RepresentThemAll: A Universal Learning Representation of Bug Reports

Sen Fang
Macau University of Science and Technology
Macau, China
fangsen1996@gmail.com

Tao Zhang *
Macau University of Science and Technology
Macau, China
tazhang@must.edu.mo

Youshuai Tan
Macau University of Science and Technology
Macau, China
tanyoushuai123@gmail.com

He Jiang
Dalian University of Technology
Dalian, China
jianghe@dlut.edu.cn

Xin Xia
Huawei
Hangzhou, China
xin.xia@acm.org

Xiaobing Sun
Yangzhou University
Yangzhou, China
xbsun@yzu.edu.cn

2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE) | 978-1-6654-5701-9/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICSE48619.2023.00060

***Abstract*—Deep learning techniques have shown promising performance in automated software maintenance tasks associated with bug reports. Currently, all existing studies learn the customized representation of bug reports for a specific downstream task. Despite early success, training multiple models for multiple downstream tasks faces three issues: complexity, cost, and compatibility, due to the customization, disparity, and uniqueness of these automated approaches. To resolve the above challenges, we propose `RepresentThemAll`, a pre-trained approach that can learn the universal representation of bug reports and handle multiple downstream tasks. Specifically, `RepresentThemAll` is a universal bug report framework that is pre-trained with two carefully designed learning objectives: one is the *dynamic masked language model* and another one is a contrastive learning objective, "*find yourself*". We evaluate the performance of `RepresentThemAll` on four downstream tasks, including duplicate bug report detection, bug report summarization, bug priority prediction, and bug severity prediction. Our experimental results show that `RepresentThemAll` outperforms all baseline approaches on all considered downstream tasks after well-designed fine-tuning.**

I. Introduction

Over the past decades, researchers have performed massive efforts to automate the process of software maintenance, for improving software development efficiency. Considering that developers generally utilize bug reports to conduct software maintenance activities, researchers thus design a series of automated software maintenance tasks associated with bug reports (ASMT-ABR for short), such as bug priority prediction [1], [2], bug severity prediction [3], [4], duplicate bug report detection [5], [6], and bug report summarization [7], [8].

Current studies mainly use deep learning techniques (i.e., neural networks) to construct their models for ASMT-ABR [2], [5], [6], [9]–[12]. For example, *Fang et al.* [2] carefully selected five elements in the bug report and concatenated them, then utilized the graph convolutional network [13] to learn the corresponding vector representation and performed automated bug priority prediction. In duplicate bug report detection, *Budhiraja et al.* [5] combined the textual elements of bug reports (e.g., summary and description) as a single sample and used a word embedding network [14] to learn the semantic representation of each sample. Afterward, they further exploited a deep fully-connected neural network to learn the probability distribution of duplicate and non-duplicate bug reports. Following by *Budhiraja et al.*, *He et al.* [6] improved the performance of duplicate bug report detection by introducing dual-channel convolutional neural networks [15]. *Li et al.* combined auto-encoder network and encoder-decoder framework to automatically generate a summary of the bug report according to its description. As for bug localization, DeepLoc [11] utilized a word embedding network to represent words in bug reports and utilized a convolutional neural network to capture semantic features, then automatically locate the buggy file.

Despite the aforementioned studies having obtained early success, they also face an important challenge: existing approaches are designed for a specific ASMT-ABR and they cannot serve for multiple downstream tasks in software maintenance. According to the prior studies [16], [17], there are multiple tasks associated with bug reports in the bug resolution process of software maintenance. **However, training multiple models to automate multiple downstream tasks in software maintenance may face three issues: complexity, cost, and compatibility, due to the customization, disparity, and uniqueness of these automated approaches**. As indicated in the left Fig. 1, it is a meaningful vision, that is, employing existing approaches to build an automated software maintenance system. From the figure, we can observe that such a system needs to parallelize multiple automated models because of existing approaches' customization, resulting in a relatively high **complexity**. Due to the automated approaches' disparity, we need to conduct targeted pre-processing for bug reports before passing them to the model. The above process appears to be reasonable, but training multiple automated models for multiple ASMT-ABR requires much **cost** of training resources, storage, and service, and may cause the waste of resources. Additionally, **compatibility** is a problem in the training pro-

* Tao Zhang is the corresponding author.

1558-1225/23/$31.00 ©2023 IEEE
DOI 10.1109/ICSE48619.2023.00060

Authorized licensed use limited to: Macau Univ of Science and Technology. Downloaded on July 14,2023 at 21:35:40 UTC from IEEE Xplore. Restrictions apply.

cess caused by the automated approaches' uniqueness. This is because there may exist version inconsistency or package inconsistency in different approaches, which requires developers' extra effort to upgrade all source code (e.g., migrate code from PyTorch to TensorFlow). Therefore, the above analysis provokes us to investigate: **To what extent, can we use a universal bug report representation to handle multiple ASMT-ABR?**

In order to resolve the aforementioned problems, we propose a universal approach that can be used for multiple ASMT-ABR. As illustrated in the right Fig. 1, its core contains two parts: one is a fundamental model to learn the universal representation of bug reports, and another one is the fine-tuning module that applies the learned representation of bug reports for different ASMT-ABR. Hence, the fundamental model needs to be capable enough to learn the context knowledge of bug reports and fully understand them, providing effective semantic representation to the fine-tuning module. To achieve our goal, we build a fundamental model, namely `RepresentThemAll`, by the combination of the pre-trained language model (PLM) [18]–[20] and contrastive learning [21]–[24]. Specifically, `RepresentThemAll` first is pre-trained by the *dynamic masked language model* objective in a self-supervised way, to learn the contextual information of each word in bug reports according to its appearing context. Afterward, we design a siamese network [25] based contrastive learning objective, "*find yourself*", to further pre-train `RepresentThemAll` in a self-supervised way. By contrastive learning pre-training, `RepresentThemAll` can model the contextual representation of bug reports at the sequence level, which helps learn the semantic differences between bug reports, improving the effect of the subsequent fine-tuning.

After finishing pre-training, we build two different fine-tuning modules: one is designed for classification tasks, such as duplicate bug report detection [6] and bug priority prediction [2], and another one is designed for the generation task, like bug report summarization [7]. For the classification task, we can connect `RepresentThemAll` with an extra classification layer and perform the corresponding fine-tuning. As for the generation task, we combine `RepresentThemAll` with the Seq2Seq framework [26], to build an end-to-end generative architecture.

To evaluate the performance of `RepresentThemAll`, we compare it with the baseline approaches on four ASMT-ABR, i.e., bug report summarization, duplicate bug report detection, bug priority prediction, and bug severity prediction. Specifically, we collected a large-scale bug report dataset that contains more than 250,000 bug reports, then split it into the training set, validation set, and test set according to 80%/10%/10%. We use the training set to pre-train `RepresentThemAll`. When finishing pre-training, we further utilize the training set to fine-tune `RepresentThemAll` on bug report summarization, bug priority prediction, and bug severity prediction tasks, then evaluate `RepresentThemAll` on the validation and test sets. Since duplicate bug report detection requires duplicate/non-duplicate pairs of bug reports to perform the experiments, we thus conduct all the relevant experiments on the Open Office dataset released by *Lazar et al.* [27]. The experimental results show that `RepresentThemAll` achieves state-of-the-art results among all four tasks. Moreover, we also compare `RepresentThemAll` with other PLMs, and experimental results demonstrate that `RepresentThemAll` is more effective for ASMT-ABR.

To sum up, we make the following contributions:

- We propose `RepresentThemAll`, a *novel* and *fundamental* approach that is pre-trained by the dynamic masked language model and contrastive learning objectives in a self-supervised way, it thus can learn the universal representation of bug reports.
- Pre-trained `RepresentThemAll` can be applied to downstream tasks in software maintenance (i.e., ASMT-ABR) by supervised fine-tuning. To our knowledge, we are the first to design an approach that can serve multiple downstream tasks in software maintenance.
- We evaluate `RepresentThemAll` on four downstream tasks and the experimental results show that it achieves state-of-the-art results on all considered tasks.
- **Data Availability**. We release pre-trained `RepresentThemAll` in Hugging Face Transformers[1]. Besides, we introduce how to fine-tune `RepresentThemAll` by some executable examples in our GitHub repository[2].

Paper Organization. The remainder of this paper is organized as follows. Section II describes the background and related work. Section III elaborates on the framework of the proposed `RepresentThemAll`. Section IV introduces the experimental setup, including research questions, dataset, baselines, evaluation metrics, and experimental environment. Section V presents the experimental results and analysis. Section VI discusses the open questions as well as threats to validity. Finally, Section VII concludes this paper and points out the future directions.

II. Background and Related Work

A. Bug Reports

A bug report is a specific report that contains information about what is wrong and where developers should fix the given bug, which is submitted to bug tracking systems (e.g., Bugzilla, JIRA) by users or developers and is beneficial for software maintenance. Considering the circle of the software maintenance [16], bug reports have the potential to improve the efficiency of developers when tackling a newly reported bug. For example, bug severity and priority in bug reports [2], [28] can reduce the time cost of bug assignment while bug localization tools [29]–[31] can reduce the time of bug fixing. As shown in Fig. 2, a bug report is composed of multiple elements, such as Summary, Description, Comment, Priority, and Severity. Generally, many studies [1], [2], [32] tend to use

[1]https://huggingface.co/Colorful/RTA

[2]https://github.com/ICSE-2023/RepresentThemALL

Authorized licensed use limited to: Macau Univ of Science and Technology. Downloaded on July 14,2023 at 21:35:40 UTC from IEEE Xplore. Restrictions apply.

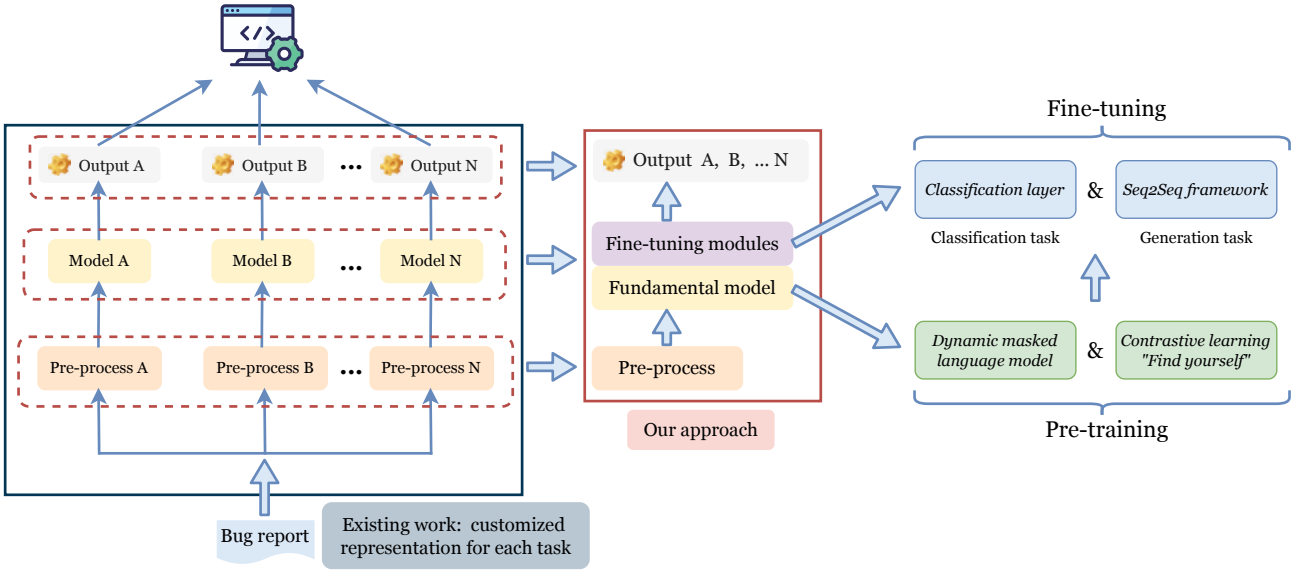


Fig. 1. The comparison of automating the process of software maintenance between existing approaches (left part) and our approach (right part).

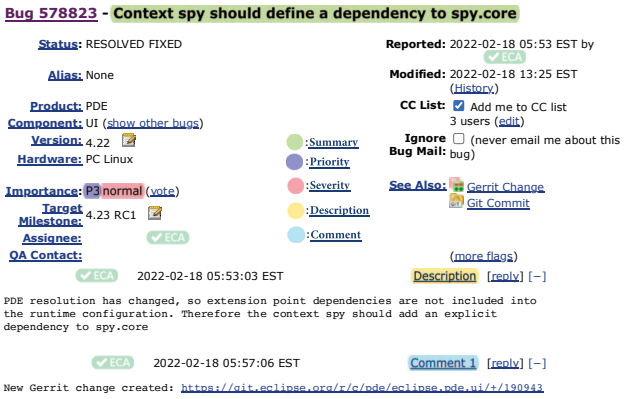


Fig. 2. An example of Eclipse bug report.

elements with enriched textual information, such as Summary and Description, to perform various of ASMT-ABR.

B. Contrastive Learning

The core concept of contrastive learning [33] is to learn a representation that can pull semantically similar sentences together and push the semantically dissimilar sentences apart. Contrastive learning training thus can enable the model to fully learn the semantic difference between sentences. Suppose that there is a set of paired sentences $\mathcal{S} = \{(s_i, s_i^+)\}_{i=1}^n$, where s_i and s_i^+ are semantically similar sentence pair. Following the contrastive learning framework proposed by *Chen et al.* [21], the training objective in contrastive learning can be defined as follows:

$$\ell_i = -\log \frac{e^{\text{sim}(r_i, r_i^+)}}{\sum_{j=1}^{N} e^{\text{sim}(r_i, r_j^+)}} \quad (1)$$

where r_i and r_i^+ are the representation of s_i and s_i^+, $\text{sim}(a, b)$ is the cosine similarity $\frac{a^T b}{||a|| \cdot ||b||}$, and N is the batch size. For a sentence pair in the batch, its negative samples are other sentence pairs in this batch.

C. Related Work

a) ***ASMT-ABR:*** `INSPect` [28] leveraged $BM25F_{ext}$ method to find top-k nearest neighbors of a newly submitted bug report in historical labeled bug reports, then used these k nearest neighbors to automatically predict the severity for the new bug. `DRONE` is the first NN-based model to achieve automated bug priority prediction. `DWEN` [5] can be used to detect duplicate bug reports by measuring their semantic similarity. Additionally, `DeepSum` [7] is an NN-based tool that can automatically generate titles for bug reports, and `DEMIBUD` [34] can detect miss information (e.g., expected behavior and steps to reproduce) in bug descriptions. Bug reports can also be used in fixer recommendation [35], bug localization [30], and program repair [36]. *Wu et al.* proposed `ST-DGNN`, which combines joint random walks and graph neural network, and can automatically recommend suitable fixers for new bugs. *Xiao et al.* [11] proposed `DeepLoc` which performs bug localization by learning semantic information of bug reports. *Koyuncu et al.* [37] proposed `iFixR`, which is a bug report driven program repair tool.

The above approaches learn the customized representation of bug reports and solve a specific ASMT-ABR. By contrast, `RepresentThemAll` aims to learn the universal represen-

Authorized licensed use limited to: Macau Univ of Science and Technology. Downloaded on July 14,2023 at 21:35:40 UTC from IEEE Xplore. Restrictions apply.

tation of bug reports and handle multiple ASMT-ABR by connecting the fine-tuning module.

*b) **PLMs in Software Engineering Field**:* As the success of PLMs in the natural language processing (NLP) community, PLMs have been widely utilized in the software engineering field. For example, *Alon et al.* [38] proposed a structural language model of code, which is pre-trained on paths of the abstract syntax tree and performs the arbitrary code completion for Java and C# code. *Feng et al.* [19] proposed `CodeBERT`, which is pre-trained on CodeSearchNet Corpus [39] and can be fine-tuned for code related tasks, e.g., code search [40], [41] and code summarization [42], [43]. PLMs are also used to perform program repairs. For instance, *Jiang et al.* [44] proposed `CURE`, a code-aware model for automatic program repair. Before training `CURE` on the dataset of program repair, they first pre-trained it on a large-scale software corpus to learn general contextual representation for code, their model thus achieved state-of-the-art results.

*c) **PLMs Related to Bug Report**: Ardimento et al.* [45] predicted bug-fixing time by directly fine-tuning `BERT` on their dataset. *Li et al.* [46] proposed `ARB-BERT`, which performs aging-related bug report classification by fine-tuning `BERT`. Except for the above two tasks, *Isotani et al.* [12] detected duplicate bug reports by fine-tuning `Sentence-BERT`. *Bo et al.* [47] achieved automatically bug question answering by fine-tuning `BERT` on their collected dataset. *Henao et al.* [48] proposed `M-BERT`, which is fine-tuned on `BERT` and effectively distinguishes feature requests and bug reports in user comments.

Different from the above approaches that only fine-tune the existing PLMs in the NLP field for a specific ASMT-ABR, `RepresentThemAll` is pre-trained on the bug report corpus to learn the universal representation of bug reports, serving multiple ASMT-ABR. In detail, it can be applied for different ASMT-ABR by connecting with suitable fine-tuning modules.

III. Approach

In this section, we first introduce how we perform text encoding for bug reports. Then, we describe the pipeline of `RepresentThemAll`, which can be seen in the right of Fig. 1. Specifically, it includes input & output representation, model architecture, dynamic masked language model, contrastive learning model, and fine-tuning process.

A. Text Encoding

Before feeding the input to `RepresentThemAll`, we need to build a vocabulary and use it to perform the text encoding to bug reports, transforming them into a set of numerical sequences. Following the literature [49], we use Byte-level BPE (BBPE) to build the vocabulary. BPPE uses UTF-8 byte n-grams to encode the text, by which a sentence is encoded into a set of byte n-grams. Considering that UTF-8 encoding contains 256 basic bytes, BBPE vocabulary has no OOV tokens. There are other methods to build vocabulary. For example, a simple method for building the vocabulary is to count the unique words in the training corpus (word-level vocabulary), but it may produce a large vocabulary and cannot solve the out-of-vocabulary (OOV) problem [50]. The former may affect the learning ability of neural networks because the large vocabulary causes the data-sparse problem [51], and the latter limits the generalizability of the model since words in other corpora may not appear in the built vocabulary. Instead of building a word-level vocabulary, *Sennrich et al.* [50] proposed byte-pair encoding (BPE), decomposing words into a set of character n-grams (subword units) and building subword-level vocabulary, and the OOV words can be represented by the combination of subword units.

BPE can alleviate the OOV problem, however, Unicode characters may account for a sizeable portion of BPE vocabulary when modeling large and diverse corpora [49], [52], which affects the performance of model learning. In contrast, Unicode characters can be encoded into 1-4 bytes by BBPE. Hence, BBPE is able to effectively control the level of the rare word decomposition and symbol sharing across different languages [49]. This helps us build the vocabulary for bug reports since they contain both natural language and code, owning more Unicode characters.

B. Input and Output Representation

To facilitate the use of `RepresentThemAll`, we insert two tokens into each bug report sequence, namely $S = \{\texttt{[CLS]}, w_1, w_2, ..., w_n, \ \texttt{[EOS]}\}$, where n is the length of the bug report sequence. `[CLS]` is a special token in front of the bug report sequence and we regard its final state representation as the aggregated sequence representation that contains the semantic information of the whole bug report, which can be used in classification tasks or contrastive learning training. For each bug report sequence, we obtain its input representation by summing up the corresponding word embedding [14] and position embedding [53]. In the word embedding, we utilize a lookup table $E_w \in \mathbb{R}^{d_e \times |\mathcal{V}|}$ to map each token in S into a vector $e_w \in \mathbb{R}^{d_e}$, where d_e is the embedding dimension and $|\mathcal{V}|$ is the vocabulary size. In the position embedding, we use another lookup table $E_p \in \mathbb{R}^{d_e \times \mathcal{L}}$ to map the ordinal position of each token in S into a vector $p_x \in \mathbb{R}^D$, where $\mathcal{L}$ is the max length of the sequence. Therefore, the input representation of S can be computed as follows:

$$S_i = E_w(S) + E_p(S). \tag{2}$$

The output representation of `RepresentThemAll` contains two parts: 1) the universal contextual representation of each token in the bug report sequence; 2) the contextual representation of `[CLS]`.

C. Model Architecture

The model architecture of `RepresentThemAll` is stacked with 12 bidirectional Transformer encoder layers [53]. As shown in Fig. 3, it is one Transformer encoder layer, which is mainly constructed by a multi-head self-attention network and a feed-forward network (FFN). Specifically, the input representation S_i is projected into Q, K, and V vectors

Authorized licensed use limited to: Macau Univ of Science and Technology. Downloaded on July 14,2023 at 21:35:40 UTC from IEEE Xplore. Restrictions apply.

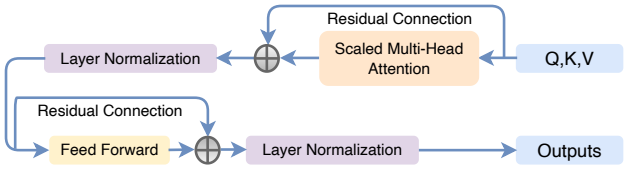


Fig. 3. Transformer encoder layer.

by three individual and learned linear projections, which are shown as follows:

$$Q = S_i W^Q, K = S_i W^K, V = S_i W^V, \tag{3}$$

where Q, K, and $V \in \mathbb{R}^{d_e}$. Then, we can compute the output of the self-attention network (SAN) as:

$$\text{SAN}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_e}}). \tag{4}$$

To obtain the output of multi-head SAN, we can repeat the Eq. 3 and Eq. 4 h times with different linear projections. In this situation, h is the number of attention heads, and we denote the dimension of Q, K, and V as d_q, d_k, and d_v, all of whom are equal to d_e/h. Hence, the output of multi-head SAN is computed as follows:

$$O_M = \text{MultiHead}(Q, K, V) = \text{Concat}(head_1, ..., head_h)W^O, \tag{5}$$

where Concat is a concatenation operation and the projection $W^O \in \mathbb{R}^{hd_v \times d_e}$. head_h is calculated as follows:

$$\text{head}_h = \text{SAN}(S_i W_h^Q, S_i W_h^K, S_i W_h^V), \tag{6}$$

where the projections $W_h^Q \in \mathbb{R}^{D_e \times d_q}, W_h^K \in \mathbb{R}^{d_e \times d_k}$, and $W_h^V \in \mathbb{R}^{d_e \times d_v}$. Afterward, we employ a residual connection [54] to the multi-head SAN, followed by a layer normalization [55], which can be expressed as:

$$O_N = \text{LayerNorm}(\text{inputs} + O_M) \tag{7}$$

where O_N is the output of the normalization layer. Next, the normalization layer is followed by a fully connected FFN, which includes two linear projections and a ReLU activation function. The output of FFN is computed as follows:

$$O_F = \text{ReLU}(O_N W_1)W_2 + b_2, \tag{8}$$

where parameter matrices $W_1 \in \mathbb{R}^{d_e \times d_{ff}}$ and $W_2 \in \mathbb{R}^{d_{ff} \times d_e}$. d_{ff} is equal to $4d_e$. Finally, the output of the encoder layer can be expressed as follows:

$$\text{outputs} = \text{LayerNorm}(O_N + O_F) \tag{9}$$

D. Dynamic Masked Language Model

Fig. 4 gives a pipeline of `RepresentThemAll` pre-trained with the masked language model objective. Given a bug report sequence S as the input, we first select a random set of tokens in S to mask out. Specifically, we replace these tokens with a special `[MASK]` token. Following `BERT` [18],

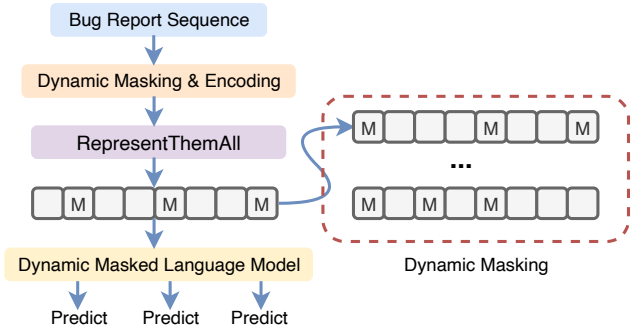


Fig. 4. `RepresentThemAll` predicts masked tokens according to their context. "M" denotes the masked token. Due to the dynamic masking, masked tokens in the bug report sequence are different at each epoch.

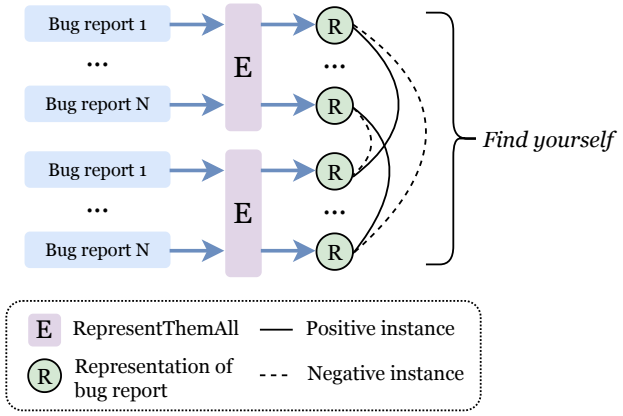


Fig. 5. Siamese `RepresentThemAll` network with shared parameters predicts the input bug report itself from in-batch negative bug reports. N is the batch size.

we select 15% of the tokens in S to mask out by replacing them with the following three ways:

- the `[MASK]` token with the 80% probability;
- a random token with the 10% probability;
- the unchanged token with the 10% probability.

Considering that our dataset is still small (compared with the pre-trained corpus in NLP community), following `RoBERTa` [52], we take a dynamic masking strategy to mask each bug report 10 times so that each bug report is masked in 10 different ways. In other words, we expand the original training set 10 times in the pre-training stage, which helps `RepresentThemAll` fully learn the contextual information of every token in bug reports.

When feeding the input to `RepresentThemAll`, we present an objective to train it, called dynamic masked language modeling. It enables `RepresentThemAll` to predict the original token from the masked token according to its context. Different from the standard autoregressive language model [56] that predicts the next token by the current context (its left context), the dynamic masked language model enables the model to predict the masked token by its left and right

Authorized licensed use limited to: Macau Univ of Science and Technology. Downloaded on July 14,2023 at 21:35:40 UTC from IEEE Xplore. Restrictions apply.

context, learning the comprehensive contextual information. The dynamic masked language model objective is formulated by maximizing the following log-likelihood:

$$\mathcal{L}_{\text{DMLM}}(\theta) = \sum_{i \in P} -\log p(w_i|\hat{S}), \tag{10}$$

where the probability p is modeled by `RepresentThemAll`, P is a set of positions of masked tokens, w_i is the masked word, and $\hat{S}$ is the rest words.

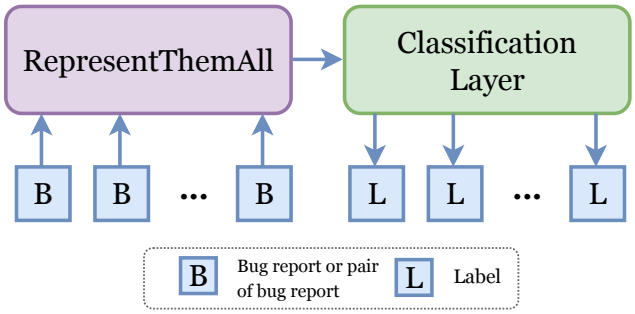
Pre-training Detail In this work, we denote the number of layers (i.e., Transformer encoder layer) as L. Following the previous work [52], the hyperparameter setting of `RepresentThemAll` is: $L = 12$, $d_e = 768, h = 12, d_{ff} = 3072, d_q, d_k, d_v = 64$. We utilize Adam [57] with a learning rate of 5e-5, $\beta_1 = 0.9$, $\beta_2 = 0.98$, L2 weight decay of 0.01, and a linear decay of the learning rate, to optimize `RepresentThemAll`. We use *dropout* probability of 0.1 on all layers to avoid overfitting. We train `RepresentThemAll` with a batch size of 16 bug report sequences whose max length is 512 for about 270,000 steps, which is equal to 20 epochs. We evaluate `RepresentThemAll` every 10K on the validation set and keep the best checkpoint for the subsequent pre-training. We use the weight of `RoBERTa` [52] to initialize `RepresentThemAll`, which is the same as `CodeBERT` [19] and `BioBERT` [58].

E. Siamese Network Based Contrastive Learning

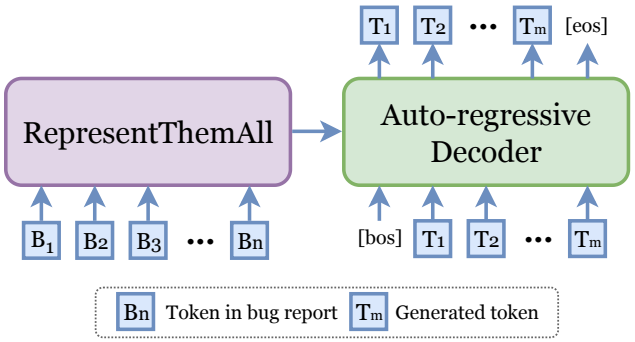
When finishing the above-mentioned pre-training, we further pre-train `RepresentThemAll` with our built contrastive learning objective, called "*find yourself*". The goal of "*find yourself*" is to let `RepresentThemAll` predict the input bug report itself from a set of negative instances. According to the introduction of contrastive learning (we described it in Section II-B), for each bug report, we need to build positive and negative instances for it. Since we use the mini-batch method to train the neural models, the bug reports in a batch are negative instances of each other. **The remaining problem is how to build the positive instance, which is also the core problem of contrastive learning.**

According to our observation, the vanilla Transformer encoder layer uses *dropout* twice: one is placed on the multi-head self-attention network layer and another one is placed on the feed-forward network layer. Considering that *dropout* works by randomly masking some neural units, we pass a bug report to the `RepresentThemAll` twice, by which we can obtain two different but semantically similar representations of the bug report and they are the positive instances of each other. As shown in Fig. 5, we achieve the above process by designing a siamese `RepresentThemAll` network with shared parameters. We pass the same batch of bug reports $\mathcal{S}$ to each `RepresentThemAll`, and get the corresponding representations:

$$R^d = \text{RTA}(\mathcal{S}; d), R^{d'} = \text{RTA}(\mathcal{S}; d') \tag{11}$$



(a) We add a classification layer to `RepresentThemAll` to perform the classification task.



(b) When `RepresentThemAll` is used for generation tasks, the whole architecture is based on the seq2seq framework.

Fig. 6. `RepresentThemAll` for classification and generation task.

where RTA is `RepresentThemAll`, and (d, d') are different *dropout* masks. Then, the training objective of "*find yourself*" is:

$$\ell = \sum_{r_i^d \in R^d} -\log \frac{e^{\text{sim}(r_i^d, r_i^{d'})}}{\sum_{r_j^{d'} \in R^{d'}} e^{\text{sim}(r_i^d, r_j^{d'})}}, j \neq i \tag{12}$$

where $r_i^{d'}$ is the positive instance of r_i^d and $r_j^{d'}$ is its negative instance. For a batch with size N, each bug report in it has $N-1$ negative instances. Therefore, "*find yourself*" lets the siamese `RepresentThemAll` network find the input bug report itself from $N-1$ negative instances.

Pre-training Detail All settings are the same as pre-training `RepresentThemAll` with the dynamic masked language model except we change the training epoch to 3 and batch size to 64. Additionally, we evaluate `RepresentThemAll` per 1K steps on the validation set and keep the best checkpoint for the subsequent fine-tuning.

F. Fine-tuning RepresentThemAll

We group all ASMT-ABR into two categories: classification and generation. Fig. 6 shows how to fine-tune `RepresentThemAll` for the classification and generation tasks. For the classification task, we add a classification layer to `RepresentThemAll`, which can be seen in Fig. 6(a). For each task, we simply pass the task-specific inputs into

Authorized licensed use limited to: Macau Univ of Science and Technology. Downloaded on July 14,2023 at 21:35:40 UTC from IEEE Xplore. Restrictions apply.

the whole model and fine-tune all the parameters. In the duplicate bug report detection task, for example, we pass the pair of bug reports into the `RepresentThemAll`, getting the representation of each bug report. Then, the classification layer predicts whether these two bug reports are duplicate. For the generation task, we put `RepresentThemAll` into the seq2seq framework, building an encoder-decoder network. For each task, we simply plug the task-specific inputs into the whole model and fine-tune all the parameters end-to-end. Taking the bug report summarization task as an example, we pass the bug description into the encoder, then the decoder outputs the bug report title.

IV. Experimental Setups

A. Research Questions

> RQ1: How effective is `RepresentThemAll` when compared to (1) the baseline approaches in the classification task, and (2) the baseline approaches in the generation task?

To resolve the produced issues (i.e., complexity, cost, and compatibility) of training multiple models for multiple downstream tasks in software maintenance, we propose a universal approach, `RepresentThemAll`, to handle multiple downstream tasks. In RQ1, we explore whether `RepresentThemAll` can replace multiple existing approaches to serve multiple downstream tasks, by retrieving its effectiveness and usability. We thus choose three classification tasks and one generation task, including duplicate bug report detection, bug priority prediction, bug severity prediction, and bug report summarization. For duplicate bug report detection, we pass the pair of bug reports to `RepresentThemAll`, then the model predicts whether these two bug reports are duplicate. For bug priority and severity predictions, we pass the bug report to `RepresentThemAll`, and then it outputs the bug report's priority or severity. As for bug report summarization, we input the bug description to `RepresentThemAll`, then gain the bug report title.

> RQ2: How effective is `RepresentThemAll` when compared to other pre-trained language models on four downstream tasks?

Considering that `RepresentThemAll` is pre-trained with the *dynamic masked language model* and "'*find yourself*' objectives, we think that it is necessary to compare it with other pre-trained language models, to further verify its effectiveness. In RQ2, therefore, we mainly compare `RepresentThemAll` with some famous pre-trained language models, retrieving its context-learning capabilities.

> RQ3: How does "*find yourself*" objective contribute to the performance of `RepresentThemAll`?

TABLE I
THE STATISTICS OF BUG REPORTS IN EACH PROJECT.

Project	Number of bug reports	Average Length
Mozilla	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 II
THE STATISTICS OF OPEN OFFICE DATASET.

Index	Pair of bug reports	Duplicate	Non_duplicate
Training set	122,297	89,027	33,270
Validation set	15,287	11,005	4,282
Test set	15,288	11,809	4,199

In this work, we design a new contrastive learning objective, "*find yourself*" and utilize it to further pre-train `RepresentThemAll`, for learning the semantic difference between bug reports. In order to explore the impact of "*find yourself*" objective, we conduct an ablation study on all downstream tasks. We first remove "*find yourself*" objective and pre-train `RepresentThemAll` from scratch, then fine-tune the pre-trained `RepresentThemAll` on all downstream tasks. Afterward, we also explore how *dropout* rate affects "*find yourself*" objective. Specifically, we use "*find yourself*" objective with different *dropout* rates, e.g., $\{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7\}$, to pre-train `RepresentThemAll`, then conduct statistics on fine-tuning results on all downstream tasks.

B. Dataset

We collect bug reports from four open-source repositories of Bugzilla, i.e., Mozilla, Eclipse, Netbeans, and GNU compiler collection (GCC). Specifically, we have collected **275,639 bug reports from Feb. 2000 to Sep. 2020**. According to our investigation, the Description and Summary of bug reports are used as the inputs in most ASMT-ABR [1], [2], [59], we concatenate the textual content of these two elements as a bug report sequence, in which Summary is the first sentence. Table I gives statistics of bug reports in each project. Then, we split 80% of all bug reports into the training set, 10% of bug reports into the validation set, and the remaining bug reports into the test set. We first use this dataset to pre-training `RepresentThemAll`, then use it to fine-tune pre-trained `RepresentThemAll` on bug priority prediction, bug severity prediction, and bug report summarization. As for the duplicate bug report detection, we conduct all experiments on the public Open Office dataset released by *Lazar et al.* [27], whose statistics are shown in Tabel II. In the Open Office dataset, *Lazar et al.* pair two bug reports and label them according to the fact whether they are semantically similar. In our experiments, we use the training set to perform pre-training and training, then use the validation and test sets

Authorized licensed use limited to: Macau Univ of Science and Technology. Downloaded on July 14,2023 at 21:35:40 UTC from IEEE Xplore. Restrictions apply.

to complete the evaluation. We do not conduct general preprocesses (e.g., rare words removal, stop word removal, special symbols removal, etc) for bug reports because BBPE can encode each token in bug reports, which reduces the time cost and improves the generalizability of RepresentThemAll.

C. Baselines and Evaluation Metrics

*a) **Bug report summarization**:* Bug report summarization, also known as bug report title generation, is a task that can automatically generate the bug report title from the bug description. Work in this area of research has generally focused on generating a short natural language title from a given description in the bug report. In this work, we select DeepSum [7], BugSum [8], PRHAN [60], and Transformer [53] as baselines, all of whom are NN-based, RNN-based or SAN-based approaches. We measure all approaches' performance by composite BLEU (c.B.) [61] and ROUGE-L (R.L), which are widely in various generations tasks such as code summarization [42], code translation [62], and machine translation [63], to measure the similarity between the sentence generated by models and the gold sentence.

*b) **Duplicate bug report detection**:* Duplicate bug report detection can help developers quickly distinguish duplicate bug reports in newly submitted bug reports, reducing their time-consuming. In this work, we choose Siamese [64], DWEN [5], and DC-CNN [6] as baselines, all of whom are built based on NN, RNN, or CNN. We utilize accuracy (A), recall (R), precision (P), and F1-score (F) to measure all approaches' performance, which is the same as the prior work [6].

*c) **Bug priority prediction**:* Bug priority prediction is to automatically predict the bug priority (i.e., P1-P5) for the given bug report. In this work, we select word2vec [14], cPur [32], and PPWGCN [2] as baselines. We utilize accuracy (A), recall (R), precision (P), F1-score (F), and weighted average (W.avg.) F1-score[3] to measure all approaches' performance.

*d) **Bug severity prediction**:* Similar to bug priority prediction, bug severity prediction can automatically predict the bug severity (i.e., Blocker (B.), Critical (C.), Major (Ma.), Minor (Mi.), and Trivial (T.)) for the given bug report. In this work, we select BSP-QASO [4], DNNSPBP [10], and PPWGCN [2] as baselines. The evaluation metrics for bug severity prediction are the same as bug priority prediction.

*e) **Baselines in RQ2**:* We choose two PLMs in the NLP community, i.e., BERT [18] and RoBERTa [52], and compare RepresentThemAll against them. We also choose CodeBERT [19] and seBERT [65] as another two baselines. The former is a PLM for code-natural language representation and achieves state-of-the-art results on code search [41] and code comment generation [42]. The latter is a domain-specific BERT pre-trained with software engineering data (i.e., Stack Overflow posts, GitHub issues, Jira issues, and GitHub commit messages). For all baselines, we only fine-tune them on the dataset for four downstream tasks.

[3]It may result in an F-score that is not between precision and recall. More details can be seen https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_recall_fscore_support.html.

TABLE III
RESULTS 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
RepresentThemAll	**39.19**	**20.57**	**35.97**	**10.13**

TABLE IV
RESULTS ON DUPLICATE BUG REPORT DETECTION.

Index	A (%)	R (%)	P (%)	F (%)
Siamese	83.99	85.86	86.38	86.12
DWEN	93.04	94.09	93.89	93.99
DC-CNN	94.29	96.70	93.65	95.15
RepresentThemAll	**97.88**	**98.54**	**98.54**	**98.54**

D. Experimental Environment

We conduct experiments on a server that owns 4x 20-core 2.2GHz Intel Xeon with 512GB memory and utilize 1 NVIDIA Tesla V100 with 32GB memory running on CUDA version 10.2. To implement our model, we use PyTorch [66] V.1.6.0, transformers [67] v.4.17.0, and datasets [68] v.1.17.0 with GPU support. For the baseline approaches whose source code is released, we directly re-run the source code on our dataset; For the baseline approaches whose source code is not provided, we re-implement them according to the corresponding literature and keep all parameter setting constant. We calculate all evaluation metrics by the datasets package and the scikit-learn package.

V. EVALUATION

A. Answer to RQ1: Retrieval Effectiveness

Table III and Table IV compare the performance of RepresentThemAll with the baseline approach in bug report summarization and duplicate bug report detection, respectively. Table V shows the results on bug priority prediction and bug severity prediction.

In bug report summarization, RepresentThemAll outperforms DeepSum, BugSum, PRHAN, and Transformer by large margins in terms of all ROUGE-L metrics (8.47 point to 21.59 point). In terms of composite BLEU, the improvements are 6.40, 4.32, 4.95, and 4.06 points, respectively. DeepSum is the worst performing model, and a potential reason is that the simple fully connected neural network layer cannot learn the contextual information from bug reports. Since the gated recurrent unit can learn partial contextual information, BugSum performs better than DeepSum. Nevertheless, the improvement is limited because the gated recurrent unit is weak at modeling long-range dependency. Transformer is the best baseline approach because self-attention networks can learn contextual information from the textual sequences without considering the distance of tokens in

Authorized licensed use limited to: Macau Univ of Science and Technology. Downloaded on July 14,2023 at 21:35:40 UTC from IEEE Xplore. Restrictions apply.

TABLE V
RESULTS ON BUG PRIORITY & SEVERITY PREDICTION. WE DENOTE "REPRESENTTHEMALL" AS "RTA".

Model	Metric	Bug Priority Prediction						Bug Severity Prediction						Metric	Model
		P1	P2	P3	P4	P5	W.avg.	W.avg.	B.	C.	Ma.	Mi.	T.		
RTA	P%	64.87	51.91	82.12	31.78	76.65	**72.32**	**65.11**	72.34	71.59	60.90	59.57	62.21	P%	RTA
	R%	65.81	45.91	87.11	9.71	54.12	**73.49**	**64.72**	68.68	63.56	69.41	60.12	53.30	R%	
	F%	65.34	48.73	84.54	14.88	63.44	**72.69**	**64.73**	70.46	67.34	64.88	59.84	57.42	F%	
	A%	**73.49**						**64.72**						A%	
PPWGCN	P%	57.60	37.43	82.37	4.30	47.90	67.34	55.54	56.30	64.74	54.75	50.79	43.75	P%	PPWGCN
	R%	58.70	41.01	56.91	46.00	57.93	54.24	55.24	64.70	57.09	56.57	44.73	51.15	R%	
	F%	58.15	39.14	67.31	7.87	52.44	59.13	55.22	60.21	60.68	55.64	47.57	47.16	F%	
	A%	54.24						55.24						A%	
word2vec	P%	56.28	26.59	64.46	3.57	44.72	54.60	54.83	62.48	64.88	51.26	45.38	48.45	P%	DNNSPBP
	R%	18.37	10.35	92.94	0.29	46.51	60.75	54.18	51.00	59.34	60.87	49.25	33.56	R%	
	F%	27.70	14.90	76.13	0.53	45.60	53.59	54.12	56.16	61.98	55.65	47.24	39.65	F%	
	A%	60.75						54.18						A%	
cPur	P%	58.75	61.89	64.22	0.00	80.26	62.03	54.21	59.92	63.50	48.22	49.37	51.93	P%	BSP-QASO
	R%	25.28	7.10	96.87	0.00	51.59	63.92	53.47	54.25	58.03	68.54	38.39	19.01	R%	
	F%	35.35	12.73	77.23	0.00	62.81	55.72	52.42	56.94	60.64	56.61	43.19	27.83	F%	
	A%	63.92						53.47						A%	

the sequence. Although PRHAN also adopts self-attention networks, it performs worse than Transformer and BugSum. We think the potential reason is that PRHAN is designed for pull request summarization [69] and is not suitable for modeling bug report sequences. Compared with the baseline approaches, RepresentThemAll is constructed by stacking Transformer encoder layers and pre-trained on the large-scale bug report corpus, which enables it to fully learn the context-related information and knowledge in bug reports, generating effective representations for the subseuqent fine-tuing. Therefore, RepresentThemAll achieves state-of-the-art results on bug report summarization.

In duplicate bug report detection, RepresentThemAll outperforms Siamese, DWEN, and DC-CNN by 12.42, 4.55, and 3.39 points in terms of F1-score, respectively. In terms of accuracy, the improvements are 13.89, 4.84, and 3.59 points. From Table IV we can observe that each approach achieves relatively high results on the Open Office dataset. Moreover, the performance improvement is positively correlated with the model's semantics learning ability. Compared with the single-layered neural network used in Siamese, word embedding and deep neural networks can learn more semantic information, which brings performance improvement for DWEN. Similarly, DC-CNN achieves higher results due to the better semantics learning ability of multi-layer dual CNN. As for RepresentThemAll, it is designed by stacking 12 Transformer encoder layers, which can fully learn the global contextual information for each token. Additionally, we conduct an effective pre-training for RepresentThemAll before fine-tuning it, making the model fully understand bug report related knowledge, thus bringing state-of-the-art results.

In bug priority prediction, we focus on weighted average Fa-score and accuracy to measure all models. In terms of weighted average F1-score, RepresentThemAll outperforms PPWGCN, word2vec, and cPur by 22.93%, 35.64%, and 30.46%. In terms of accuracy, the improvements are 35.49%, 20.97%, and 14.97%. Diving into the model's performance on every priority label, we can find that each model has a relatively low F1-score on priority label P4. Particularly, word2vec and cPur can hardly make any effective prediction for bug reports with P4 priority. According to our investigation, we find the potential reason is that P4 is a rare priority label, which causes the label imbalance problem. Since PPWGCN introduces the weighted loss function to alleviate the label imbalance problem, it achieves a 7.87% F1-score on the prediction of P4 priority. As for RepresentThemAll, we further pre-train it with "*find yourself*" objective when finishing pre-training with the *dynamic masked language model*, to fully learn the semantic differences between bug reports. As a result, fine-tuned RepresentThemAll achieves the highest F1-score on the prediction of P4 priority and outperforms PPWGCN by 89.07%.

In bug severity prediction, we focus on weighted average F1-score and accuracy to measure all models. In terms of weighted average F1-score, RepresentThemAll outperforms PPWGCN, DNNSPBP, and BSP-QASO by large margins in terms of these two metrics (from 9.48 to 12.31 points). The reason is that, during the pre-training phase, RepresentThemAll has fully learned the semantic difference between bug reports, which helps it distinguish bug reports with different severity in the fine-tuning phase. In addition to the best overall performance, RepresentThemAll also achieves state-of-the-art results in the prediction of each severity label.

Authorized licensed use limited to: Macau Univ of Science and Technology. Downloaded on July 14,2023 at 21:35:40 UTC from IEEE Xplore. Restrictions apply.

The above experimental results show that RepresentThemAll can effectively be applied to multiple downstream tasks. When considering the complexity, RepresentThemAll successfully replaces multiple models and works for multiple downstream tasks, thus RepresentThemAll has a lower complexity when constructing an automated software maintenance system. As for cost, training RepresentThemAll requires less cost than training multiple models since we need not perform repeated training for different tasks, and fine-tuning has a low training cost. Moreover, as we just use one model for different downstream tasks, the compatibility problem generated by using multiple models is not in RepresentThemAll.

> Answer to RQ1: As a universal approach, RepresentThemAll can replace existing approaches to handle different downstream tasks in software maintenance.

B. Answer to RQ2: Effectiveness Comparison of Different PLMs

Table VI presents the results of the baseline PLMs on four downstream tasks. Due to the limited space, we focus on two main evaluation metrics to measure each PLM and put the completed results into our GitHub repository.

From Tabel VI, we find that RepresentThemAll can only deliver slight performance improvement (less than 1 point in terms of F1-score and accuracy) for duplicate bug report detection compared with the baseline PLMs. The reason is that the accuracy and F1-score of all baseline PLMs exceed 96%, so it is difficult to obtain a significant performance improvement. In bug priority predictions, RepresentThemAll outperforms BERT, RoBERTa, CodeBERT, and seBERT by 2.73, 1.98, 2.37, and 2.11 points in terms of weighted average F1-score. In terms of accuracy, the improvements become 3.03, 2.41, 2.74, and 0.71 points. In bug severity prediction, RepresentThemAll outperforms BERT, RoBERTa, CodeBERT, and seBERT by 4.09, 3.87, 3.68, and 4.03 points in terms of weighted average F1-score. In terms of accuracy, the improvements become 4.10, 3.86, 3.82, and 4.11 points. RepresentThemAll brings similar performance to these two tasks since they are both multi-classification tasks with five categories. Besides, the considerable performance gain also supports the effectiveness of two pre-training objectives, especially "*find yourself*" that lets RepresentThemAll learn the semantic differences between bug reports, which helps the model to identify bug reports with different labels. In bug report summarization, RepresentThemAll brings a noteworthy improvement. Specifically, RepresentThemAll outperforms BERT, RoBERTa, CodeBERT, and seBERT by large margins (3.81 to 12.66 and 1.89 to 9.4) in terms of ROUGE-2 and composite BLEU, respectively. This is because the generation task is more difficult than the classification task and has a higher requirement for the contextual information of bug reports. However, PLMs in other fields do not learn the contextual knowledge of bug reports in the pre-training phase, which cannot be compensated by fine-tuning alone. We fully pre-train RepresentThemAll with two carefully designed pre-training objectives, which help it effectively obtain contextual knowledge of bug reports in the pre-training phase, providing effective enough contextual information to the subsequent fine-tuning.

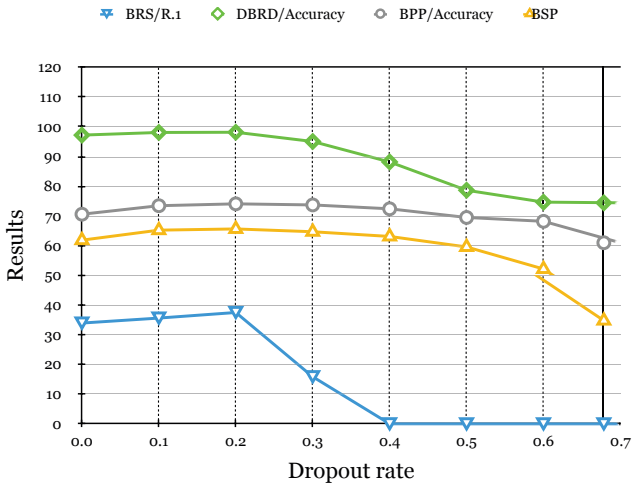


Fig. 7. Results on downstream tasks with different *dropout* rate settings. BPP, BSP, BRS, and DBRD denote bug priority prediction, bug severity prediction, bug report summary, and duplicate bug report detection.

> Answer to RQ2: By comparison with the baseline PLMs, RepresentThemAll can learn the domain knowledge related to bug reports in the pre-training phase, which enables it to gain even better performance on four downstream tasks.

C. Answer to RQ3: Ablation Study

Table VII shows the results of RepresentThemAll on four downstream tasks when we pre-train it from scratch without "*find yourself*" objective and Fig. 7 presents the results of RepresentThemAll on four downstream tasks when setting different *dropout* rate for "*find yourself*" objective.

In Table VII, due to limited space, we focus on ROUGE-2 and composite BLEU to measure models in bug report summarization, F1-score and accuracy to measure models in duplicate bug report detection, and weighted average F1-score and accuracy to measure models in bug priority and severity predictions. Specifically, we can observe that except for duplicate bug report detection, "*find yourself*" objective can bring at least one point improvement to all downstream tasks, which supports its effectiveness.

In Fig. 7, we focus on ROUGE-1 to measure models in bug report summarization and focus on accuracy to measure models in duplicate bug report detection, bug priority prediction, and bug severity prediction. From the figure, we can find that

Authorized licensed use limited to: Macau Univ of Science and Technology. Downloaded on July 14,2023 at 21:35:40 UTC from IEEE Xplore. Restrictions apply.

TABLE VI
RESULTS OF PLMS ON FOUR DOWNSTREAM TASKS.

-	Bug report summary		Duplicate bug report detection		Bug priority prediction		Bug severity prediction	
	ROUGE-2	c.B.	F%	Accuracy%	W.avg. F%	Accuracy%	W.avg. F%	Accuracy%
RTA	**20.57**	**10.13**	**98.54**	**97.88**	**72.69**	**73.49**	**64.73**	**64.72**
BERT	7.91	0.73	98.26	97.47	69.96	70.46	60.64	60.62
RoBERTa	16.76	8.24	98.22	97.43	70.71	71.08	60.86	60.86
CodeBERT	16.30	7.74	97.91	96.98	70.32	70.75	61.05	60.90
seBERT	16.05	4.19	98.10	97.25	70.58	72.78	60.70	60.61

TABLE VII
RESULTS OF ABLATION STUDY ON DOWNSTREAM TASKS.

-	Bug report summary		Duplicate bug report detection		Bug priority prediction		Bug severity prediction	
	ROUGE-2	c.B.	F%	Accuracy%	W.avg. F%	Accuracy%	W.avg. F%	Accuracy%
RTA	**20.57**	**10.13**	**98.54**	**97.88**	**72.69**	**73.49**	**64.73**	**64.72**
w/o "*find yourself*"	18.36	9.10	98.43	97.61	71.52	72.36	62.51	62.55

as *dropout* rate increases, "*find yourself*" can bring continuous performance improvement for RepresentThemAll on four downstream tasks. When *dropout* rate exceeds 0.2, the performance of RepresentThemAll starts to degrade. As *dropout* rate reaches 0.7, RepresentThemAll's performance on all downstream tasks is degraded by more than 10 points. Setting *dropout* rate to a large value means that the model needs to drop out much contextual information of bug reports. Consequently, the two representations of one bug report generated from the siamese RepresentThemAll network may not be semantically similar again. To better explain the above problem, we take a simple example. There is a sentence "Fading is true while flowering is past.". We mask half of the words in it twice and the words masked twice cannot be the same:

- [MASK] is [MASK] while [MASK] is past.
- Fading [MASK] true while [MASK] is [MASK].

Although these two masked sentences are identical, it is hard for us to regard them as semantically similar sentences. Therefore, a high *dropout* rate means that when pre-training RepresentThemAll with "*find yourself*" objective, it is difficult to find the input bug report itself from a set of negative bug reports, which severely hurts the model's representation learning ability. Additionally, the generation task is more sensitive to the dropout rate since our model performs no effect when the dropout rate reaches 0.4. This is because the generation task has a higher requirement of context learning ability for the model, while a high dropout rate may significantly impair the context modeling ability of the model due to introducing a large noise.

VI. THREATS TO VALIDITY

The conclusion of this paper suffers from several threats to validity. A key threat to the internal validity is the hyperparameter setting we used to pre-train and fine-tune RepresentThemAll. A mitigating factor is that most parameters in the pre-training and fine-tuning phases were reported in other prior reputable literature as recommended or optimal [18], [52], and we also perform a study on other important parameters.

A threat to the external validity is that we only use bug reports from the Bugzilla platform to conduct our experiments. Actually, there are some other bug tracking systems, like Trac[4], thus we cannot make sure that RepresentThemAll still performs well if we use bug reports from these platforms. A mitigating factor is that bug reports in these platforms also contain the Summary and Description elements, and we only use the textual information from these two elements to pre-train and fine-tune RepresentThemAll.

VII. CONCLUSION

In this paper, we propose RepresentThemAll, which can learn the universal representation of bug reports, serving multiple software maintenance tasks associated with bug reports. To learn effective representation for tine-tuning, we construct RepresentThemAll by stacking multiple Transformer encoder layers and pre-train it with two carefully designed objectives: *dynamic masked language model* and "*find yourself*". We evaluate RepresentThemAll by fine-tuning it on four software maintenance tasks associated with bug reports. The results show that RepresentThemAll outperforms all baseline approaches on these four downstream tasks. We also demonstrate that RepresentThemAll is more effective than PLMs in other fields for software maintenance tasks associated with bug reports. In the future, we plan to develop a RepresentThemAll based software maintenance system that simultaneously serves multiple tasks.

ACKNOWLEDGMENT

We sincerely appreciate Dr. Zhou Xu from Chongqing University who provide a lot of valuable suggestions on paper writing. We also sincerely appreciate Dr. He Ye from KTH for her valuable and constructive comments, which significantly improve the quality of our paper. This work was supported by the Macao Science and Technology Development Fund under Grant 0047/2020/A1 and 0014/2022/A.

[4]https://trac.edgewall.org/

Authorized licensed use limited to: Macau Univ of Science and Technology. Downloaded on July 14,2023 at 21:35:40 UTC from IEEE Xplore. Restrictions apply.

REFERENCES

[1] Y. Tian, D. Lo, X. Xia, and C. Sun, "Automated prediction of bug report priority using multi-factor analysis," *Empirical Software Engineering*, vol. 20, no. 5, pp. 1354–1383, 2015.

[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 Transactions on Reliability*, vol. 70, no. 2, pp. 563–574, 2021.

[3] T. Zhang, J. Chen, G. Yang, B. Lee, and X. Luo, "Towards more accurate severity prediction and fixer recommendation of software bugs," *Journal of Systems and Software*, vol. 117, pp. 166–184, 2016.

[4] Y. Tan, S. Xu, Z. Wang, T. Zhang, Z. Xu, and X. Luo, "Bug severity prediction using question-and-answer pairs from stack overflow," *Journal of Systems and Software*, vol. 165, p. 110567, 2020.

[5] A. Budhiraja, K. Dutta, R. Reddy, and M. Shrivastava, "Dwen: deep word embedding network for duplicate bug report detection in software repositories," in *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*, 2018, pp. 193–194.

[6] J. He, L. Xu, M. Yan, X. Xia, and Y. Lei, "Duplicate bug report detection using dual-channel convolutional neural networks," in *Proceedings of the 28th International Conference on Program Comprehension*, 2020, pp. 117–127.

[7] X. Li, H. Jiang, D. Liu, Z. Ren, and G. Li, "Unsupervised deep bug report summarization," in *Proceedings of the 26th International Conference on Program Comprehension*, 2018, pp. 144–155.

[8] H. Liu, Y. Yu, S. Li, Y. Guo, D. Wang, and X. Mao, "Bugsum: Deep context understanding for bug report summarization," in *Proceedings of the 28th International Conference on Program Comprehension*, 2020, pp. 94–105.

[9] M. Kumari and V. Singh, "An improved classifier based on entropy and deep learning for bug priority prediction," in *Proceedings of the 18th International Conference on Intelligent Systems Design and Applications*, 2018, pp. 571–580.

[10] W. Y. Ramay, Q. Umer, X. C. Yin, C. Zhu, and I. Illahi, "Deep neural network-based severity prediction of bug reports," *IEEE Access*, vol. 7, pp. 46 846–46 857, 2019.

[11] Y. Xiao, J. Keung, K. E. Bennin, and Q. Mi, "Improving bug localization with word embedding and enhanced convolutional neural networks," *Information and Software Technology*, vol. 105, pp. 17–29, 2019.

[12] H. Isotani, H. Washizaki, Y. Fukazawa, T. Nomoto, S. Ouji, and S. Saito, "Duplicate bug report detection by using sentence embedding and fine-tuning," in *Proceedings of the 37th International Conference on Software Maintenance and Evolution*, 2021, pp. 535–544.

[13] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 1, pp. 4–24, 2020.

[14] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," 2013.

[15] Y. LeCun, Y. Bengio *et al.*, "Convolutional networks for images, speech, and time series," *The Handbook of Brain Theory and Neural Networks*, vol. 3361, no. 10, p. 1995, 1995.

[16] T. Zhang, H. Jiang, X. Luo, and A. T. Chan, "A literature review of research in bug resolution: Tasks, challenges and future directions," *The Computer Journal*, vol. 59, no. 5, pp. 741–773, 2016.

[17] W. Zou, D. Lo, Z. Chen, X. Xia, Y. Feng, and B. Xu, "How practitioners perceive automated bug report management techniques," *IEEE Transactions on Software Engineering*, vol. 46, no. 8, pp. 836–862, 2018.

[18] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, 2019, pp. 4171–4186.

[19] 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.

[20] C. Niu, C. Li, V. Ng, J. Ge, L. Huang, and B. Luo, "Spt-code: sequence-to-sequence pre-training for learning source code representations," in *Proceedings of the 44th International Conference on Software Engineering*, 2022, pp. 01–13.

[21] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *Proceedings of the 37th International Conference on Machine Learning*, 2020, pp. 1597–1607.

[22] T. Gao, X. Yao, and D. Chen, "SimCSE: Simple contrastive learning of sentence embeddings," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 2021, pp. 6894—-6910.

[23] M. Wei, N. S. Harzevili, Y. Huang, J. Wang, and S. Wang, "Clear: Contrastive learning for api recommendation," in *Proceedings of the 44th International Conference on Software Engineering*, 2022, pp. 376–387.

[24] Q. Chen, J. Lacomis, E. J. Schwartz, G. Neubig, B. Vasilescu, and C. L. Goues, "Varclr: Variable semantic representation pre-training via contrastive learning," in *Proceedings of the 44th International Conference on Software Engineering*, 2022, pp. 2327–2339.

[25] J. Huang, D. Tang, L. Shou, M. Gong, K. Xu, D. Jiang, M. Zhou, and N. Duan, "CoSQA: 20,000+ web queries for code search and question answering," in *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing*, 2021, pp. 5690–5700.

[26] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proceedings of the 28th Conference on Neural Information Processing Systems (NIPS)*, 2014, pp. 3104–3112.

[27] A. Lazar, S. Ritchey, and B. Sharif, "Generating duplicate bug datasets," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, 2014, pp. 392–395.

[28] Y. Tian, D. Lo, and C. Sun, "Information retrieval based nearest neighbor classification for fine-grained bug severity prediction," in *Proceedings of the 19th Working Conference on Reverse Engineering*, 2012, pp. 215–224.

[29] A. T. Nguyen, T. T. Nguyen, J. Al-Kofahi, H. V. Nguyen, and T. N. Nguyen, "A topic-based approach for narrowing the search space of buggy files from a bug report," in *Proceedings of the 26th International Conference on Automated Software Engineering*, 2011, pp. 263–272.

[30] J. Zhou, H. Zhang, and D. Lo, "Where should the bugs be fixed? more accurate information retrieval-based bug localization based on bug reports," in *Proceedings of the 34th International Conference on Software Engineering*, 2012, pp. 14–24.

[31] R. K. Saha, M. Lease, S. Khurshid, and D. E. Perry, "Improving bug localization using structured information retrieval," in *Proceedings of the 28th International Conference on Automated Software Engineering*, 2013, pp. 345–355.

[32] Q. Umer, H. Liu, and I. Illahi, "Cnn-based automatic prioritization of bug reports," *IEEE Transactions on Reliability*, vol. 69, no. 4, pp. 1341–1354, 2019.

[33] R. Hadsell, S. Chopra, and Y. LeCun, "Dimensionality reduction by learning an invariant mapping," in *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2006, pp. 1735–1742.

[34] O. Chaparro, J. Lu, F. Zampetti, L. Moreno, M. Di Penta, A. Marcus, G. Bavota, and V. Ng, "Detecting missing information in bug descriptions," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, 2017, pp. 396–407.

[35] T. Zhang, J. Chen, H. Jiang, X. Luo, and X. Xia, "Bug report enrichment with application of automated fixer recommendation," in *Proceedings of the 25th International Conference on Program Comprehension*, 2017, pp. 230–240.

[36] R. K. Saha, Y. Lyu, H. Yoshida, and M. R. Prasad, "Elixir: Effective object-oriented program repair," in *2017 32nd IEEE/ACM International Conference on Automated Software Engineering*, 2017, pp. 648–659.

[37] A. Koyuncu, K. Liu, T. F. Bissyandé, D. Kim, M. Monperrus, J. Klein, and Y. Le Traon, "ifixr: Bug report driven program repair," in *Proceedings of the 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 314–325.

[38] U. Alon, R. Sadaka, O. Levy, and E. Yahav, "Structural language models of code," in *Proceedings of the 37th International Conference on Machine Learning*, 2020, pp. 245–256.

[39] H. Husain, H.-H. Wu, T. Gazit, M. Allamanis, and M. Brockschmidt, "Codesearchnet challenge: Evaluating the state of semantic code search," 2020.

[40] X. Gu, H. Zhang, and S. Kim, "Deep code search," in *Proceedings of the IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, 2018, pp. 933–944.

Authorized licensed use limited to: Macau Univ of Science and Technology. Downloaded on July 14,2023 at 21:35:40 UTC from IEEE Xplore. Restrictions apply.

[41] S. Fang, Y.-S. Tan, T. Zhang, and Y. Liu, "Self-attention networks for code search," *Information and Software Technology*, vol. 134, p. 106542, 2021.
[42] X. Hu, G. Li, X. Xia, D. Lo, and Z. Jin, "Deep code comment generation," in *Proceedings of the IEEE/ACM 26th International Conference on Program Comprehension (ICPC)*, 2018, pp. 200–20 010.
[43] A. LeClair, S. Jiang, and C. McMillan, "A neural model for generating natural language summaries of program subroutines," in *Proceedings of the 41st International Conference on Software Engineering (ICSE)*, 2019, pp. 795–806.
[44] N. Jiang, T. Lutellier, and L. Tan, "Cure: Code-aware neural machine translation for automatic program repair," in *Proceedings of the 43rd International Conference on Software Engineering*, 2021, pp. 1161–1173.
[45] P. Ardimento and C. Mele, "Using bert to predict bug-fixing time," in *Proceedings of the 14th Conference on Evolving and Adaptive Intelligent Systems*, 2020, pp. 1–7.
[46] M. Li and B.-B. Yin, "Arb-bert: An automatic aging-related bug report classification method based on bert," in *Proceedings of the 8th International Conference on Dependable Systems and Their Applications*, 2021, pp. 474–483.
[47] L. Bo and J. Lu, "Bug question answering with pretrained encoders," in *Proceedings of the 28th IEEE International Conference on Software Analysis, Evolution and Reengineering*, 2021, pp. 654–660.
[48] P. R. Henao, J. Fischbach, D. Spies, J. Frattini, and A. Vogelsang, "Transfer learning for mining feature requests and bug reports from tweets and app store reviews," in *Proceedings of the 29th International Requirements Engineering Conference Workshops*, 2021, pp. 80–86.
[49] C. Wang, K. Cho, and J. Gu, "Neural machine translation with byte-level subwords," in *Proceedings of the 34th AAAI Conference on Artificial Intelligence*, pp. 9154–9160.
[50] R. Sennrich, B. Haddow, and A. Birch, "Neural machine translation of rare words with subword units," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2016, pp. 1715–1725.
[51] J. Xu, H. Zhou, C. Gan, Z. Zheng, and L. Li, "Vocabulary learning via optimal transport for neural machine translation," in *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, 2021, pp. 7361–7373.
[52] 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.
[53] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proceedings of the 31st Conference on Neural Information Processing Systems (NIPS)*, 2017, pp. 5998–6008.
[54] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.
[55] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.
[56] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," *Technical Report, OpenAI*, 2018.
[57] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.
[58] J. Lee, W. Yoon, S. Kim, D. Kim, S. Kim, C. H. So, and J. Kang, "Biobert: a pre-trained biomedical language representation model for biomedical text mining," *Bioinformatics*, vol. 36, no. 4, pp. 1234–1240, 2020.
[59] A. Ciborowska and K. Damevski, "Fast changeset-based bug localization with bert," *arXiv preprint arXiv:2112.14169*, 2021.
[60] 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," *Journal of Systems and Software*, vol. 185, p. 111160, 2022.
[61] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2002, pp. 311–318.
[62] X. Chen, C. Liu, and D. Song, "Tree-to-tree neural networks for program translation," *Advances in Neural Information Processing Systems*, vol. 31, 2018.
[63] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," 2016.
[64] J. Deshmukh, K. Annervaz, S. Podder, S. Sengupta, and N. Dubash, "Towards accurate duplicate bug retrieval using deep learning techniques," in *Proceedings of the 33rd International Conference on Software Maintenance and Evolution*, 2017, pp. 115–124.
[65] J. Von der Mosel, A. Trautsch, and S. Herbold, "On the validity of pre-trained transformers for natural language processing in the software engineering domain," *IEEE Transactions on Software Engineering*, 2022.
[66] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in Neural Information Processing Systems*, vol. 32, 2019.
[67] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz *et al.*, "Transformers: State-of-the-art natural language processing," in *Proceedings of the 2020 conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 2020, pp. 38–45.
[68] Q. Lhoest, A. V. del Moral, Y. Jernite, A. Thakur, P. von Platen, S. Patil, J. Chaumond, M. Drame, J. Plu, L. Tunstall *et al.*, "Datasets: A community library for natural language processing," *arXiv preprint arXiv:2109.02846*, 2021.
[69] Z. Liu, X. Xia, C. Treude, D. Lo, and S. Li, "Automatic generation of pull request descriptions," in *Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2019, pp. 176–188.

Authorized licensed use limited to: Macau Univ of Science and Technology. Downloaded on July 14,2023 at 21:35:40 UTC from IEEE Xplore. Restrictions apply.