IEEE Access
Multidisciplinary : Rapid Review : Open Access Journal

Received 1 April 2024, accepted 2 May 2024, date of publication 6 May 2024, date of current version 10 June 2024.
Digital Object Identifier 10.1109/ACCESS.2024.3397326

RESEARCH ARTICLE

SumLLaMA: Efficient Contrastive Representations and Fine-Tuned Adapters for Bug Report Summarization

BANGMENG XIANG AND YUNNA SHAO
Zhejiang College of Security Technology, Wenzhou, Zhejiang 325000, China
Corresponding author: Yunna Shao (shaoyunna2024@163.com)

This work was supported in part by Wenzhou Municipal Science and Technology Plan Project under Grant R2023138, and in part by Wenzhou Philosophy and Social Sciences Planning Annual Topic under Grant 23WSK208YBM.

ABSTRACT In software maintenance, concise summaries of bug reports are crucial, significantly enhancing developer efficiency and ultimately improving software quality and user experience. Large language models (LLMs) have become the standard method for bug report summarization due to their powerful representation capabilities. However, LLM-based approaches face two primary challenges: accurately modeling the contextual relationships between various components within a bug report and the risk of overfitting when fine-tuning LLMs on datasets of limited size. To address these challenges, we propose a novel approach, SumLLaMA, which leverages contrastive learning pre-training and parameter-efficient fine-tuning. Contrastive learning pre-training is employed to construct contextual relations between components in a single bug report, enabling SumLLaMA to learn sequence-level representations. For parameter-efficient fine-tuning, we fine-tune a smaller adapter instead of the entire LLM, reducing the number of parameters trained to about 1/1500 of the original model, effectively mitigating the risk of overfitting. To evaluate the effectiveness of SumLLaMA, we compare it against five baseline models, including a state-of-the-art model, on a publicly available dataset. The experimental results show that SumLLaMA outperforms all baselines by up to 26.66, 17.10, and 24.01 points in ROUGE-1, ROUGE-2, and ROUGE-L metrics, respectively, achieving a state-of-the-art result for automated bug report summarization.

INDEX TERMS Bug report summarization, efficient fine-tuning, software maintenance, contrastive representation.

I. INTRODUCTION

Automated bug report summarization is crucial in the software maintenance lifecycle, especially with the increasing number of bug reports submitted to tracking systems [1]. Concise summaries of bug reports can quickly help developers assess the priority and severity of a bug [2], which improves the efficiency of bug assignment and fixing, further boosting the quality of software products and enhancing user experience. With the powerful learning capabilities of neural networks, deep learning techniques [3] have become the mainstream solution for automated bug report summarization. These approaches [4], [5] typically rely on the Sequence-to-Sequence framework [6] and utilize different neural network architectures to summarize bug reports in an encoder-decoder manner. The encoder transforms the bug report sequence into a semantic vector, from which the decoder generates the summary. However, these approaches often struggle to capture deep semantic features from the input due to the complexity of bug reports and the limitations of shallow network architectures.

Large language models (LLMs) [7], [8], currently one of the most effective deep learning techniques, are able to learn deep contextual representation due to their architecture, which stacks attention networks. Fang et al. [2] propose RTA, the first LLM pre-trained on their collected bug report

The associate editor coordinating the review of this manuscript and approving it for publication was Francisco J. Garcia-Penalvo.

corpus, which can be applied to various bug report-related tasks through fine-tuning. As a result, their fine-tuned RTA on the task-specific dataset achieves state-of-the-art results in the task of bug report summarization. To further improve the learning ability of LLMs, Shao and Xiang [9] propose KSCLP and perform Knowledge-Specific pre-training for LLM, which helps their model to learn domain-specific representation, improving the performance on the bug report summarization task.

Although LLM-based approaches are effective, they still face two primary limitations that compromise summarization quality in the context of bug reports. First, unlike the continuous code input in tasks like code search [10], clone code detection [11], program repair [12], and code optimization [13], bug reports comprise multiple distinct components. Consequently, a bug report sequence contains multiple sub-sequences derived from these components. However, current LLM-based approaches overlook the contextual relationship between these components, i.e., the semantic interaction between different sub-sequences. This oversight negatively impacts the summarization quality of bug reports. Second, in contrast to natural language processing (NLP) tasks such as machine translation, which have many large-scale publicly available datasets, the dataset scale for bug report summarization is relatively limited. As a result, LLMs may struggle to achieve full training on small-scale datasets, leading to a heightened risk of overfitting. Ultimately, this can significantly impair their ability to generalize to new datasets.

To tackle the challenges outlined above, we introduce a new approach based on LLMs for automated summarization of bug reports, which we refer to as SumLLaMA. This approach leverages the capabilities of LLMs combined with a custom pre-training objective and efficient fine-tuning strategies. For the first challenge, we propose a novel pre-training method for SumLLaMA using contrastive learning. Specifically, we generate multiple positive samples from a given bug report by masking different components. The pre-training objective is designed to enable the model to distinguish these positive samples from negative ones, thereby enhancing the semantic understanding of the relationships between various components of the bug report and finally achieving sequence-level contextual representation learning. To address the second challenge, we opt to fine-tune only a subset of the LLM's parameters by low-ranking adaption. We introduce a summarization adapter, which comprises only about 1/1500 of the original LLM's parameters. This approach is feasible due to the adequacy of the current dataset for fine-tuning such a small neural architecture for bug report summarization, allowing us to minimize the risk of overfitting while improving the generalization capabilities of SumLLaMA.

To evaluate the effectiveness of SumLLaMA, we carefully select four state-of-the-art baseline approaches: DeepSum [4], BugSum [5], PRHAN [14], and Transformer [15]. We then conduct extensive comparative experiments on an open-source dataset released by Fang et al. [2]. Following previous work, we divide the dataset into training, validation, and testing sets. We train our model using the training set and assess its performance on both the validation and testing sets. The experimental results reveal that SumLLaMA outperforms all baseline approaches in terms of ROUGE-1, ROUGE-2, and ROUGE-L scores, with improvements ranging from 2.93 to 24.01 points. This significant enhancement demonstrates the effectiveness of SumLLaMA. To further validate SumLLaMA's capabilities, we compare it with three open LLMs (i.e., BERT, RoBERT, and CodeBERT) [16], [17], [18] and two LLM-based bug report summarization approaches (i.e., KSCLP and RTA). The experimental results show that SumLLaMA consistently outperforms all these baseline approaches by at least 2.93 points on all metrics, indicating its superior summarization ability.

In this paper, we make the following contributions,

- We introduce SumLLaMA, a novel approach based on large language models (LLMs) for automated bug report summarization. To enhance summarization quality, we devise a customized pre-training objective that accounts for the distinctive characteristics of bug reports. Furthermore, we develop an effective fine-tuning strategy to mitigate the risk of overfitting due to the dataset's limited size.
- We present a series of experiments to verify the effectiveness of SumLLaMA. The experimental results demonstrate that SumLLaMA performs better than all baseline approaches.
- To further substantiate the effectiveness and efficiency of SumLLaMA, we compare it with three open-source LLMs and two LLM-based bug report summarization methods. The results support the superior summarization capabilities of our approach.

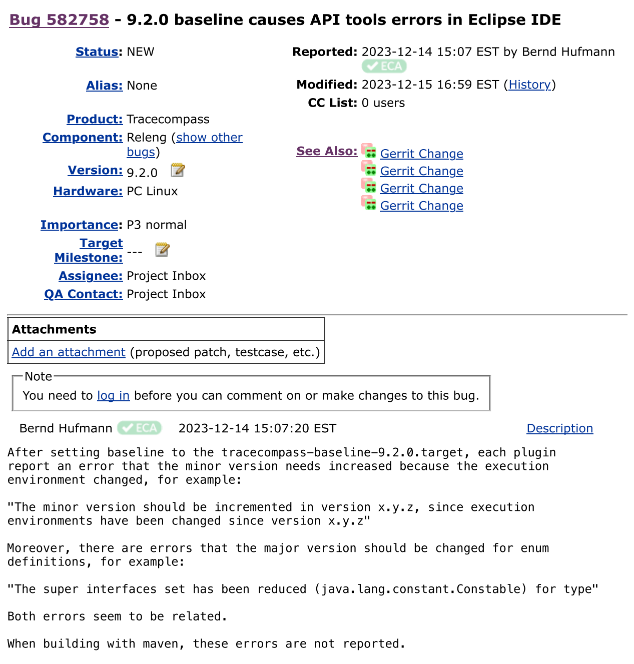
The remainder of this paper is organized as follows. Section II elaborates on the background knowledge, and Section III presents the construction of SumLLaMA. Section IV and Section V describe the experimental setups and results, respectively. Section VII discusses the threats to validity. Finally, we conclude this paper and point out the future work in Section VIII.

II. BACKGROUND

A. BUG REPORTS

Figure 1 presents a bug report from the Eclipse project,[1] 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 developers in resolving bugs. For example, the Description component provides comprehensive details about the bug, including its symptoms, effects, and impact on software performance. The Summary component offers a succinct overview of the problem, enabling developers to

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



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 [9].

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. According to previous work [1], [2], we consider five components ("Summary", "Product", "Component", "Description", "Importance") in the pre-training stage. This approach helps to fully establish the contextual relationship between different components. In the fine-tuning stage, we remove the "Summary" component from the input to train the model to generate it based on the remaining four components.

B. LARGE LANGUAGE MODEL

Large language models (LLMs) [7], [19], developed through unsupervised pre-training on large-scale corpora, have shown remarkable capability in learning universal data representations, leading to their proficiency in a wide array of NLP tasks, including sentiment analysis [20], text summarization [21], and machine translation [22]. This attribute has spurred the adaptation of LLMs for various software engineering tasks, such as code search [23], clone code detection [11], code summarization [24], code optimization [13], and program repair [25]. By leveraging their powerful representation learning capabilities, these models, pre-trained on massive software engineering datasets like source code from GitHub and subsequently fine-tuned for specific tasks, have achieved state-of-the-art results in various software engineering tasks, demonstrating the versatility and effectiveness of LLMs in both natural language and software engineering domains.

C. PARAMETER-EFFICIENT FINE-TUNING

As the scale of LLMs continuously increases [7], they exhibit increasingly powerful capabilities in various tasks. However, the large scale of LLMs also makes it difficult to fine-tune for specific tasks due to the high computational resource requirements. For example, recent work reported that fine-tuning the LLaMA-7B model [7] with all parameters requires more than 100GB peak GPU memory when the batch size is set to the smallest value. Consequently, although modern LLMs are powerful, fine-tuning them on task-specific datasets is a significant challenge. Parameter-efficient fine-tuning methods, such as Low-Rank Adaptation (LoRA) [26], can fine-tune parts of the parameters of LLMs. Specifically, LoRA fine-tunes low-rank matrices, i.e., an adapter, rather than the parameters in LLMs. In this way, LoRA can fine-tune 1/1500 of LLMs' parameters without affecting the fine-tuning performance. In our work, considering that fine-tuning the full parameters of LLMs on the current dataset of bug report summarization might lead to a serious overfitting risk, we propose applying LoRA, a parameter-efficient fine-tuning method, to the task of bug report summarization. There are two advantages: 1) Training a few parameters can effectively alleviate the overfitting risk. 2) It is possible to train LLMs on the limited GPU resources.

D. RELATED WORK

Early approaches for bug report summarization relied heavily on information retrieval techniques. Rastkar et al. [27] and Jiang et al. [28] employed classifiers to select relevant sentences from bug reports, with their effectiveness largely dependent on the quality of the training corpus [29]. Arya et al. [30] proposed a method for categorizing comments based on their content, enabling users to choose sentences that satisfy their specific needs. Radev et al. [31] introduced a novel approach that compresses sentences into vectors by using TF-IDF values and selecting sentences similar to the centroid of all sentence vectors. Further, Zhu et al. [32] and Mei et al. [33] explored sentence selection based on reference relations, with Mani et al. [34] refining this technique through a noise removal strategy.

As deep learning techniques have become increasingly powerful, Li et al. [4] propose the first deep learning-based approach for bug report summarization, which aims to minimize the inclusion of controversial sentences in summaries through neural networks. Subsequently, Fang et al. [2] introduce the first generative approach to generate summaries for bug reports using a sequence-to-sequence method. Specifically, they first pre-train an LLM on their collected data to learn domain-specific representations, and then further fine-tune the model for bug report summarization in a supervised way. Shao and Xiang [9] propose KSCLP, an approach that improves LLM-based summarization by incorporating knowledge-specific and contrastive learning pre-training. They designed two novel pre-training objectives to enable the model to learn the inner knowledge of bug reports in the pre-training stage.

Unlike the approaches mentioned earlier, SumLLaMA is designed with two distinctive considerations: Firstly, bug reports consist of several distinct components, and establishing the contextual relationship between these components enhances the model's ability to learn sequence-level representations. Secondly, acquiring a high-quality, large-scale, task-specific dataset for fine-tuning LLMs for bug report summarization is challenging, making parameter-efficient fine-tuning more practical than full-parameter fine-tuning.

III. APPROACH

In this section, we elaborate on the pipelines of SumLLaMA for summarizing bug reports, which is shown in Figure 2. The pipeline is composed of three core components: contrastive learning-based pre-training, parameter-efficient fine-tuning, and finally, inference.

A. MODEL INTRODUCTION

Due to the significant performance of recent decoder-only Large LLMs [7], [35], [36], we select CodeLLaMA [37] as the foundational model for this work. CodeLLaMA, distinguished by its pre-training on over 4TB of open-source corpus (equivalent to 500B tokens after pre-processing and tokenization), adopts the GPT-3 [38] model architecture. It utilizes the Transformer decoder layer, with a notable modification: the replacement of post-normalization with pre-normalization. This adjustment enables CodeLLaMA to normalize the input of each Transformer layer rather than the output. In contrast to previous LLMs that employed absolute positional embedding [16], [17], CodeLLaMA introduces rotary positional embedding [39]. This relative position embedding technique allows the model to learn deep positional relations, enhancing its understanding of context. The pre-training of CodeLLaMA involves the use of an AdamW optimizer, coupled with a cosine learning rate schedule, weight decay, gradient clipping, and warmup. These strategies collectively contribute to the model's optimal performance. Upon completion of pre-training, CodeLLaMA demonstrates exceptional zero-shot learning capabilities, outperforming all open-source LLMs and achieving competitive results with OpenAI's GPT-3.5 [37]. Moreover, fine-tuned CodeLLaMA [12] achieves state-of-the-art results on program repair, further underscoring its potential in advancing the field of software engineering.

B. CONTRASTIVE LEARNING PRE-TRAINING

In Fig. 2, we first construct positive and negative samples for contrastive learning pre-training. For samples in a mini-batch of size n, each sample is considered a negative sample with respect to the others, resulting in $n-1$ negative samples for each. The challenge then lies in constructing positive samples for a given sample. Given that bug reports consist of multiple distinct components, we construct positive samples by masking different components and adopt the following criteria for masking:

- For a given bug report, only one component is masked. Therefore, for a bug report with m components, we can generate m positive samples, effectively enlarging the original dataset by a factor of m.
- We choose five components in bug reports to build a corresponding sequence, including "Summary", "Product", "Component", "Description", and "Importance".
- When masking a component, if it is shorter than 15

1) With a 70
2) With a 15
3) For the remaining probability, the selected token is replaced with a random token.

After these processing steps, we successfully construct positive samples for each bug report. We then design a Siamese network for contrastive learning pre-training. This Siamese network consists of two CodeLLaMA models with shared parameters. For a mini-batch denoted as $\mathbf{B}$, we construct positive samples for the bug reports within it, resulting in a derived mini-batch $\mathbf{B}'$. Subsequently, these two mini-batches are fed into the CodeLLaMA models in the Siamese network, obtaining the contextual vector representations for each bug report. Finally, we use the following contrastive learning objective to pre-train the model:

$$\ell_i = -\log \frac{e^{\mathrm{sim}(\mathbf{B}_i, \mathbf{B}'_i)/\tau}}{\sum_{j=1}^{N} e^{\mathrm{sim}(\mathbf{B}_i, \mathbf{B}'_j)/\tau}} \tag{1}$$

In the above equation, $\mathrm{sim}(\cdot)$ represents the cosine similarity function. The variables B_i and $B_i{}'$ denote the contextual representations of the i-th bug report and its corresponding positive sample, respectively, and τ refers to a temperature parameter that mitigates the issue of gradient vanishing. By adopting this contrastive learning objective, we encourage the model to identify the positive sample derived from the original bug report sequence. This training process has two advantages: (1) By masking one component in each positive sample, the model is encouraged to fully build the contextual interaction between the masked component and other components, and (2) upon completing the pre-training, the model achieves a comprehensive understanding of the contextual interaction among all components, facilitating the learning of sequence-level contextual representation for bug reports.

Table 1 provides a comprehensive overview of the pre-training parameters for SumLLaMA, a novel LLM-based approach for automated bug report summarization. Aligning with the configuration of CodeLLaMA [37], we set the hyperparameters of SumLLaMA as follows: 32 layers, 32 attention heads, a word embedding dimension of 4096, an attention head size of 256, and an intermediate representation size of 11008. For optimization, we employ the AdamW optimizer [40] with a learning rate of 1×10^{-4}, and decay rates for the first and second moments (β_1 and β_2) set to 0.9 and 0.95, respectively. We also implement a cosine

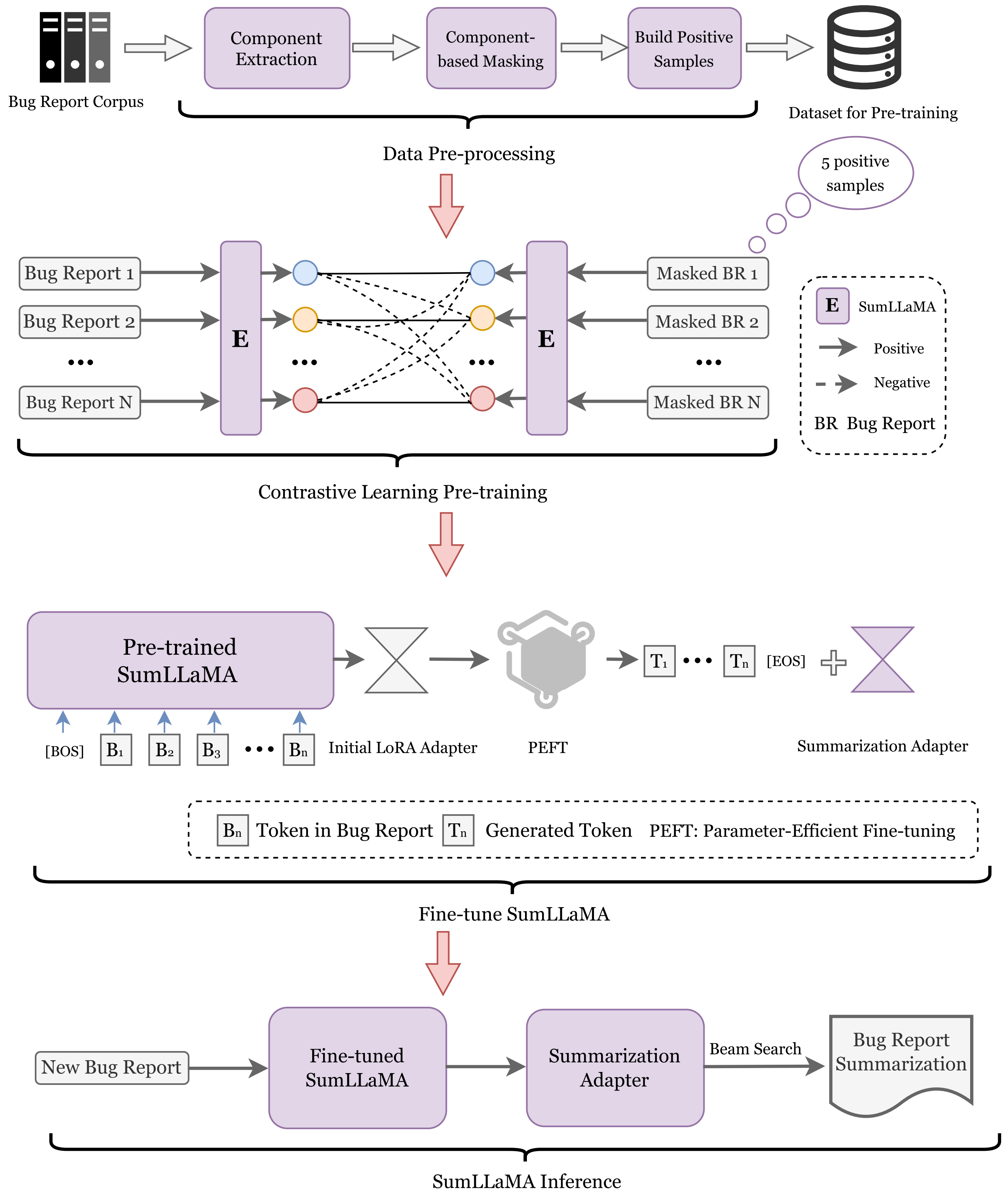


FIGURE 2. The pipeline of SumLLaMA.

decay strategy for the learning rate with a warmup step of 8000. The batch size is 32, and the maximum sequence length for the bug reports is 300 tokens. SumLLaMA undergoes a pre-training regime spanning 10 epochs, leveraging the

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

Introduction	Size
Transformer decoder layer	32
Embedding size	4096
Hidden size	11008
Number of attention heads	32
Attention head size	256
Vocabulary size	32016
Learning rate	1e-4
Batch size	32
Training epoch	10
Learning rate decay	Cosine
Warmup steps	8000
Adam ϵ	1e-8
AdamW β_1	0.9
AdamW β_2	0.95
Max input length	300

initial weight configuration from CodeLLaMA. Instead of pre-training all parameters of SumLLaMA, we utilize LoRA to perform parameter-efficient pre-training.

C. PARAMETER-EFFICIENT FINE-TUNING OF SUMLLAMA

After completing the contrastive learning pre-training, we apply the pre-trained SumLLaMA to bug report summarization through supervised fine-tuning. In this phase, we construct the input by omitting the "Summary" component from the bug report used in the pre-training stage. The model is then fine-tuned to generate summarizations in an auto-regressive manner, which involves predicting each token sequentially based on the preceding context. Specifically, for a given bug report sequence of tokens $X = x_1, x_2, \ldots, x_{n-1}$, the model aims to predict the probability of the next summarization token x_n. The probability of the next token, given the preceding tokens, is computed as follows:

$$\mathcal{L}\text{summary} = \sum_{n=1}^{N} \log P(x_n|x_{<n}; \theta), \tag{2}$$

where $\mathcal{L}$summary is the loss function for summarization, N is the total number of tokens in the sequence, $x < n$ denotes the sequence of tokens before position n, and θ represents the parameters of the model. $P(x_n|x_{<n}; \theta)$ is the probability of the token x_n given the context $x_{<n}$, as predicted by the model with parameters θ. During fine-tuning, the model's parameters θ are updated to maximize the likelihood of correctly predicting the next token in a large corpus of text. This objective enables the model to learn domain-specific representations tailored to the task of bug report summarization.

Due to the large scale of CodeLLaMA (a language model for code) and the limited dataset available for bug report summarization, we employ Low-Rank Adaptation (LoRA) to perform parameter-efficient fine-tuning. Instead of fine-tuning all the parameters, SumLLaMA focuses on fine-tuning a low-rank adapter, specifically a summarization adapter, which contains approximately 1/1500 of the CodeLLaMA parameters. The core concept of LoRA is to approximate updates to the weight matrices in the transformer layers (such as the attention and feed-forward layers) using low-rank matrices. More specifically, for a weight matrix $W \in \mathbb{R}^{d\times d}$ in a transformer layer, the update is approximated as follows:

$$\Delta W = UV^T, \tag{3}$$

where $U \in \mathbb{R}^{d\times r}$ and $V \in \mathbb{R}^{d\times r}$ are low-rank matrices, and $r \ll d$ is the rank of the low-rank approximation. The rank r is a hyperparameter that controls the number of trainable parameters. During fine-tuning, only the parameters in U and V are updated, while the original weight matrix W remains fixed. The updated weight matrix W' used in the forward pass is then given by:

$$W' = W + \alpha\Delta W = W + \alpha UV^T, \tag{4}$$

where α is a scaling factor that controls the magnitude of the update.

Training Details: For the fine-tuning of SumLLaMA, we configured the settings as follows: a batch size of 16, a learning rate of 5×10^{-4} with cosine decay, a training epoch of 2, and a maximum output length of 30. For LoRA, we set the rank to 8, α to 16, and the dropout rate to 0.05. Additionally, we connected the adaptation matrices to q_{proj} and v_{proj} in each Transformer decoder layer.

D. INFERENCE

Figure 2 illustrates the inference process after fine-tuning. For each new bug report in the test set, we input it into the fine-tuned SumLLaMA. The summarization adapter within SumLLaMA then processes this input, converting it into a semantic vector representation. This representation serves as the basis for generating the summary, which is constructed token by token. Following the approach of Hu et al. [24], we employ a beam search decoding strategy to produce the summary. Specifically, we set the beam size to 10. This means that for each bug report, SumLLaMA generates 10 candidate summaries, allowing for a diverse set of potential summarizations to be considered before selecting the most appropriate one.

IV. EXPERIMENTAL SETUPS

In this section, we present the experimental setup, which includes the research questions we aim to investigate, the dataset and baseline models for comparison, the evaluation metrics, and the details of the experimental environment, such as the configurations and tools used to conduct the experiments.

A. RESEARCH QUESTIONS

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

- **RQ1:** How effective is SumLLaMA when compared with the baseline approaches?
- **RQ2:** How effective is SumLLaMA when compared with LLM-based approaches?

TABLE 2. The statistics of the original bug report corpus.

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

TABLE 3. The dataset used for experiments.

Dataset	Data Size
Training Set	123,297
Validation Set	15,4475
Testing Set	15,510

- **RQ3:** How do our proposed strategies contribute to the quality of bug report summarization?

The research goal of RQ1 is to evaluate the SumLLaMA's effectiveness in summarizing bug reports, thus we compare it with state-of-the-art approaches for this task. By performance comparison, we aim to verify that our proposed pre-training and fine-tuning strategies could help LLMs build task-specific representations for the task of bug report summarization. Since SumLLaMA is constructed based on LLMs, the research goal of RQ2 is to determine whether our approach surpasses other LLM-based approaches, such as BERT [16], CodeBERT [18], RTA [2], and KSCLP [9]. By comparing the performance of each model, we aim to further verify that SumLLaMA could learn more effective task-specific representations for the task of bug report summarization. Furthermore, the research goal of RQ3 is to verify the effectiveness of our proposed pre-training and fine-tuning strategies. Consequently, we design the corresponding ablation experiment to evaluate the contribution of per proposed method.

B. DATASET AND BASELINES

1) DATASET

Our study employs the public bug report corpus curated by Fang et al. [2], which encompasses over 270,000 bug reports from four renowned projects on BugZilla: Mozilla, Eclipse, Netbeans, and the GNU Compiler Collection (GCC). A comprehensive breakdown of the dataset, including the distribution of reports across projects and their average length, 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. The training set is utilized for initial model pre-training, subsequent fine-tuning, and training baseline methods. Performance evaluation of these baselines is conducted on the validation and testing sets. In the pre-training stage, we utilize five components, i.e., "Summary", "Product", "Component", "Description", and "Importance", to build the input sequence. As for the fine-tuning stage, we build the input sequence by removing the "Summary component", and then we fine-tune the model to generate corresponding summarization.

2) BASELINE SELECTION FOR RQ1

To achieve the research goal of RQ1, we select four state-of-the-art approaches to verify the SumLLaMA's effectiveness, which is the same as the previous work [9]. The details of these baselines are shown as follows,

- **DeepSum** [4]: It is the first approach to utilize neural networks, i.e., word2vec [41], for the task of bug report summarization.
- **BugSum** [5]: This approach further improves the performance of **DeepSum** by connecting word2vec with Bi-directional gated recurrent unit.
- **PRHAN** [14]: It is constructed by stacking hybrid attention networks that can learn global and local context relations of the bug report sequence.
- **Transformer** [15]: It is built by stacking self-attention networks and its effectiveness is verified by various generative tasks.

By comparing SumLLaMA against these state-of-the-art baseline approaches, we could perform an overall evaluation of the effectiveness of SumLLaMA.

Baselines in RQ2 In RQ2, we select five effective LLMs to further evaluate the effectiveness of SumLLaMA on the task of bug report summarization. The first three models are general LLMs, i.e., BERT [16], RoBERTa [17], and CodeBERT [18]. To apply them for bug report summarization, we fine-tune them on our built datasets. Moreover, we also choose two LLM-based approaches for bug report summarization, i.e., RTA and KSCLP, both of which achieve state-of-the-art results on the task of bug report summarization. For RTA, we fine-tune it for bug report summarization, and for KSCLP, we directly evaluate it on the testing set since it is already fine-tuned for the task of bug report summarization.

C. EVALUATION METRICS

Building on previous studies [2], [4], we employ composite BLEU (c.B.) and ROUGE-L (R.L) as metrics to evaluate model performance, commonly used in tasks like code summarization [24], machine translation [6], and code translation [42].

The BLEU score is defined as:

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

where BP is a brevity penalty, $w_n = \frac{1}{N}$ is the weight for each n-gram, and p_n is the geometric mean of modified n-gram precision. Following [24], we set $N = 4$. The BLEU score ranges from 0 to 100, with higher scores indicating better accuracy.

TABLE 4. The performance comparison between SumLLaMA against baseline approaches.

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
SumLLaMA	**44.26**	**25.15**	**41.01**	**15.12**

TABLE 5. The performance comparison between SumLLaMA and LLM-based approaches 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
RTA	39.19	20.57	35.97	10.13
KSCLP	41.33	22.02	37.89	12.05
SumLLaMA	**44.26**	**25.15**	**41.01**	**15.12**

For ROUGE-N evaluation, the F1 score is:

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

with recall $R_{\text{ROUGE-N}}$ and precision $P_{\text{ROUGE-N}}$ defined as:

$$R_{\text{ROUGE-N}} = \frac{\sum_{(gen,ref)\in S} \sum_{gram_n \in ref} Cnt_{gen}(gram_n)}{\sum_{(gen,ref)\in S} \sum_{gram_n \in ref} Cnt_{ref}(gram_n)}, \quad (7)$$

$$P_{\text{ROUGE-N}} = \frac{\sum_{(gen,ref)\in S} \sum_{gram_n \in ref} Cnt_{gen}(gram_n)}{\sum_{(gen,ref)\in S} \sum_{gram_n \in gen} Cnt_{gen}(gram_n)}, \quad (8)$$

where *gen* and *ref* are the model-generated summary and ground truth, respectively, and S is the test set. The ROUGE score, like BLEU, ranges from 0 to 100, with higher values indicating better summarization.

D. EXPERIMENTAL ENVIRONMENT

In this study, all experiments were conducted on a deep learning server equipped with four NVIDIA Tesla A100 GPUs, each with 40GB of memory. The implementation of SumLLaMA and its training utilized a suite of Python packages, including PyTorch version 2.0.1, transformers version 4.33.2 [43], and datasets version 2.14.5. For the second research question (RQ2), we used publicly available LLMs sourced from the Transformers Hub.[2] These models are then fine-tuned for bug report summarization on the training set.

V. EVALUATION

A. ANSWER TO RQ1: EFFECTIVENESS COMPARISON

Table 4 presents the experimental results and performance comparison between SumLLaMA and state-of-the-art baseline approaches. We employ four metrics for evaluation: 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 that SumLLaMA significantly outperforms the baseline models across all metrics. Specifically, SumLLaMA surpasses the best baseline model, the Transformer, by at least 9.05 points in terms of Composite BLEU. Among the baseline approaches, DeepSum exhibits the worst performance. This is because it is built by solely utilizing a simple word embedding layer. In contrast, other baseline approaches demonstrate enhanced performance, highlighting the effectiveness of advanced neural network architectures such as Bi-GRU, hybrid attention networks, and self-attention networks.

Unlike conventional approaches that utilize shallow neural architectures, SumLLaMA employs a Large Language Model (LLM)-based approach. LLMs are typically constructed by stacking multiple layers of Transformer encoders and decoders, resulting in a deep and powerful architecture. To further enhance the performance of SumLLaMA, we implement improvements in both the pre-training and fine-tuning stages. In the pre-training phase, we enhance SumLLaMA's capability to construct representations by incorporating contrastive learning. This technique strengthens the contextual interaction between components extracted from bug reports, leading to more effective sequence-level modeling. During the transition to the fine-tuning phase, we adopt parameter-efficient training by integrating Low-Rank Adaptation (LoRA). This approach requires training only a small subset of parameters, enabling comprehensive training on datasets of limited size. Compared to the traditional full-parameter fine-tuning of LLMs, SumLLaMA exhibits superior generalization capabilities. Ultimately, the integration of these two training strategies enables SumLLaMA to achieve state-of-the-art performance in the task of bug report summarization, setting a new benchmark in the field.

B. ANSWER TO RQ2: EFFECTIVENESS COMPARISON AGAINST LLMS

Table 5 presents experimental results and performance comparison of SumLLaMA against general LLMs and LLM-based bug report summarization approaches. Aligning with RQ1, we utilize four performance metrics to measure these models, including ROUGE-1 (R.1), ROUGE-2 (R.2), ROUGE-L (R.L), and Composite BLEU (c.B.).

Upon analyzing the experimental results in Table 5, it becomes evident that LLM-based approaches outperform all general LLMs. This superior performance is attributed to the additional pre-training on bug report corpora, enabling the model to learn domain-specific representations. Moreover, KSCLP incorporates knowledge-specific pre-training to grasp the inner knowledge of bug reports, thus surpassing the fine-tuned RTA. Despite the high performance of these approaches, SumLLaMA still significantly outperforms them, with a margin ranging from 2.93 to 3.13 points. This improvement is due to our two proposed methods

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

TABLE 6. The impact of our proposed methods on the performance of SumLLaMA. PEFT and CLP denote parameter-efficient fine-tuning and contrastive learning pre-training.

Model	R.1	R.2	R.L	c.B.
SumLLaMA (w/o CLP)	38.53	19.91	34.36	8.28
SumLLaMA (w/o PEFT)	42.63	23.92	39.69	13.33
SumLLaMA	**44.26**	**25.15**	**41.01**	**15.12**

that enhance LLMs in both the pre-training and fine-tuning phases. During pre-training, we establish inner semantic relations among different bug report components by designing a task-specific contrastive learning objective. In the fine-tuning stage, we employ parameter-efficient fine-tuning to thoroughly train core parameters for bug report summarization. Although CodeLLaMA-7B is much larger than the LLMs used in previous bug report summarization models, our trainable parameters are approximately 1/25 of theirs. Consequently, with the limited scale of the bug report corpus, SumLLaMA receives adequate training and effectively avoids the risk of overfitting. Owing to these advanced methodologies, SumLLaMA outperforms all LLM-based approaches and further enhances the quality of automated bug report summarization.

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

Table 6 presents an ablation study through a comprehensive comparison of our proposed two training methods. We establish two primary comparative groups with different training strategies: fine-tuning SumLLaMA solely with the parameter-efficient method and fine-tuning SumLLaMA with contrastive learning pre-training. Subsequently, we compare these two derived versions against the original SumLLaMA. The results in Table 6 unequivocally demonstrate the necessity of performing domain-specific pre-training for applying LLMs to a specific task. Specifically, SumLLaMA, with our tailored contrastive learning pre-training, can outperform the state-of-the-art KSCLP model. More impressively, even when we directly fine-tune a small subset of the parameters of SumLLaMA using the parameter-efficient method, it still achieves results close to those of RTA, despite the latter being pre-trained on a bug report corpus. When we integrate these two methods, SumLLaMA finally achieves state-of-the-art results and further improves the quality of bug report summarization.

VI. DISCUSSION

SumLLaMA can be utilized for various software maintenance activities associated with bug reports, extending beyond merely summarizing them. Firstly, our method for training a summarization adapter using an open-source dataset can be customized to fit any specific development context or company-specific dataset. Secondly, once a SumLLaMA model is adequately trained, we envisage equipping developers with a maintenance assistant. This assistant would provide pertinent information at various stages of software maintenance. Although the integration of such a system is more aligned with engineering practices and is beyond the scientific purview of our study, it could prove beneficial for companies adopting a SumLLaMA-like approach.

VII. THREATS TO THE VALIDITY

In this study, we identify two potential threats to the validity of our research. The first threat concerns the optimal configuration of hyperparameters for our chosen parameter-efficient method, namely LoRA. To mitigate this issue, we adopt the same settings as used in previous work, where it was demonstrated that these settings enable the model to achieve state-of-the-art results in a specific task. This selection effectively addresses the threat of internal validity to our research.

The external validity, or generalizability, of our study is a concern due to the exclusive use of data from the Bugzilla platform for SumLLaMA. When applied to bug reports from other platforms, SumLLaMA may encounter generalization issues. However, we propose two methods to mitigate this threat: parameter-efficient training and transfer learning [44]. Parameter-efficient training allows us to train a small subset of the model's parameters on a limited dataset, enhancing its generalization capability. Additionally, transfer learning enables us to fine-tune SumLLaMA on a user-customized dataset, adapting its representation to the specific domain.

VIII. CONCLUSION

In this study, we introduce SumLLaMA, a novel LLM-based approach for bug report summarization. Specifically, we propose a novel contrastive learning objective for SumLLaMA's pre-training. This objective enables the model to learn the contextual interactions between different components extracted from bug reports, thereby building effective sequence-level representations. Subsequently, we introduce a parameter-efficient training method to fine-tune a small subset of SumLLaMA's parameters. This method allows for full training of this parameter subset on a limited dataset scale, enhancing the model's generalization. To evaluate SumLLaMA's effectiveness, we conducted comprehensive comparative experiments against traditional and LLM-based approaches. Our results show that SumLLaMA significantly outperforms all baseline approaches. In the future, we aim to further explore how to efficiently and effectively apply LLMs to automate various activities in software maintenance. Additionally, we will also explore how to apply the LLM-based approach to real-world software maintenance.

REFERENCES

[1] 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.

[2] 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.

[3] E. Akleman, "Deep learning," *Computer*, vol. 53, no. 9, p. 17, Sep. 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, pp. 144–155.

[5] 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*, 2020, pp. 94–105.

[6] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proc. 28th Conf. Neural Inf. Process. Syst. (NIPS)*, 2014, pp. 3104–3112.

[7] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample, "LLaMA: Open and efficient foundation language models," 2023, *arXiv:2302.13971*.

[8] Y. Li, T. Zhang, X. Luo, H. Cai, S. Fang, and D. Yuan, "Do pre-trained language models indeed understand software engineering tasks?" *IEEE Trans. Softw. Eng.*, vol. 49, no. 2023, pp. 4639–4655, Oct. 2023.

[9] Y. Shao and B. Xiang, "Towards effective bug report summarization by domain-specific representation learning," *IEEE Access*, vol. 12, pp. 37653–37662, 2024.

[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] 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.

[12] A. Silva, S. Fang, and M. Monperrus, "RepairLLaMA: Efficient representations and fine-tuned adapters for program repair," 2023, *arXiv:2312.15698*.

[13] Z. Chen, S. Fang, and M. Monperrus, "Supersonic: Learning to generate source code optimizations in C/C++," 2023, *arXiv:2309.14846*.

[14] 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.

[15] 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. 31st Conf. Neural Inf. Process. Syst. (NIPS)*, 2017, pp. 5998–6008.

[16] J. Devlin, M. 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. (NAACL-HLT)*, 2019, pp. 4171–4186.

[17] 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*.

[18] 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," in *Findings of the Association for Computational Linguistics*. Association for Computational Linguistics, 2020, pp. 1536–1547.

[19] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "ALBERT: A lite BERT for self-supervised learning of language representations," 2020, *arXiv:1909.11942*.

[20] 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, 2018, Art. no. e1253.

[21] 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.

[22] T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," in *Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP)*, 2015, pp. 1412–1421.

[23] 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.

[24] X. Hu, G. Li, X. Xia, D. Lo, and Z. Jin, "Deep code comment generation," in *Proc. IEEE/ACM 26th Int. Conf. Program Comprehension (ICPC)*, 2018, pp. 200–210.

[25] N. Jiang, T. Lutellier, and L. Tan, "CURE: Code-aware neural machine translation for automatic program repair," in *Proc. 43rd Int. Conf. Softw. Eng.*, 2021, pp. 1161–1173.

[26] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "LoRA: Low-rank adaptation of large language models," 2021, *arXiv:2106.09685*.

[27] 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.

[28] 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, 2017, Art. no. 12107.

[29] R. Lotufo, Z. Malik, and K. Czarnecki, "Modelling the 'hurried' bug report reading process to summarize bug reports," *Empirical Softw. Eng.*, vol. 20, no. 2, pp. 516–548, Jun. 2015.

[30] 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.

[31] 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, 2004.

[32] X. Zhu, A. B. Goldberg, J. Van Gael, and D. Andrzejewski, "Improving diversity in ranking using absorbing random walks," in *Proc. Hum. Lang. Technol., Conf. North Amer. Chapter Assoc. Comput. Linguistics, Main Conf.*, 2007, pp. 97–104.

[33] 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*, 2010, pp. 1009–1018.

[34] 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.*, 2012, pp. 1–11.

[35] J. Achiam et al., "GPT-4 technical report," 2023, *arXiv:2303.08774*.

[36] J. Bai et al., "Qwen technical report," 2023, *arXiv:2309.16609*.

[37] B. Rozière et al., "Code llama: Open foundation models for code," 2023, *arXiv:2308.12950*.

[38] T. Brown et al., "Language models are few-shot learners," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 1877–1901.

[39] J. Su, M. Ahmed, Y. Lu, S. Pan, W. Bo, and Y. Liu, "RoFormer: Enhanced transformer with rotary position embedding," *Neurocomputing*, vol. 568, Feb. 2024, Art. no. 127063.

[40] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," 2017, *arXiv:1711.05101*.

[41] T. Mikolov et al., "Distributed representations of words and phrases and their compositionality," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 26, 2013.

[42] 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.

[43] 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.

[44] K. Weiss, T. M. Khoshgoftaar, and D. Wang, "A survey of transfer learning," *J. Big Data*, vol. 3, no. 1, pp. 1–40, 2016.

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.

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.

• • •