels, and to investigate whether the developer thinks them useful.

In detail, we used the four experimental projects and also randomly selected two other projects following the criteria in Section 3.1. We crawled the newly submitted issues, and run our prediction model on these issues, and obtained the predicted priority of each issue. We then submitted a comment below the related issue, suggesting the issue's priority level with the primary idea about how we determine it. In total, 8 responses had been received among the 18 submitted comments. Of all the responses, 6 issues were confirmed as with the right priority label, while 2 issues were denied. We put the detailed information in Table 8.

Some developers present the detailed comments about whether they think the prediction is correct or wrong. For example, for issue "woocommerce #32702" whose ground truth priority is high, the developers respond that *I think this should be the high priority,*

Table 8: Details of responded issues

Result	Issue	Prediction	Ground truth
correct prediction	kubernetes #109500	Mid	Mid
	zephyr #44018	Mid	Mid
	zephyr #44531	Low	Low
	amphtml #38061	Low	Low
	woocommerce #32702	High	High
	yugabyte-db #12190	High	High
wrong prediction	kubernetes #109543	High	Mid
	zephyr #44704	Mid	Low

it looks like it could be a regression (bad) but it's in a high value area regardless. Another example is for issue "kubernetes #109543" whose ground truth priority is median, the developers respond that *No. This problem is expected to exist for quite some time. Also, this test is just for measuring performance and does not block merging. Correct labeling requires background knowledge of the issue. For example, if, hypothetically, this test is built into ci and blocks the merge when it fails, then you are right, it would be a high priority.* These further indicate the complexity of the priority prediction task, and also the potential of our proposed approach in recommending the right priority with the learned background knowledge.

6.2 Threats to Validity

The external threats concern the generality of the proposed approach. We trained and evaluated PRIMA on the dataset consisting of four open-source projects. The selected projects are from various domains, relatively large-scale, popular, well-maintained, etc, which potentially reduce this threat. The construct threats mainly question the disposal of the five levels of priority for two of the experimental projects. To provide a unified performance demonstration across the four experimental projects, we combine the lowest three priority types as low priority, and reorganize the five levels of priority to three levels. This might slightly influence the performance of PRIMA, yet the relative high performance achieved on the other two projects (with three priority levels) has demonstrated the effectiveness of our approach.

7 RELATED WORK

Issue Report Management. The issue reports in open-source projects provide important clues for improving the quality of the software. There are various approaches targeting at the automatic management of issue reports, e.g., classification, duplication detection [24, 30, 31], bug triage [2, 10, 32], bug localization [11, 13, 37].

There are some studies focusing on the issue's priority prediction [7, 26, 27, 29]. Tian et al. [26, 27] extracted features from six dimensions, i.e., temporal, textual, author, related-report, severity, and product, and built a machine learning model for priority prediction. Umer et al. [29] utilized a convolutional neural network to implement an automated priority assignment model, which can eliminate the efforts for feature definition and modeling. Fang et al. [7] proposed to utilize graph convolutional networks together with weighted loss function for priority prediction, to better capture the syntactic and semantic information. We have proposed a new approach and achieved higher performance compared with them.

There are also some studies focusing on the severity prediction of issue reports [8, 18, 19], which is similar to this work. They utilized text mining, machine learning, or deep learning techniques to predict the issue's priority. However, severity is different from priority, in that severity is assigned by customers while priority is provided by developers [26], and our experimental projects hosted on Github do not have the severity label. Nevertheless, there is no attempt for utilizing the multi-task learning for severity prediction, and our proposed approach can potentially be tailored for the severity prediction and facilitate performance improvement.

Multi-task Learning in SE. Some studies have used MTL to simultaneously address two related tasks, or improve target tasks by pre-training language models and vector representations on auxiliary tasks. For example, DEMAR [14] proposed a deep MTL-based model to jointly conduct requirements discovery and requirements annotation tasks. MTFuzz [23] employed MTL to learn a compact embedding of program input spaces to guide the mutation process. Other studies about code analysis with MTL included modeling source code with pre-trained language model for code understanding and generation [17], enabling knowledge sharing between related tasks for code completion [16], and leveraging method name generation and method name informativeness prediction to improve code summarization [34]. This work addresses the issue priority prediction problem with multi-task learning, meanwhile, our approach is not a simple and direct application of multi-task learning framework, the incorporation of indicative attributes, and the design of two types of shared layers improves the model further.

8 CONCLUSION

The true benefits of issue reports in open-source projects have largely been attributed to the effective management of these issues, and this paper focus on the automatic priority prediction to help direct the corrective maintenance and improve software quality. This paper proposes PRIMA, a deep multi-task learning based approach to automate the priority prediction while taking the issue category prediction as the second task. We evaluate PRIMA on four large-scale open-source projects, and the results confirm the effectiveness of the proposed approach, and the benefits of multi-task learning mechanism. The proposed approach provides an effective way for automatic priority prediction, and sheds light on jointly learning two tasks of the issue management.

ACKNOWLEDGMENTS

This work is supported by the Fundamental Research Funds for the Central Universities, Southwest Minzu University (Grant Number 2020PTJS18002), the Southwest Minzu University Research Startup Funds (Grant No.RQD2021096), Sichuan Science and Technology Program (2021JDRC0066); and the National Natural Science Foundation of China under grant No.61602450.

REFERENCES

[1] Rafi Almhana, Thiago do Nascimento Ferreira, Marouane Kessentini, and Tushar Sharma. 2020. Understanding and Characterizing Changes in Bugs Priority: The Practitioners' Perceptive. In *20th IEEE International Working Conference on Source*

Code Analysis and Manipulation, SCAM 2020, Adelaide, Australia, September 28 - October 2, 2020. IEEE, 87–97.

[2] John Anvik, Lyndon Hiew, and Gail C Murphy. 2006. Who should fix this bug?. In *Proceedings of the 28th international conference on Software engineering*. 361–370.

[3] Rich Caruana. 1993. Multitask Learning: A Knowledge-Based Source of Inductive Bias. In *Machine Learning, Proceedings of the Tenth International Conference, University of Massachusetts, Amherst, MA, USA, June 27-29, 1993*. Morgan Kaufmann, 41–48.

[4] Rich Caruana. 1998. Multitask Learning. In *Learning to Learn*. Springer, 95–133.

[5] Ronan Collobert and Jason Weston. 2008. A unified architecture for natural language processing: deep neural networks with multitask learning. In *Machine Learning, Proceedings of the Twenty-Fifth International Conference (ICML 2008), Helsinki, Finland, June 5-9, 2008 (ACM International Conference Proceeding Series)*, Vol. 307. ACM, 160–167.

[6] Long Duong, Trevor Cohn, Steven Bird, and Paul Cook. 2015. Low Resource Dependency Parsing: Cross-lingual Parameter Sharing in a Neural Network Parser. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 2: Short Papers*. The Association for Computer Linguistics, 845–850.

[7] Sen Fang, Youshuai Tan, Tao Zhang, Zhou Xu, and Hui Liu. 2021. Effective Prediction of Bug-Fixing Priority via Weighted Graph Convolutional Networks. *IEEE Trans. Reliab.* 70, 2 (2021), 563–574.

[8] Luiz Alberto Ferreira Gomes, Ricardo da Silva Torres, and Mario Lúcio Côrtes. 2019. Bug report severity level prediction in open source software: A survey and research opportunities. *Inf. Softw. Technol.* 115 (2019), 58–78.

[9] Ian J. Goodfellow, Yoshua Bengio, and Aaron C. Courville. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org/

[10] Gaeul Jeong, Sunghun Kim, and Thomas Zimmermann. 2009. Improving bug triage with bug tossing graphs. In *Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*. 111–120.

[11] Dongsun Kim, Yida Tao, Sunghun Kim, and Andreas Zeller. 2013. Where should we fix this bug? a two-phase recommendation model. *IEEE transactions on software Engineering* 39, 11 (2013), 1597–1610.

[12] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1412.6980

[13] Tien-Duy B Le, Richard J Oentaryo, and David Lo. 2015. Information retrieval and spectrum based bug localization: Better together. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. 579–590.

[14] Mingyang Li, Lin Shi, Ye Yang, and Qing Wang. 2020. A Deep Multitask Learning Approach for Requirements Discovery and Annotation from Open Forum. In *35th IEEE/ACM International Conference on Automated Software Engineering, ASE 2020, Melbourne, Australia, September 21-25, 2020*. IEEE, 336–348.

[15] Tsung-Yi Lin, Priya Goyal, Ross B. Girshick, Kaiming He, and Piotr Dollár. 2017. Focal Loss for Dense Object Detection. *CoRR* abs/1708.02002 (2017). arXiv:1708.02002 http://arxiv.org/abs/1708.02002

[16] Fang Liu, Ge Li, Bolin Wei, Xin Xia, Zhiyi Fu, and Zhi Jin. 2020. A Self-Attentional Neural Architecture for Code Completion with Multi-Task Learning. In *ICPC '20: 28th International Conference on Program Comprehension, Seoul, Republic of Korea, July 13-15, 2020*. ACM, 37–47.

[17] Fang Liu, Ge Li, Yunfei Zhao, and Zhi Jin. 2020. Multi-task Learning based Pre-trained Language Model for Code Completion. In *35th IEEE/ACM International Conference on Automated Software Engineering, ASE 2020, Melbourne, Australia, September 21-25, 2020*. IEEE, 473–485.

[18] Wenjie Liu, Shanshan Wang, Xin Chen, and He Jiang. 2018. Predicting the Severity of Bug Reports Based on Feature Selection. *Int. J. Softw. Eng. Knowl. Eng.* 28, 4 (2018), 537–558.

[19] Tim Menzies and Andrian Marcus. 2008. Automated severity assessment of software defect reports. In *24th IEEE International Conference on Software Maintenance (ICSM 2008), September 28 - October 4, 2008, Beijing, China*. IEEE Computer Society, 346–355.

[20] Tomás Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. In *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1301.3781

[21] Jinfeng Rao, Ferhan Türe, and Jimmy Lin. 2018. Multi-Task Learning with Neural Networks for Voice Query Understanding on an Entertainment Platform. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2018, London, UK, August 19-23, 2018*. ACM, 636–645.

[22] Sebastian Ruder. 2017. An Overview of Multi-Task Learning in Deep Neural Networks. *CoRR* abs/1706.05098 (2017).

[23] Dongdong She, Rahul Krishna, Lu Yan, Suman Jana, and Baishakhi Ray. 2020. MTFuzz: fuzzing with a multi-task neural network. In *ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020*. ACM, 737–749.

[24] Chengnian Sun, David Lo, Xiaoyin Wang, Jing Jiang, and Siau-Cheng Khoo. 2010. A discriminative model approach for accurate duplicate bug report retrieval. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*. 45–54.

[25] Jeniya Tabassum, Mounica Maddela, Wei Xu, and Alan Ritter. 2020. Code and Named Entity Recognition in StackOverflow. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetreault (Eds.). Association for Computational Linguistics, 4913–4926. https://doi.org/10.18653/v1/2020.acl-main.443

[26] Yuan Tian, David Lo, and Chengnian Sun. 2013. Drone: Predicting priority of reported bugs by multi-factor analysis. In *2013 IEEE International Conference on Software Maintenance*. IEEE, 200–209.

[27] Yuan Tian, David Lo, Xin Xia, and Chengnian Sun. 2015. Automated prediction of bug report priority using multi-factor analysis. *Empir. Softw. Eng.* 20, 5 (2015), 1354–1383.

[28] Feifei Tu, Jiaxin Zhu, Qimu Zheng, and Minghui Zhou. 2018. Be careful of when: an empirical study on time-related misuse of issue tracking data. In *Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2018, Lake Buena Vista, FL, USA, November 04-09, 2018*, Gary T. Leavens, Alessandro Garcia, and Corina S. Pasareanu (Eds.). ACM, 307–318.

[29] Qasim Umer, Hui Liu, and Inam Illahi. 2019. CNN-based automatic prioritization of bug reports. *IEEE Transactions on Reliability* 69, 4 (2019), 1341–1354.

[30] Junjie Wang, Mingyang Li, Song Wang, Tim Menzies, and Qing Wang. 2019. Images don't lie: Duplicate crowdtesting reports detection with screenshot information. *Inf. Softw. Technol.* 110 (2019), 139–155.

[31] Xiaoyin Wang, Lu Zhang, Tao Xie, John Anvik, and Jiasu Sun. 2008. An approach to detecting duplicate bug reports using natural language and execution information. In *Proceedings of the 30th international conference on Software engineering*. 461–470.

[32] Xin Xia, David Lo, Ying Ding, Jafar M Al-Kofahi, Tien N Nguyen, and Xinyu Wang. 2016. Improving automated bug triaging with specialized topic model. *IEEE Transactions on Software Engineering* 43, 3 (2016), 272–297.

[33] Xin Xia, David Lo, Emad Shihab, and Xinyu Wang. 2015. Automated bug report field reassignment and refinement prediction. *IEEE Transactions on Reliability* 65, 3 (2015), 1094–1113.

[34] Rui Xie, Wei Ye, Jinan Sun, and Shikun Zhang. 2021. Exploiting Method Names to Improve Code Summarization: A Deliberation Multi-Task Learning Approach. *CoRR* abs/2103.11448 (2021).

[35] Yongxin Yang and Timothy M. Hospedales. 2017. Trace Norm Regularised Deep Multi-Task Learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings*. OpenReview.net.

[36] Tianzhu Zhang, Bernard Ghanem, Si Liu, and Narendra Ahuja. 2012. Robust visual tracking via multi-task sparse learning. In *2012 IEEE Conference on Computer Vision and Pattern Recognition, Providence, RI, USA, June 16-21, 2012*. IEEE Computer Society, 2042–2049.

[37] Jian Zhou, Hongyu Zhang, and David Lo. 2012. Where should the bugs be fixed? more accurate information retrieval-based bug localization based on bug reports. In *2012 34th International Conference on Software Engineering (ICSE)*. IEEE, 14–24.

[38] Weiqin Zou, David Lo, Zhenyu Chen, Xin Xia, Yang Feng, and Baowen Xu. 2018. How practitioners perceive automated bug report management techniques. *IEEE Transactions on Software Engineering* 46, 8 (2018), 836–862.