Received 2 January 2024, accepted 12 February 2024, date of publication 22 February 2024, date of current version 14 March 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3368915

RESEARCH ARTICLE

Enhancing Bug Report Summaries Through Knowledge-Specific and Contrastive Learning Pre-Training

YUNNA SHAO AND BANGMENG XIANG
Zhejiang College of Security Technology, Wenzhou, Zhejiang 325000, China

Corresponding author: Bangmeng Xiang (13957787633@163.com)

This work was supported in part by Wenzhou Municipal Science and Technology Plan Project under Grant 2023R0016, and in part by Wenzhou Philosophy and Social Sciences Planning Annual Topic under Grant 23WSK208YBM.

ABSTRACT Bug reports are crucial in software maintenance, with concise summaries significantly enhancing the efficiency of bug triagers and ultimately contributing to the development of high-quality software products. Contemporary methods for automatic bug report summarization primarily utilize neural networks' robust learning capabilities. However, these approaches often produce suboptimal summaries due to two primary limitations: 1) the difficulty in assimilating the domain-specific knowledge inherent in bug reports, and 2) the limitations of purely supervised learning in comprehending the comprehensive context of bug reports. To address the above two problems, in this paper, we propose a new approach for bug report summarization, namely KSCLP, which leverages large language models and domain-specific pre-training strategies, i.e., Knowledge-Specific and Contrastive Learning Pre-training. Specifically, the Knowledge-Specific strategy allows to pre-train KSCLP on project-specific bug reports corpus, by which the model can fully learn internal knowledge of bug reports, learning bug report-aware representation. As for the Contrastive Learning strategy, it performs a sequence-level pre-training for KSCLP, helping it capture the semantic information of bug reports on a global level. Upon completion of the pre-training phase, KSCLP undergoes further refinement through a Sequence-to-Sequence framework specifically tailored for bug report summarization. The efficacy of KSCLP is rigorously evaluated against five baseline models using a publicly available dataset. The empirical results demonstrate that KSCLP outperforms all baselines, achieving remarkable improvements by up to 23.73, 13.97, and 20.89 points in ROUGE-1, ROUGE-2, and ROUGE-L metrics, thereby setting new benchmarks in the field of bug report summarization.

INDEX TERMS Bug report summarization, domain-specific pre-training, software maintenance, representation learning.

I. INTRODUCTION

Bug reports play a pivotal role in the software maintenance lifecycle [1]. However, the surge in bug reports submitted to tracking systems like Bugzilla[1] and Jira,[2] as mentioned in Fang et al.'s work [2], has rendered activities such as severity and priority assessment of bugs increasingly time-consuming and cumbersome for triagers. This escalation in workload adversely affects the efficiency of the entire software maintenance cycle. Consequently, there is a pressing need for an effective mechanism to swiftly and accurately summarize bug reports. Such a summarization process would enable triagers to comprehend bug reports more efficiently, thereby enhancing the overall efficiency of software maintenance.

Automating bug report summarization can significantly enhance the efficiency of bug triagers, thereby improving software product quality. To this end, researchers have proposed various approaches, ranging from information retrieval-based methods [3] to neural network-based (NN-based) techniques [4], for automatically summarizing newly

The associate editor coordinating the review of this manuscript and approving it for publication was Giuseppe Destefanis.

[1]https://www.bugzilla.org/
[2]https://www.atlassian.com/software/jira

submitted bug reports. Recently, NN-based approaches have gained prominence due to their powerful learning capabilities. Li et al. introduce DeepSum, the first NN-based method for automatic bug report summarization. This approach utilizes word embeddings [5] to construct an auto-encoder network, transforming bug reports into semantic vectors based on individual sentences. It then employs a dynamic programming method for sentence selection during summarization. Building upon this, Liu et al. develop BugSum, which leverages the deeper context of bug reports. Rather than relying on word embeddings, BugSum uses a Bi-GRU-based auto-encoder and introduces a novel metric, the believability score, to enhance sentence vector representation. This method also employs a dynamic selection strategy to choose appropriate sentences for summarization. Diverging from these methods, Sutskever et al. propose RTA, which is pre-trained a large language model on an extensive bug report corpus, and subsequently fine-tuned it for bug report summarization under the Sequence-to-Sequence framework [6].

Although neural network (NN)-based methods demonstrate efficacy, they encounter two primary limitations that lead to suboptimal summarization quality in the context of bug reports. Firstly, these methods often fail to assimilate domain-specific knowledge inherent in bug reports. Typically, bug reports are replete with detailed descriptions and essential information about the bugs. However, conventional neural networks and generic pre-training techniques struggle to extract these critical features. Consequently, such models tend to grasp only the superficial semantic content of bug reports, yielding summaries that lack professional depth. Secondly, the challenge intensifies with supervised training's inability to comprehend the holistic context of bug reports. Bug reports generally blend natural language with technical elements like APIs, code lines, and snippets. Traditional supervised learning models are often inadequate in discerning the subtle semantic interplays and contextual nuances within bug reports. This limitation limits the models' capacity to deeply understand different bug reports at a global level, which makes them finally produce unuseful bug report summarization.

To address the challenges previously outlined, we introduce a novel approach for bug report summarization, designated as KSCLP. This method synergizes large language models with tailored domain-specific pre-training strategies. Our initial step involves the development of a Knowledge-Specific masked language model for pre-training KSCLP. This method is uniquely focused on predicting tokens related to project-specific bug reports, thereby facilitating the acquisition of in-depth bug report knowledge. Additionally, we have developed a contrastive learning pre-training objective. This objective empowers KSCLP to conduct sequence-level learning, granting it a comprehensive understanding of entire bug reports within a broader contextual framework. Upon completing the pre-training phase, KSCLP is integrated into a Sequence-to-Sequence learning framework and subsequently fine-tuned to effectively generate concise and accurate bug report summaries. Contrasting with previous methods that predominantly extract sentences directly from bug reports, our approach is capable of crafting new sentences, thereby achieving more precise and informative summarizations.

To assess the efficacy of KSCLP, we selected five cutting-edge methodologies as benchmarks: DeepSum [4], BugSum [7], PRHAN [8], Transformer [9], and RTA [1]. We conducted an extensive performance comparison using a publicly available bug report corpus provided by Fang et al. [1]. Initially, we partitioned the corpus into training, validation, and testing sets, adhering to previously established splitting methods [1], [10]. Subsequently, we utilized the training set to pre-train and fine-tune KSCLP, followed by its performance evaluation on both the validation and testing sets. The experimental outcomes demonstrate that KSCLP surpasses all benchmark approaches in terms of ROUGE-1, ROUGE-2, and ROUGE-L scores, showing improvements ranging from 1.45 to 23.73 points. This significant enhancement in performance corroborates the effectiveness of KSCLP. Additionally, we also compare KSCLP with other large language models [11], [12], [13], and the experimental results show that KSCLP can get at least 4.31 points improvement overall evaluated metrics, indicating its superiority.

Our contributions are summarized as follows,

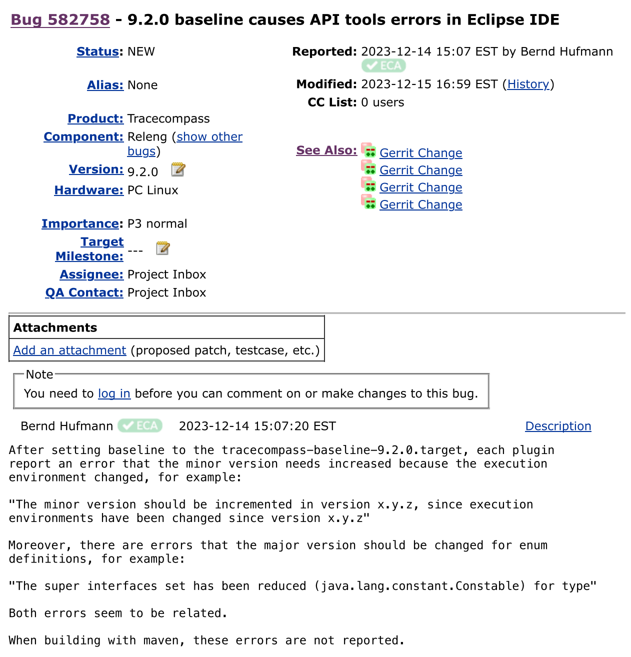
- We propose KSCLP and design two pre-training objectives for training it, including a Knowledge-Specific masked language model and a contrastive learning objective. To the best of our knowledge, this is the first work to employ domain-specific knowledge in the task of bug report summarization.
- We conduct a series of comparative experiments to evaluate the effectiveness of KSCLP and compare it with five baseline approaches. The experimental results demonstrate our designed strategies for KSCLP are effective for the bug report summarization task.
- To further evaluate the effectiveness of KSCLP, we compare it with the existing large language models, and the experimental results show its effectiveness on the task of bug report summarization.

The remainder of this paper is organized as follows. Section II describes the background knowledge of this paper, and Section III elaborates on our proposed approach, KSCLP. Section IV and Section V present the experimental setups and results, respectively. Section VI discusses the threats to validity. Finally, we conclude this paper and point out the future work in Section VII.

II. BACKGROUND

A. BUG REPORTS

Bug reports are integral to both software development and maintenance, serving as formal documentation of detected issues, problems, or errors within a software application [14],



Bug 582758 - 9.2.0 baseline causes API tools errors in Eclipse IDE

Status: NEW
Alias: None
Product: Tracecompass
Component: Releng (show other bugs)
Version: 9.2.0
Hardware: PC Linux
Importance: P3 normal
Target Milestone: ---
Assignee: Project Inbox
QA Contact: Project Inbox

Reported: 2023-12-14 15:07 EST by Bernd Hufmann ECA
Modified: 2023-12-15 16:59 EST (History)
CC List: 0 users

See Also: Gerrit Change
Gerrit Change
Gerrit Change
Gerrit Change

Attachments
Add an attachment (proposed patch, testcase, etc.)

Note
You need to log in before you can comment on or make changes to this bug.

Bernd Hufmann ECA 2023-12-14 15:07:20 EST Description

```
After setting baseline to the tracecompass-baseline-9.2.0.target, each plugin
report an error that the minor version needs increased because the execution
environment changed, for example:

"The minor version should be incremented in version x.y.z, since execution
environments have been changed since version x.y.z"

Moreover, there are errors that the major version should be changed for enum
definitions, for example:

"The super interfaces set has been reduced (java.lang.constant.Constable) for type"

Both errors seem to be related.

When building with maven, these errors are not reported.
```

FIGURE 1. An example of bug report with id 582758 in Eclipse platform.

[15]. Typically, these reports encompass detailed descriptions of the bug's symptoms, the requisite steps for its reproduction, and pertinent environmental or system information. By furnishing this comprehensive data, bug reports empower developers to pinpoint, diagnose, and rectify software bugs effectively. Furthermore, they facilitate the tracking of bug resolutions and ensure that all identified bugs are addressed prior to the software's release to end-users. The efficacy of bug reporting is paramount in maintaining software quality and reliability, as it enables developers to tackle issues in a timely and efficient manner.

Figure 1 presents a bug report from the Eclipse project,[3] retrieved from the Bugzilla platform. Bug reports typically comprise several components, such as Description, Summary, Component, Version, Assignee, Comment, among others. Each component serves a distinct function, contributing vital information for resolving bugs. The Description element, for example, usually contains comprehensive details about the bug, including its symptoms, effects, and impact on software performance. The Summary element offers a succinct overview of the problem, enabling developers to quickly ascertain the nature of the bug. The Component element identifies the specific module or segment of the software application affected by the bug, while the Version element indicates the software application's version in which the bug was found.

B. AUTOMATED BUG REPORT SUMMARIZATION

Automated bug report summarization, as explored in studies by Li et al. [4], represents a pivotal component of software maintenance. This process leverages machine learning techniques to distill extensive bug reports into concise, natural language summaries. The primary goal is to encapsulate the essence of a bug report in a few succinct sentences. Such a technique is instrumental for software developers and testers, facilitating rapid bug resolution by providing a clear and immediate understanding of the core issues. Moreover, automated summarization can substantially alleviate the time and labor involved in manual bug triage and classification, offering a streamlined and focused conclusion for bug triagers. Consequently, this automation not only conserves time for developers and testers but also enhances the efficiency of bug fixing. Ultimately, it contributes to elevating the overall software quality, thereby fostering greater user satisfaction.

C. LARGE LANGUAGE MODEL

Large language models (LLMs) [11], [16] are developed through pre-training on extensive corpuses using unsupervised learning methods. This approach enables them to acquire a universal representation of the data they are trained on. Subsequently, through supervised fine-tuning, these models are adept at performing a wide array of Natural Language Processing (NLP) tasks. Examples of such tasks include sentiment analysis [17], text summarization [18], and language translation [19].

LLMs offer a host of benefits that can greatly reduce the reliance on labeled data in downstream natural language processing tasks. This is because their pre-existing knowledge of natural language enables them to achieve state-of-the-art performance with relatively small amounts of labeled data, which is particularly advantageous in situations where obtaining labeled data is difficult or expensive. Due to the above advantages, more and more LLMs are proposed for various software engineering tasks [1], [13], [20]. Similar to LLMs for natural languages, these models are first pre-trained on massive software engineering data, e.g., source code collected from GitHub, then further fine-tuned for specific downstream tasks, like code search [21], clone code detection [22], code summarization [23], code optimization [24], and program repair [25]. Due to the powerful representation learning ability of LLMs, they achieve state-of-the-art results on these tasks.

D. RELATED WORK

Early methodologies in bug report summarization were primarily anchored in information retrieval concepts. Rastkar et al. [26] and Jiang et al. [27] developed approaches to select pertinent sentences from entire bug reports, utilizing classifiers trained on specific features. Consequently, the effectiveness of these models hinges significantly on the training corpus's quality [28]. Arya et al. [29] introduced a method for categorizing comments based on the information they contain, allowing users to select sentences that meet their specific requirements. In a novel approach, Radev et al. [30]

[3]https://bugs.eclipse.org/bugs/

proposed compressing sentences into vectors based on their TF-IDF values and then selecting sentences with similarities to the centroid of all sentence vectors. Other studies, such as those by Zhu et al. [31] and Mei et al. [32], have explored sentence selection based on reference relations, a technique further refined through a noise removal strategy proposed by Mani et al. [33]. In a departure from these methodologies, Liu et al. introduced an innovative unsupervised algorithm for bug report summarization, designed to minimize the inclusion of controversial sentences in summaries. In contrast to these sentence-selection-based methods, Fang et al. [1] proposed the RTA model, a novel approach where they fine-tuned a generative model to create summaries for bug reports, representing a significant evolution in bug report summarization techniques.

Contrary to previous methods, the KSCLP model merges large language models with contrastive learning, enabling it to assimilate domain knowledge from bug reports while capturing their contextual nuances. Our innovative contrastive learning objective empowers KSCLP to perform sequence-level analysis, thereby extracting profound semantic representations across various bug reports. This capability endows KSCLP with the proficiency to produce superior summarizations of bug reports.

III. APPROACH

This section outlines the KSCLP pipeline for summarizing bug reports, as depicted in Figure 2. The pipeline encompasses four key stages: pre-training KSCLP with objectives specific to Knowledge-Specific and contrastive learning, fine-tuning KSCLP for bug report summarization within the Sequence-to-Sequence framework, and finally, evaluating its performance on the testing set.

A. MODEL ARCHITECTURE

In line with previous large language models such as BERT [11] and CodeBERT [13], our approach utilizes Transformer encoder layers, as detailed in [9], to construct the KSCLP model. To process a bug report sequence $br = \{t_1, t_2, \ldots, t_n\}$ with length n, we prepend and append two special tokens, resulting in $br = \{[CLS], t_1, t_2, \ldots, t_n, [EOS]\}$. Initially, we apply word embedding [5] and position embedding [34] to the sequence, yielding the vector representation of br:

$$V_{br} = \mathbb{E}(br) + \mathbb{P}(br), \tag{1}$$

where $\mathbb{E} \in \mathbf{R}^{d_e \times |\mathcal{V}|}$ and $\mathbb{P} \in \mathbf{R}^{d_e \times |\mathcal{L}|}$ denote lookup matrices that convert the br sequence into a vector. Here, d_e, $|\mathcal{V}|$, and $|\mathcal{L}|$ represent the embedding dimensions, the vocabulary size, and the maximum length of bug reports, respectively. Subsequently, we feed V_{br} into the core architecture of KSCLP, namely the Stacked Transformer Encoder Layers, and get the contextual representation of the bug report sequence:

$$C_{br} = \text{Trans}(V_{br}). \tag{2}$$

B. KNOWLEDGE-SPECIFIC PRE-TRAINING

In Fig. 2, the pre-training stage 1 illustrates a straightforward pipeline for training KSCLP using a Knowledge-Specific objective. This approach differs significantly from the masked language modeling technique described in [11], which randomly masks only 15% of tokens in a bug report sequence. In contrast, Knowledge-Specific masks 40% of the tokens, often targeting entire sentences or specific entities like API name or function name, which can guide the model to learn domain-specific knowledge from bug reports. KSCLP then attempts to reconstruct these masked sentences based on their context and the overall content of the bug report. Specifically, sentences in the bug report are masked using one of three methods:

- With a 70% probability, all tokens in a sentence are replaced with [MASK] tokens. This method challenges KSCLP to comprehend the intrinsic knowledge of the bug report to reconstruct the masked sentence.
- With a 15% probability, two tokens in the sentence are randomly swapped. This simulates real-world scenarios where, despite word rearrangements, the sentence's meaning remains comprehensible. This method aids KSCLP in modeling the contextual relationship of each sentence within the bug report.
- For the remaining probability, 15% of the tokens in a sentence are replaced with random tokens. This technique enables KSCLP to understand the context surrounding individual tokens in the bug report.

To further refine the contextual representation learning of bug reports using KSCLP, we incorporate a dynamic masking operation as described by Liu et al. [12] during the masking stage of each report's processing. Dynamic masking facilitates the variation in sentence masking within the same bug report across diverse pre-training iterations. This approach notably expands the scale of the pre-training corpus, thereby facilitating a more extensive acquisition of contextual knowledge by KSCLP. Consequently, KSCLP achieves a more precise understanding of the contextual nuances inherent in each bug report.

After the KSCLP model successfully extracts the contextual vector from the masked bug reports, it endeavors to predict the masked tokens. This prediction is based on maximizing the log-likelihood, which is formulated as follows:

$$\mathcal{L}MLM(\theta) = \sum i \in \mathcal{M} - \log p(t_i|\hat{S}) \tag{3}$$

In this equation, θ denotes the parameters learned by the KSCLP model. The symbol $\mathcal{M}$ represents the set of masked tokens. The probability function $p(\cdot)$ is modeled explicitly by KSCLP. Here, t_i signifies each individual masked token, and $\hat{S}$ refers to the sequence of remaining tokens present in the bug report.

Table 1 provides a comprehensive overview of the pre-training parameters for KSCLP. In alignment with established methodologies in the field, as documented

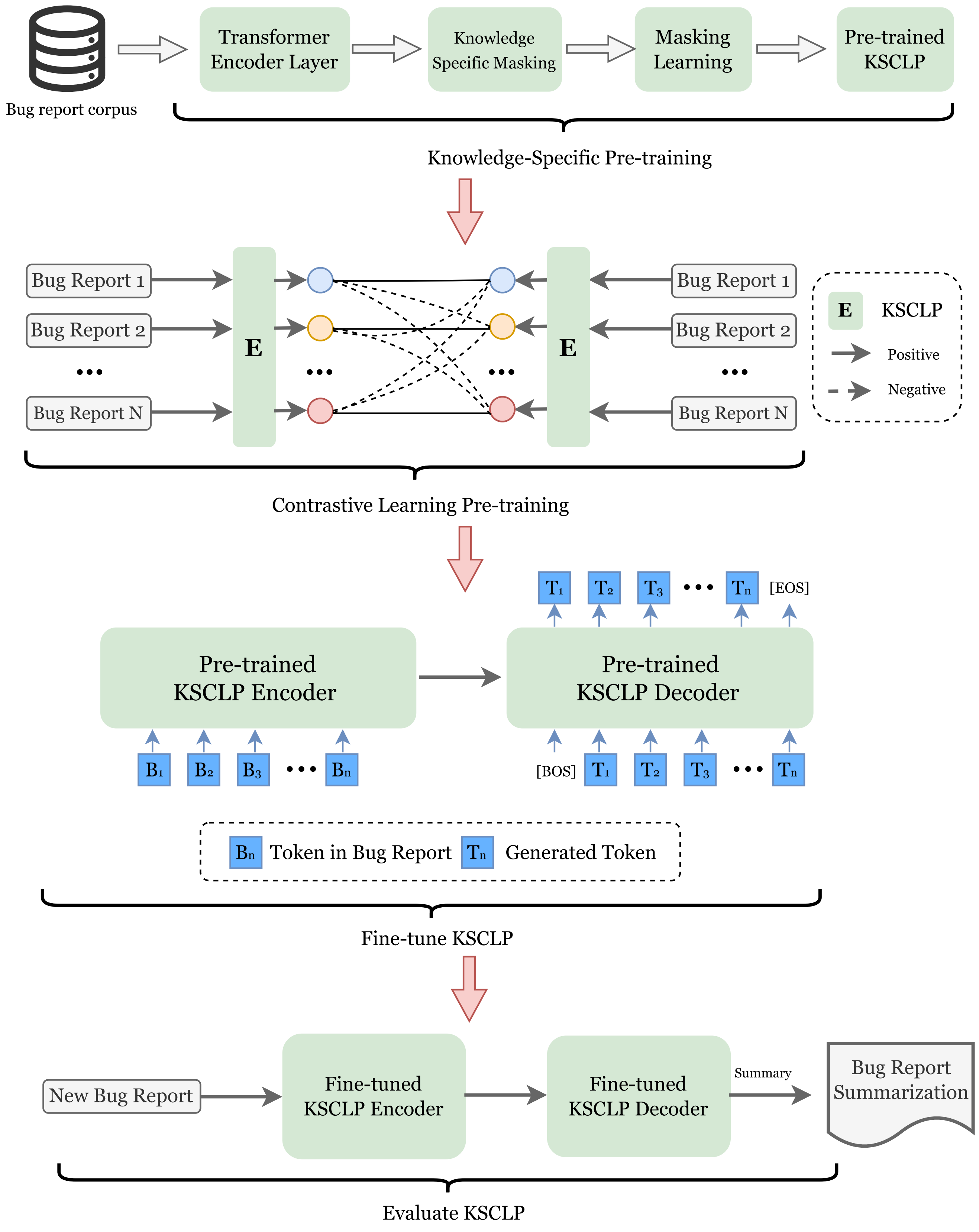


FIGURE 2. The pipeline of KSCLP.

TABLE 1. Statistics of parameters used in the pretraining stage.

Parameter	Introduction	Size
L	Transformer encoder layer	12
d_w, d_m	Hidden size	768
dff	Hidden size	3072
h	Attention heads	12
d_q, d_k, d_v	Attention head size	64
Dropout	Dropout rate	0.1
Learning rate	-	5e-5
Batch Size	Mini-batch training	32
Weight Decay	-	0.01
Max Steps	Training steps	400K
Learning rate decay	-	Linear
Adam ϵ	Parameter in Adam optimizer	1e-8
AdamW β_1	Parameter in AdamW optimizer	0.9
AdamW β_2	Parameter in AdamW	0.999

by Devlin et al. [11] and Liu et al. [12], we configured the hyperparameters of KSCLP as follows: the number of layers (L) is set to 12, the number of attention heads (h) is 12, the dimension of word embeddings (d_e) is 768, and the feed-forward/filter size (dff) is 3072.

For the optimization process, the AdamW optimizer [35] is employed, featuring a learning rate of 5×10^{-5}, with first and second moment decay rates (β_1 and β_2) set to 0.9 and 0.999, respectively. Additionally, we incorporate an L2 weight decay of 0.01 and implemented a linear decay strategy for the learning rate. The batch size is set to 32, and the maximum sequence length for bug report analysis is set to 512 tokens.

KSCLP underwent a pre-training regime spanning 40 epochs, leveraging the initial weight configuration from CodeBERT. This strategic choice of initialization facilitated KSCLP's enhanced capacity to assimilate and process the semantic intricacies inherent in source code as presented within bug reports.

C. CONTRASTIVE LEARNING PRE-TRAINING

In the second stage of pre-training, depicted in Figure 2, we illustrate the process of training the Knowledge Synthesis for KSCLP by using a contrastive learning objective. The KSCLP typically undergoes training in mini-batches. Within each batch, individual bug reports function as negative samples for one another. The principal challenge lies in generating positive samples for each bug report. To overcome this, we implement the dropout technique as described by Srivastava et al. [36] within the Transformer encoder layer, a strategy aimed at mitigating overfitting. This method involves the random deactivation of a subset of neurons in the neural network during each training cycle, thereby compelling the active neurons to independently adapt and learn relevant features. As a result, by inputting identical data into the Transformer encoder on two separate occasions, we acquire two semantically analogous but numerically distinct vectors. This approach simplifies the generation of positive samples for each bug report; each report merely needs to be processed twice through KSCLP. The contrastive learning objective is then optimized by minimizing the loss function expressed as follows:

$$\ell_i = -\log \frac{e^{\mathrm{sim}(p_i, p_i')/\tau}}{\sum j = 1^N e^{\mathrm{sim}(p_i, p_j)/\tau}} \tag{4}$$

In this equation, sim(·) represents the cosine similarity function. The variables p_i and $p_i{}'$ denote the contextual representations of the i-th bug report and its corresponding positive sample, respectively. The term τ refers to a temperature parameter that prevents the issue of gradient vanishing. Through this specialized pre-training process, the KSCLP is enabled to execute sequence-level modeling of bug reports, thereby deeply understanding the bug report from the perspective of developers.

1) PRE-TRAINING DETAILS

The pre-training of KSCLP using the contrastive learning objective largely adheres to the hyper-parameter settings established in the initial pre-training stage. However, we made specific adjustments to optimize performance: the batch size was set to 64, the learning rate was fine-tuned to 3e-5, and the number of training epochs was limited to 3. Prior to this phase of pre-training, KSCLP was initialized using the weights obtained from the first stage of pre-training. This strategic initialization aims to leverage foundational learning while adapting KSCLP to the nuances of the contrastive learning approach.

D. FINE-TUNING KSCLP FOR BUG REPORT SUMMARIZATION

After completing the pre-training phase, we integrate the pre-trained KSCLP within a Sequence-to-Sequence framework. This integration is specifically tailored for the task of bug report summarization. As depicted in Figure 2, we also use the KSCLP model as a decoder to generate summaries based on the contextual representations produced by the encoder. To fully leverage the knowledge acquired during pre-training, we implement parameter sharing between the encoder and decoder. This approach marks a departure from traditional methods, which typically involve selecting sentences from the bug report to create a summary. In contrast, our method synthesizes unique, concise, and precise summaries, thus enhancing the summarization process.

1) TRAINING DETAILS

For the fine-tuning of the KSCLP model, we modified several parameters: the batch size was set to 64, the learning rate was adjusted to 5×10^{-5}, and the number of training epochs was fixed at 10.

E. EVALUATING KSCLP

Upon refining the KSCLP model for bug report summarization, it is crucial to evaluate its efficacy using the test dataset. Figure 2 illustrates this evaluation process. For each new bug report in the test set, the model first processes it

TABLE 2. The statistics of the bug report in each project.

Project	Number of Bug Reports	Average Length
Mozillag	112,750	142.61
Eclipse	106,627	114.13
Netbeans	23,236	200.15
GCC	33,026	229.21
Overall	275,639	171.53

through an encoder to generate a contextual representation. This representation is then fed into a decoder, which generates the summary. Consistent with previous work [9], [23], our approach employs a beam search decoding algorithm with a beam size of 10.

IV. EXPERIMENTAL SETUPS

This section introduces the experimental frameworks employed in our study. Initially, we delineate the research questions that guide the investigation. Subsequently, we describe the dataset and baseline models used in the experiments. This is followed by a discussion of the evaluation metrics chosen to assess the baselines. Furthermore, we provide details about the experimental environment, introducing the configurations and tools utilized for conducting the experiments.

A. RESEARCH QUESTIONS

Our work focuses on the following three research questions (RQ):

- **RQ1:** How effective is KSCLP when compared with the baseline approaches?
- **RQ2:** How effective is KSCLP when compared with large language models?
- **RQ3:** How does each pre-training objective affect the performance of KSCLP?

Our first research question (RQ1) seeks to evaluate the efficacy of the KSCLP by comparing it with established baselines. This comparison aims to ascertain if KSCLP effectively assimilates domain-specific knowledge from bug reports and their overarching contextual information. Given that KSCLP is developed upon the foundation of LLMs, a secondary goal of our research is to determine whether KSCLP surpasses the performance of existing prominent LLMs, such as BERT [11] and CodeBERT [13]. This investigation forms the basis of our second research question (RQ2). Furthermore, to optimize the pre-training of KSCLP, we have devised a pair of targeted pre-training objectives. Consequently, our third research question (RQ3) delves into the impact of each specific pre-training objective on the overall efficacy of KSCLP.

B. DATASET AND BASELINES

1) DATASET

Our study employs the public bug report corpus curated by Fang et al. [1], encompassing over 270,000 bug reports from four renowned projects on BugZilla: Mozilla, Eclipse, Netbeans, and the GNU Compiler Collection (GCC). A comprehensive breakdown of this dataset is presented in Table 2. The corpus is partitioned into three subsets: 80% of the reports form the training set, 10% constitute the validation set, and the remaining 10% are allocated to the testing set, as detailed in Table 3. These subsets are integral to our experimental framework, where the training set is utilized for initial model pre-training, subsequent fine-tuning, and for training baseline methods. Performance evaluation of these baselines is conducted on the validation and testing sets. In our approach, the pre-training stage involves treating bug report summarization as an integral part of the input sequence. During fine-tuning, the model is fed with bug report sequences excluding their summarizations, and is tasked with generating these summaries autonomously.

TABLE 3. The statistics of the bug report in each set.

Project	Number of Bug Reports
Training Set	123,297
Validation Set	15,4475
Testing Set	15,510

2) BASELINE SELECTION FOR RQ1

In our study, we selected five leading-edge approaches in automated bug report summarization to evaluate the efficacy of KSCLP. These approaches are:

- **DeepSum** [4]: This approach leverages word embedding techniques to generate summaries.
- **BugSum** [7]: This method utilizes a Bi-directional Gated Recurrent Unit (Bi-GRU) architecture for summarization tasks.
- **PRHAN** [8]: PRHAN employs a hybrid attention network to enhance summarization quality.
- **Transformer** [9]: This well-known model is based on the self-attention network, known for its effectiveness in various generative tasks.
- **RTA** [1]: RTA utilizes advanced language modeling techniques for summarization.

Each of these approaches represents a significant advancement in the field, offering diverse methodologies and perspectives. By benchmarking KSCLP against these state-of-the-art techniques, we aim to provide a comprehensive assessment of its performance in the context of automated bug report summarization.

3) BASELINES IN RQ2

In this study, we selected three effective LLMs for further validation of the KSCLP's effectiveness. These models are BERT [11], RoBERTa [12], and CodeBERT [13]. Each of these models employs a design architecture based on stacked layers of the Transformer encoder, a notable feature in their structural composition.

C. EVALUATION METRICS

Following the prior work [1], [4], we choose composite BLEU (c.B.) and ROUGE-L (R.L) to measure the performance of each model, which are widely used in various generation tasks like code summarization [23], machine translation [6], and code translation [37].

BLEU score is computed as follows:

$$BLEU = BP \cdot \exp\left(\sum_{n=1}^{N} w_n \log(p_n)\right), \tag{5}$$

where BP represents a brevity penalty, introduced to counterbalance the inflated scores of excessively short generations. The factor w_n, assigned the value of $\frac{1}{N}$, serves as a weight for each n-gram's contribution to the overall score. The term p_n denotes the geometric mean of the modified precision for n-grams. Consistent with existing literature [23], we set N to 4. The BLEU score, which can range from 0 to 100, is a quantifiable indicator of generative accuracy, with higher scores corresponding to more precise generation.

In the evaluation of ROUGE-N, the F1 score is computed as follows:

$$F1_{\text{ROUGE-N}} = \frac{2 \times R_{\text{ROUGE-N}} \times P_{\text{ROUGE-N}}}{R_{\text{ROUGE-N}} + P_{\text{ROUGE-N}}}, \tag{6}$$

where $R_{\text{ROUGE-N}}$ and $P_{\text{ROUGE-N}}$ represent the recall and precision metrics for ROUGE-N, respectively. These metrics are defined as:

$$R_{rouge-n} = \frac{\sum_{(gen,ref)\in S} \sum_{gram_n \in ref} Cnt_{gen}(gram_n)}{\sum_{(gen,ref)\in S} \sum_{gram_n \in ref} Cnt_{ref}(gram_n)}, \tag{7}$$

$$P_{rouge-n} = \frac{\sum_{(gen,ref)\in S} \sum_{gram_n \in ref} Cnt_{gen}(gram_n)}{\sum_{(gen,ref)\in S} \sum_{gram_n \in gen} Cnt_{gen}(gram_n)}, \tag{8}$$

where gen, ref, and S denote the bug report summarization generated by the model, the ground truth, and the test set, respectively. The functions $Cnt_{gen}(gram_n)$ and $Cnt_{ref}(gram_n)$ quantify the frequency of occurrence of $gram_n$ in gen and ref, respectively. Analogous to the BLEU score, the ROUGE score varies from 0 to 100, with higher values indicating more effective summarization.

D. EXPERIMENTAL ENVIRONMENT

In this study, all experiments are conducted on a deep learning server equipped with two NVIDIA Tesla V100 GPUs, each with 32GB of memory. The implementation of the KSCLP and its training utilized a suite of Python packages: PyTorch version 2.0.1, transformers version 4.33.2 [38], and datasets version 2.14.5. For RQ2, baselines are publicly available large language models sourced from the Transformers Hub.[4] They are subsequently fine-tuned for the specific task of bug report summarization.

[4]https://huggingface.co/models

TABLE 4. The performance comparison between KSCLP and baselines on bug report summarization.

Model	R.1	R.2	R.L	c.B.
DeepSum	17.60	8.05	17.00	3.73
BugSum	25.91	11.66	24.69	5.81
PRHAN	23.95	10.71	22.14	5.18
Transformer	26.76	12.10	24.65	6.07
RTA	39.19	20.57	35.97	10.13
KSCLP	**41.33**	**22.02**	**37.89**	**12.05**

V. EVALUATION

A. ANSWER TO RQ1: EFFECTIVENESS COMPARISON BETWEEN KSCLP AND BASELINES

Table 4 offers a comparative analysis of the performance between our KSCLP model and established baseline methods. We employ various metrics for this comparison, including ROUGE-1 (R.1), ROUGE-2 (R.2), ROUGE-L (R.L), and composite BLEU (c.B.), as detailed in the table. The findings indicate a notable superiority of KSCLP over the baseline models across all metrics. Notably, DeepSum exhibits the least effective performance, attributable primarily to the fact that it is built solely based on the simple word embedding layer. In contrast, other baseline models demonstrate enhanced performance, underscoring the efficacy of advanced neural network architectures such as Bi-GRU, hybrid attention networks, and self-attention networks. Among these, RTA emerges as the most effective, highlighting the potency of advanced language models.

Distinguishing itself from these baseline approaches, KSCLP incorporates a Knowledge-Specific pre-training methodology. This approach enables the model to assimilate domain-specific knowledge from bug reports, thereby comprehensively understanding the semantic interplay between tokens. Additionally, KSCLP's contrastive learning objective is pivotal in acquiring a global contextual grasp of bug reports, along with their semantic variances. This aspect is crucial for effective sequence-level modeling. As a result, KSCLP not only achieves superior outcomes but also consistently outperforms the baseline models. One notable advantage of KSCLP lies in its utilization of stacked deep Transformer encoder layers, which are adept at modeling the semantic representations of entire bug report sequences. This capability is a significant leap from the capabilities of shallow neural networks such as Bi-GRU. While these networks are efficient in processing local semantic information, they falter in mining domain-specific knowledge and struggle with long-sequence modeling. KSCLP, with its advanced architecture, effectively overcomes these limitations.

B. ANSWER TO RQ2: EFFECTIVENESS COMPARISON BETWEEN KSCLP AND PRE-TRAINED LANGUAGE MODEL

In Table 5, we present a comparative analysis of the performance of our KSCLP model against three established pre-training language models: BERT, RoBERTa, and CodeBERT. As with RQ1, the performance metrics include

TABLE 5. The performance comparison between KSCLP and existing LLMs on bug report summarization.

Model	R.1	R.2	R.L	c.B.
BERT	24.68	7.91	21.36	0.73
RoBERTa	35.99	16.76	32.01	8.24
CodeBERT	35.07	16.30	30.86	7.74
KSCLP	**41.33**	**22.02**	**37.89**	**12.05**

TABLE 6. The impact of each pre-training objective on the performance of KSCLP. PT and KS denote pre-training and knowledge-specific, respectively.

Model	R.1	R.2	R.L	c.B.
KSCLP (w/o PT)	35.07	16.30	30.86	7.74
KSCLP (w/o KS PT)	38.99	20.22	34.77	9.96
KSCLP (Full PT)	**41.33**	**22.02**	**37.89**	**12.05**

ROUGE-1 (R.1), ROUGE-2 (R.2), ROUGE-L (R.L), and Composite BLEU (c.B.).

Upon examining the data in Table 5, it becomes evident that KSCLP outperforms the baseline large language models across all evaluated metrics. This outcome substantiates the efficacy of the pre-training strategies we have proposed. In particular, the Knowledge-Specific pre-training, tailored to the bug reports corpus, empowers KSCLP to assimilate domain-specific insights effectively. Consequently, KSCLP demonstrates a heightened ability to discern and internalize the nuanced relationships between bug reports and their corresponding summarization.

It is also noteworthy that, with the exception of BERT, the other large language models exhibit commendable performance, closely paralleling the baseline methodologies established in RQ1. This observation underscores the potential of large language models to develop profound contextual understandings of bug reports. Furthermore, our model's incorporation of a contrastive learning objective significantly contributes to KSCLP's capacity to grasp global semantic contexts within diverse bug reports. This aspect is instrumental in enhancing the model's proficiency in generating accurate and contextually relevant summarizations for given bug reports.

C. ANSWER TO RQ3: THE EFFECT OF DIFFERENT PRE-TRAINING OBJECTIVES

Table 6 presents a comparison of the KSCLP model's performance when pre-trained with various objectives. This study involves three primary conditions: fine-tuning KSCLP without any pre-training (akin to the baseline, CodeBERT), fine-tuning KSCLP with Knowledge-Specific pre-training, and finally, evaluating KSCLP following comprehensive pre-training. The results in Table 6 clearly demonstrate that pre-training KSCLP with a Knowledge-Specific objective yields a minimum improvement of 2 points across all evaluated metrics. More impressively, subjecting KSCLP to a full pre-training regimen not only enhances its performance further but also enables it to surpass all baseline models, thereby establishing new benchmarks in the field.

VI. THREATS TO THE VALIDITY

In this study, we acknowledge two principal threats to the validity of our research. The foremost threat pertains to internal validity, specifically focusing on the optimal configuration of hyper-parameters within the KSCLP framework. To address this issue, we adopt a methodical approach, grounding our hyper-parameter settings in the empirical evidence presented in seminal works such as Devlin et al., Liu et al., and Fang et al. [1], [11], [12]. These studies provide a robust foundation for determining the most effective parameter configurations, thereby enhancing the internal validity of our research.

The external validity of our study presents a potential limitation, particularly concerning the generalizability of KSCLP. Our evaluation of the KSCLP is confined to projects within the BugZilla platform. This restriction raises questions about the model's applicability and effectiveness in diverse bug-tracking environments, especially for systems other than BugZilla. Nevertheless, this limitation can be addressed through the application of transfer learning techniques [39]. By fine-tuning KSCLP on datasets from various projects, it is possible to adapt and extend the model's capabilities for bug report summarization across different bug-tracking systems. This approach enhances the model's versatility and broadens its potential application spectrum.

VII. CONCLUSION

In this study, we introduce KSCLP, an innovative approach to automated bug report summarization that synergizes large language models with contrastive learning techniques. KSCLP demonstrates superior performance over existing methods in this domain. This enhancement is attributable to its capacity for assimilating domain-specific knowledge and contextual nuances from bug reports, thereby achieving a thorough comprehension of diverse report contents. A key feature of KSCLP is its utilization of contrastive learning objectives, which facilitates the acquisition of profound semantic representations across various bug reports. To evaluate KSCLP's efficacy, we executed comprehensive comparative experiments using a widely recognized public dataset. The outcomes conclusively show that KSCLP surpasses all benchmarked methods in performance. In the future, we aim to augment KSCLP by integrating structural elements of bug reports and expanding its application across different software engineering tasks that involve bug report analysis.

REFERENCES

[1] S. Fang, T. Zhang, Y. Tan, H. Jiang, X. Xia, and X. Sun, "RepresentThemAll: A universal learning representation of bug reports," in *Proc. IEEE/ACM 45th Int. Conf. Softw. Eng. (ICSE)*, May 2023, pp. 602–614.

[2] S. Fang, Y.-S. Tan, T. Zhang, Z. Xu, and H. Liu, "Effective prediction of bug-fixing priority via weighted graph convolutional networks," *IEEE Trans. Rel.*, vol. 70, no. 2, pp. 563–574, Jun. 2021.

[3] S. G. Jindal and A. Kaur, "Automatic keyword and sentence-based text summarization for software bug reports," *IEEE Access*, vol. 8, pp. 65352–65370, 2020.
[4] X. Li, H. Jiang, D. Liu, Z. Ren, and G. Li, "Unsupervised deep bug report summarization," in *Proc. IEEE/ACM 26th Int. Conf. Program Comprehension (ICPC)*, May 2018, p. 144.
[5] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proc. Adv. Neural Inf. Process. Syst.*, 2013, pp. 1–9.
[6] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proc. Annu. Conf. Neural Inf. Process. Syst.*, Dec. 2014, pp. 3104–3112.
[7] H. Liu, Y. Yu, S. Li, Y. Guo, D. Wang, and X. Mao, "BugSum: Deep context understanding for bug report summarization," in *Proc. 28th Int. Conf. Program Comprehension*, Jul. 2020, pp. 94–105.
[8] S. Fang, T. Zhang, Y.-S. Tan, Z. Xu, Z.-X. Yuan, and L.-Z. Meng, "PRHAN: Automated pull request description generation based on hybrid attention network," *J. Syst. Softw.*, vol. 185, Mar. 2022, Art. no. 111160.
[9] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. Adv. Neural Inform. Process. Syst. (NIPS)*, 2017, pp. 5998–6008.
[10] X. Gu, H. Zhang, and S. Kim, "Deep code search," in *Proc. IEEE/ACM 40th Int. Conf. Softw. Eng. (ICSE)*, May 2018, pp. 933–944.
[11] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics, Hum. Lang. Technol.* Minneapolis, MI, USA: Association for Computational Linguistics, vol. 1, Jun. 2019, pp. 4171–4186.
[12] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "RoBERTa: A robustly optimized BERT pretraining approach," 2019, *arXiv:1907.11692*.
[13] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, and M. Zhou, "CodeBERT: A pre-trained model for programming and natural languages," 2020, *arXiv:2002.08155*.
[14] J. A. Khan, A. Yasin, R. Fatima, D. Vasan, A. A. Khan, and A. W. Khan, "Valuating requirements arguments in the online user's forum for requirements decision-making: The CrowdRE-VArg framework," *Softw., Pract. Exper.*, vol. 52, no. 12, pp. 2537–2573, Dec. 2022.
[15] T. Ullah, J. A. Khan, N. D. Khan, A. Yasin, and H. Arshad, "Exploring and mining rationale information for low-rating software applications," *Soft Comput.*, vol. 2023, pp. 1–26, Aug. 2023.
[16] Y. Li, T. Zhang, X. Luo, H. Cai, S. Fang, and D. Yuan, "Do pre-trained language models indeed understand software engineering tasks?" 2022, *arXiv:2211.10623*.
[17] L. Zhang, S. Wang, and B. Liu, "Deep learning for sentiment analysis: A survey," *Wiley Interdiscipl. Rev., Data Mining Knowl. Discovery*, vol. 8, no. 4, p. e1253, 2018.
[18] Y. Liu and M. Lapata, "Text summarization with pretrained encoders," in *Proc. Conf. Empirical Methods Natural Lang. Process. 9th Int. Joint Conf. Natural Lang. Process. (EMNLP-IJCNLP)*, 2019, pp. 3728–3738.
[19] T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, 2015, pp. 1412–1421.
[20] D. Guo, S. Ren, S. Lu, Z. Feng, D. Tang, S. Liu, L. Zhou, N. Duan, A. Svyatkovskiy, S. Fu, M. Tufano, S. K. Deng, C. Clement, D. Drain, N. Sundaresan, J. Yin, D. Jiang, and M. Zhou, "GraphCodeBERT: Pre-training code representations with data flow," 2020, *arXiv:2009.08366*.
[21] S. Fang, Y.-S. Tan, T. Zhang, and Y. Liu, "Self-attention networks for code search," *Inf. Softw. Technol.*, vol. 134, Jun. 2021, Art. no. 106542.
[22] D. Yuan, S. Fang, T. Zhang, Z. Xu, and X. Luo, "Java code clone detection by exploiting semantic and syntax information from intermediate code-based graph," *IEEE Trans. Rel.*, vol. 72, no. 2, pp. 511–526, Jun. 2022.
[23] X. Hu, G. Li, X. Xia, D. Lo, and Z. Jin, "Deep code comment generation," in *Proc. 26th Conf. Program Comprehension*, May 2018, p. 200.
[24] Z. Chen, S. Fang, and M. Monperrus, "Supersonic: Learning to generate source code optimizations in C/C++," 2023, *arXiv:2309.14846*.
[25] D. Li, W. E. Wong, M. Jian, Y. Geng, and M. Chau, "Improving search-based automatic program repair with neural machine translation," *IEEE Access*, vol. 10, pp. 51167–51175, 2022.
[26] S. Rastkar, G. C. Murphy, and G. Murray, "Automatic summarization of bug reports," *IEEE Trans. Softw. Eng.*, vol. 40, no. 4, pp. 366–380, Apr. 2014.
[27] H. Jiang, J. Zhang, H. Ma, N. Nazar, and Z. Ren, "Mining authorship characteristics in bug repositories," *Sci. China Inf. Sci.*, vol. 60, no. 1, p. 12107, Jan. 2017.
[28] R. Lotufo, Z. Malik, and K. Czarnecki, "Modelling the 'hurried' bug report reading process to summarize bug reports," *Empirical Softw. Eng.*, vol. 20, pp. 516–548, Jun. 2015.
[29] D. Arya, W. Wang, J. L. C. Guo, and J. Cheng, "Analysis and detection of information types of open source software issue discussions," in *Proc. IEEE/ACM 41st Int. Conf. Softw. Eng. (ICSE)*, May 2019, pp. 454–464.
[30] D. R. Radev, H. Jing, M. Styś, and D. Tam, "Centroid-based summarization of multiple documents," *Inf. Process. Manage.*, vol. 40, no. 6, pp. 919–938, Nov. 2004.
[31] X. Zhu, A. B. Goldberg, J. Van Gael, and D. Andrzejewski, "Improving diversity in ranking using absorbing random walks," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics*, 2007, pp. 97–104.
[32] Q. Mei, J. Guo, and D. Radev, "DivRank: The interplay of prestige and diversity in information networks," in *Proc. 16th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Jul. 2010, pp. 1009–1018.
[33] S. Mani, R. Catherine, V. S. Sinha, and A. Dubey, "AUSUM: Approach for unsupervised bug report summarization," in *Proc. ACM SIGSOFT 20th Int. Symp. Found. Softw. Eng.*, Nov. 2012, pp. 1–11.
[34] P. Shaw, J. Uszkoreit, and A. Vaswani, "Self-attention with relative position representations," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics, Hum. Lang. Technol.*, 2018, pp. 464–468.
[35] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," 2017, *arXiv:1711.05101*.
[36] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, pp. 1929–1958, Sep. 2014.
[37] X. Chen, C. Liu, and D. Song, "Tree-to-tree neural networks for program translation," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 31, 2018, pp. 1–11.
[38] T. Wolf et al., "Transformers: State-of-the-art natural language processing," in *Proc. Conf. Empirical Methods Natural Lang. Process., Syst. Demonstrations*, 2020, pp. 38–45.
[39] K. Weiss, T. M. Khoshgoftaar, and D. Wang, "A survey of transfer learning," *J. Big Data*, vol. 3, no. 1, pp. 1–40, May 2016.

YUNNA SHAO was born in Wenzhou, Zhejiang, China, in 1984. She received the master's degree from Tongji University. She is currently with the College of Artificial Intelligence, Zhejiang College of Security Technology. Her research interests include computer networks and information security.

BANGMENG XIANG was born in Wenzhou, Zhejiang, China, in 1984. He received the master's degree from Tongji University. He is currently with the College of Artificial Intelligence, Zhejiang College of Security Technology. His research interests include information security and cloud computing technology.