Duplicate Bug Report Detection by Using Sentence Embedding and Fine-tuning

Haruna Isotani
Dept. Computer Science and Engineering
Waseda University
Tokyo, Japan
hototogisu@fuji.waseda.jp

Hironori Washizaki
Dept. Computer Science and Engineering
Waseda University
Tokyo, Japan
washizaki@waseda.jp

Yoshiaki Fukazawa
Dept. Computer Science and Engineering
Waseda University
Tokyo, Japan
fukazawa@waseda.jp

Tsutomu Nomoto
Software Innovation Center
NTT CORPORATION
Tokyo, Japan
tsutomu.nomoto.cb@hco.ntt.co.jp

Saori Ouji
Software Innovation Center
NTT CORPORATION
Tokyo, Japan
saori.ouji.eu@hco.ntt.co.jp

Shinobu Saito
Software Innovation Center
NTT CORPORATION
Tokyo, Japan
shinobu.saitou.cm@hco.ntt.co.jp

***Abstract*—Industrial software maintenance devotes much time and effort to find duplicate bug reports. In this paper, we propose an automated duplicate bug report detection system to improve software maintenance efficiency. Our system detects duplicate reports by vectorizing the contents of each report item by deep-learning-based sentence embedding and calculating the similarity of the whole report from those of the item vectors. The Sentence-BERT fine-tuned with report texts is used for sentence embedding. Finally, we verify that the combination of processing separately by item and Sentence-BERT fine-tuned with reports effectively detects duplicate bug reports in industrial experiments that compare the performance of existing methods.**

Keywords*—*Bug reports, duplicate detection, BERT, sentence embedding, natural language processing, information retrieval

I. Introduction

Software maintenance includes fixing user-reported bugs, which are present in bug reports. The first step is to screen whether a newly reported bug is unique or has been previously reported. A bug report containing an already reported bug is called a duplicate bug report. Some organizations devote many hours to manual triage of duplicate bug reports. In this paper, we propose an automated duplicate bug report detection system, which addresses this issue for a specific maintenance method. We target the maintenance of certain software products developed and maintained by the NTT Corporation.

The proposed system uses sentence embedding (distributed representations of semantics) to generate vectors from the description content for each item in a bug report. The system calculates the semantic similarities of reports from vector representation similarities. A high degree of similarity is used to identify duplicate reports. For sentence embedding, we use Sentence-BERT (SBERT) [1], which is initially trained on general texts and then fine-tuned with texts in bug reports. We build a specialized SBERT for each item in a report by changing part of the texts in bug reports used for fine-tuning.

As baselines, we also built two systems to generate vectors from bug reports of all items using basic natural language processing techniques. Then we compare the proposed system to these baselines systems in industrial experiments. We confirmed that the proposed system outperforms the baseline systems. The combination of processing separately by item and fine-tuning SBERT with bug reports effectively detects the duplicate bug reports targeted by this research.

This paper addresses the following research questions:

RQ1. For the target type of bug reports, can duplicate reports be identified? Bug reports are comprised of various elements, making direct comparisons difficult. RQ1 investigates which elements affect the performance of detecting duplicate bug reports. In addition, the technique with the optimal performance may differ by the type of bug report. To address RQ1, we identified issues in achieving duplicate bug report detection and selected a technique that solves each issue.

RQ2. Does the proposed system outperform the baseline systems? The core strategy of our proposed system is a combination of separately processing by item and fine-tuning SBERT with bug reports. By elucidating the contribution of our system to identify duplicate reports, this research may be applicable detect duplicate bug reports in practical situations. To address RQ2, we built baseline systems using basic natural language processing techniques to generate vectors from all items in a report. Then we compared the proposed system and the baseline systems using a performance evaluation metric.

Our paper makes the following contributions. Firstly, duplicate bug report detection system suitable for a specific method of software maintenance is reported. Secondly, the performance of the proposed system is compared to baseline systems.

The rest of this paper is organized as follows. Section II describes the background regarding the targeted maintenance

2576-3148/21/$31.00 ©2021 IEEE
DOI 10.1109/ICSME52107.2021.00054

method and SBERT. Section III details the proposed system. Section IV evaluates the proposed system. Section V introduces related work. Finally, Section VI discusses the conclusions and future work.

II. Background

A. Maintenance Process

Figure 1 shows the general flow of maintenance that our research targets. Figure 2 depicts a detailed flow of maintenance for a project from the time a problem report is received until the time a response of the findings is sent to the user.

Three types of bug reports are used in the maintenance. The first is a problem report from a user to the reception center. The second is an inquiry report in which the reception center requests that the maintenance team investigates the problem in detail. The third is a failure report in which the maintenance team records the investigation results and a solution with the development vendor. All three types are written in Japanese.

When a user finds a problem, her or she reports it to the reception center. The reception center initially investigates it by focusing on manuals, screen displays, and logs. If it confirms that the problem is not caused by a bug or has been solved previously, it offers a solution to the user. If not, an inquiry report, which denotes the problem and the opinion of the reception center, is generated and given to the maintenance team. The maintenance team investigates the problem by reviewing the design document and the code. If the problem is not caused by a bug or has been solved previously, the maintenance team writes the findings in the answer column and returns the inquiry report to the reception center. If the problem is caused by a new bug, it issues a failure report, which includes details of the problem. Additionally, the maintenance team documents that a new bug caused the problem, a failure report was issued to fix it, and how to avoid the problem if documentation is listed in the answer column of the inquiry report. Then the failure report is returned to the reception center. The reception center refers to the answer column of the inquiry report and responds to the user. The maintenance team and the development vendor investigate the cause of the problem, decide on a solution and fill out the failure report. After that, the development vendor fixes the bug accordingly.

One problem with this flow is that maintenance team must investigate and judge whether a new bug caused the problem in the inquiry report. This determination is time consuming. Additionally, it is burdensome and requires highly skilled maintenance engineers because each previous inquiry and failure report is analyzed individually. Hence, investigation knowledge is unevenly distributed. To solve these problems, we built a system that automatically detects if the new inquiry report is a duplicate report among previous inquiry and failure reports.

The following items are described in inquiry reports:

1) *Title:* A summary of the reported problem, which is completed by the reception center.

2) *Content:* Details of the problem, including the phenomena, environment, settings, and operations at the time

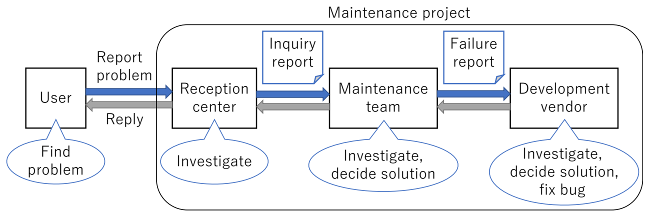

Fig. 1. Overview of the maintenance process.

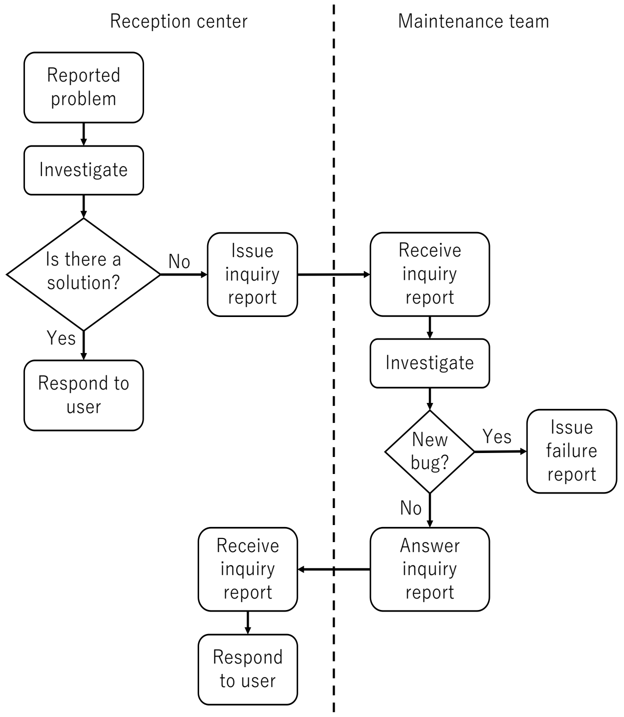

Fig. 2. Maintenance process of the reception center and the maintenance team.

of the incident. This information is provided by the reception center.

3) *Commercial impact:* Impact of the problem on the service and users, which is either noted by the reception center or left blank.

4) *Answer:* Results of the maintenance team's investigation of the problem. If it is caused by a new bug, then the fact that a failure report was issued and how to avoid the issue in the current situation are denoted. If the problem is not caused by a bug, an explanation of the problem is provided. If the problem has been solved previously, the ID of the previous inquiry or failure report is included. This information is documented by the maintenance team.

The following items are described in failure reports:

1) *Title:* A summary of the problem of the failure report, which differs from the title of the inquiry report and is completed by the maintenance team.

2) *Test environment:* The environment in which the problem was observed, which is provided by the maintenance team.

3) Details: Details of the problem, including the phenomena, environment, settings, and operations at the time of the incident. Investigation results are noted by the reception center, while the maintenance team documents their initial opinion.

4) Test environment: The environment in which the problem was observed, which is provided by the maintenance team.

5) Details: Details of the problem, including the phenomena, environment, settings, and operations at the time of the incident. Investigation results are noted by the reception center, while the maintenance team documents their initial opinion.

6) Failure cause: This is entered by the maintenance team and the development vendor, and is further divided as follows:

a) Cause: Cause of the problem.

b) Conditions: Conditions in which the problem occurs.

c) Effect: Abnormal behavior of the product when the problem occurs and particularly troubling points for users.

d) Workaround: How to work around the problem until the bug is fixed.

e) Affected version: The versions of the product that cause the problem.

7) Measures: This is completed by the maintenance team and the development vendor, and is further divided as follows:

a) Measures policy: How to fix the bug.

b) Source: A simple description of where and how to modify the source code. This may only refer to a different file.

c) Document: A simple description of where and how to modify the document. This typically refers to a different file.

8) Feedback: This is completed by the maintenance team and the development vendor, and is further divided as follows:

a) Injection cause: The injection cause of the bug, including background and history. If the development vendor has changed since the time of development, this may be omitted.

b) Reason for delay in removal: The reason why the bug could not be found before the release. If the development vendor has changed since the time of development, this may be omitted.

c) Inspection: The results of inspection for similar bugs in parts of the product with a comparable structure.

d) Measures to prevent recurrence: Measures to prevent similar bugs from being injected in the future. If the development vendor has changed since the time of development, this may be omitted.

From the contents of these items, duplicate reports are detected.

B. Sentence-BERT

Sentence-BERT (SBERT) [1] is a sentence embedding, which refers to a technique and a model for converting sentences into vectors that capture their content features. It was state-of-the-art at the time of the paper's publication (2019) and is still considered as such in the latest research [2][3][4]. Its output is a vector comparable with a cosine similarity. It is a model with pooling operations added to the output of a pre-trained BERT. BERT is explained later. The pooling operation calculates the fixed-length vector of the entire sentence from the vector of each token in the output of BERT. Training SBERT is fine-tuning BERT using either a siamese network or a triplet network. A siamese network [5] is a neural network that uses two subnetworks to extract features from two inputs and measures the distance between the feature vectors. A triplet network [6] is an embedding network, which is trained by triplet loss. The proposed system is trained using a triplet network.

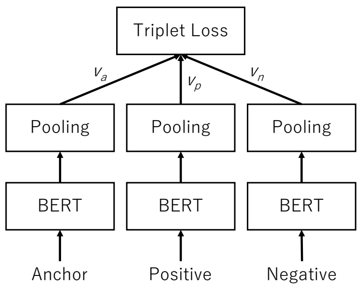

Fig. 3. Structure of a triplet network.

First, we explain the details of training SBERT using a triplet network. Figure 3 shows the structure of the triplet network. The input is a set of three sentences (triplet): anchor, positive, and negative. For the positive, a sentence similar to the anchor is chosen, while a sentence not similar to the anchor is used for the negative. SBERT is trained so that the distance between the anchor vector v_a and the positive vector v_p is at least a constant ε (margin) smaller than the distance between the anchor vector v_a and the negative vector v_n. The loss function used in this process is triplet loss, which is expressed as follows

$$\text{Triplet loss} = \max(\| v_a - v_p \| - \| v_a - v_n \| + \varepsilon, 0) \quad (1)$$

where $\| \cdot \|$ is a distance metric.

BERT [7] is a deep-learning-based language representation model. It is based on Transformer [8], a language representation model that connects encoders and decoders through an attention mechanism only. BERT is designed to pre-train deep bidirectional representations with unlabeled texts. Pre-training gives BERT understanding of context and sentence relationships. Simply fine-tuning a pre-trained BERT model with an output layer can build high-performance models for a wide range of natural language processing tasks. However, BERT is not well suited for large-scale semantic similarity comparisons because converting sentences into vectors comparable with cosine similarity is time consuming.

III. Duplicate Bug Report Detection Using Sentence Embedding and Fine-tuning

We propose a system that automatically detects if a new inquiry report is a duplicate of a previous inquiry or failure report for the maintenance described in Section II.

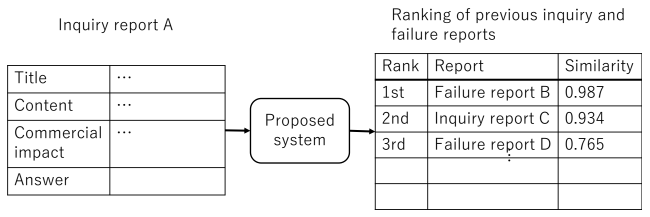
Inquiry report A
Title ···
Content ···
Commercial impact ···
Answer
Proposed system
Ranking of previous inquiry and failure reports
Rank Report Similarity
1st Failure report B 0.987
2nd Inquiry report C 0.934
3rd Failure report D 0.765

Fig. 4. Input and output of the proposed system.

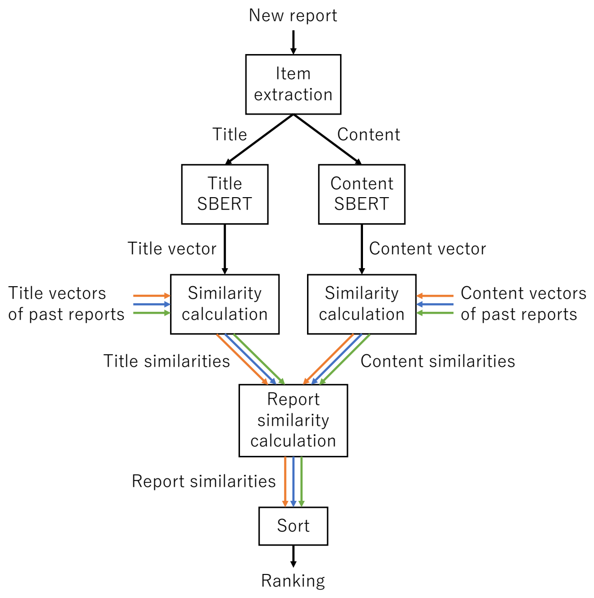

Fig. 5. Flow to detect duplicate reports in the proposed system.

A. Method for Detecting Duplicate Bug Reports

The first issue in achieving automatic detection of duplicate bug reports is how to judge whether bug reports are duplicates. Our method generates vectors from reports based on their components and judges whether reports with a high degree of vector similarity are duplicates. This solution is a standard method for duplicate bug report detection [9][10][11][12][13][14]. In addition, the reports targeted by our research have a fixed format and are completed by people inside the maintenance organization. Hence, they are generally written using a formula that follows explicit or implicit rules, although there are minor differences. Therefore, converting both new and previous reports into vectors using one conversion method that can absorb minor differences should be sufficient for easy comparison.

The second issue is how to convert bug reports into vectors that capture the characteristics of the problems being reported. We employ vectorizing report descriptions by item using sentence embedding because sentence embedding refers to a technique and a model for converting sentences into vectors that capture the characteristics of their contents. Since the reports targeted by our research are mainly written in a natural language, sentence embedding should be suitable to capture the characteristics of the problem. The reason for vectorizing by item is that each item in a report has different information.

The third issue is selecting a suitable sentence embedding model for bug reports. Our method uses a supervised learning model trained with general texts and further fine-tunes with texts from bug reports because bug reports contain many domain-specific and product-specific expressions, whose vocabulary differs from that of general texts. By training a model with texts from bug reports, expressions used in bug reports should be appropriately reflected in the vectors. On the other hand, since the number of reports available for training is small, the amount of training for general expressions is compensated by training the model with many general texts in advance.

In the proposed system, SBERT is used for sentence embedding. Because each inquiry or failure report is completed by different maintenance engineer, wording and explanations of the same problem may differ slightly between reports. Even for problems with the same cause, abnormalities that appear to users may vary. We expect that SBERT can absorb such differences well because it is a state-of-the-art sentence embedding method. Moreover, an attentive neural network such as SBERT can learn long-distance dependencies and is especially suitable for tasks that need to consider the context of long sentences [15]. Furthermore, a triplet network should absorb differences specific to reports of a particular organization or product by training the model with examples of duplicate reports.

The proposed system is designed to help maintenance engineers find duplicate reports. Figure 4 shows its input and output. The input is a new inquiry report with a title and content. The commercial impact and the answer can be either completed or left blank. The output is a ranking of previous inquiry and failure reports in descending order of similarity to the input report. If a new inquiry report A is inputted and the first rank in the output ranking is failure report B, then the report with the highest possibility of being a duplicate report of inquiry report A among previous reports is failure report B. Then a maintenance engineer checks the contents of the reports in order starting from the top report in the output ranking and judges whether they are a duplicate of the input report. Since the rank allows the reports to be referred in order of likelihood of being a duplicate, the search is more efficient.

Figure 5 depicts the flow for detecting duplicate reports inside the proposed system. First, the title and content of a new (input) report are extracted. Second, they are vectorized using the SBERT model for titles and the SBERT model for contents, respectively. Third, the similarity between the title vector of the new report and that of each previous report generated by the title model (the title similarity) is calculated. Additionally, the similarity between the content vector of the new report and that of each previous report (the content similarity) is calculated. For each previous report, the similarity to the new report as a whole (the report similarity) is calculated from the item similarities (the title similarity and the content similarity). Finally, previous reports are ranked and outputted according to the report similarity in descending order.

To realize the system, we need to build a title SBERT model and a content SBERT model, prepare the title and content vectors of previous reports, and decide how to calculate item similarities and a report similarity. The title and content models are built by first training them with large-scale Japanese texts and then training them with descriptions of inquiry and failure reports. Training details are explained later. The title vectors of previous reports are generated by vectorizing their titles using the title model. The content vectors of previous inquiry reports are generated by vectorizing their contents using the content model, while those of the previous failure reports are generated by vectorizing concatenations of their test environments and details using the content model. The title and the content vectors of all previous reports are stored so that the system can refer to them. Since vectors outputted by SBERT are comparable with the cosine similarity, the calculation method for both title and content similarities is cosine similarity. The calculation method for report similarity is explained later. Once the system is operational, additional learning of the model is conducted appropriately with newly created reports.

B. *Training of the Title Sentence-BERT Model and the Content Sentence-BERT Model*

Our system employs a pre-trained Japanese BERT model as a base to build title and content SBERTs. The experiments use the model by Inui and Suzuki Laboratory, Tohoku University [16]. SBERT is trained on a large triplet set of Japanese sentences using a triplet network.

The models in the experiment utilized a large-scale Japanese image caption dataset, STAIR Captions [17]. We prepared about 100,000 triplets, referring to [18]. The caption of each image was vectorized with GiNZA [19]. A caption pair whose cosine similarity was above a certain threshold was chosen as the anchor and the positive, while a caption of another image whose cosine similarity was below the threshold was chosen as the negative. Next, the system was trained on the triplet set prepared from the descriptions of inquiry and failure reports (report triplet set). The distance metric for triplet loss was the cosine distance (1 - cosine similarity). For the experiment, margin ε was set to 1, and the other hyperparameter settings followed those in the experiments in the paper proposing SBERT [1] and in the manual of its code.

We used the following procedure to prepare a report triplet set:

1. Extract the contents of the items to be used for the training from the reports.
2. Select a report with one or more duplicate reports among the reports for the training, and set the content of one of the items extracted as the anchor.
3. Select one of the duplicate reports of the report used for the anchor, and set the content of the item corresponding to the item of the anchor as the positive.
4. Select one report not related to the report used for the anchor, and set the content of the item corresponding to the item of the anchor or the positive as the negative.
5. Repeat steps 2 – 4 to prepare triplets with all possible combinations of reports and items, and use them as a report triplet set. Two triplets whose anchor and positive are swapped but whose negative is the same are identical; thus, only one is included in the report triplet set.

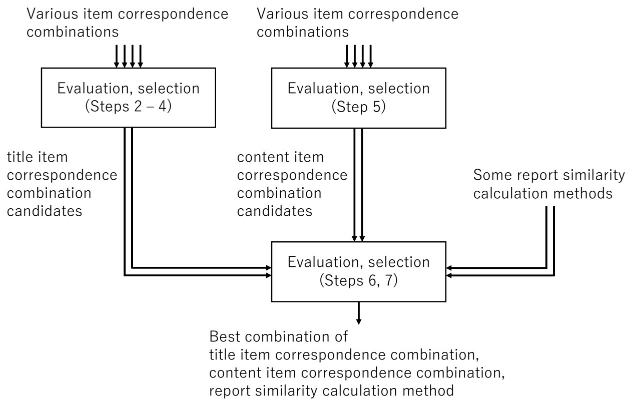
Various item correspondence combinations
Various item correspondence combinations
Evaluation, selection (Steps 2 – 4)
Evaluation, selection (Step 5)
title item correspondence combination candidates
content item correspondence combination candidates
Some report similarity calculation methods
Evaluation, selection (Steps 6, 7)
Best combination of title item correspondence combination, content item correspondence combination, report similarity calculation method

Fig. 6. Overview of the flow to decide the item correspondences for training models and the report similarity calculation method.

TABLE I. EXAMPLES OF TRIPLET

Anchor	Positive	Negative
(The title of report A) "Abnormal operation due to operation of function X"	(The title of report B) "Behavior of function X during process Y execution"	(The title of report C) "History cannot be displayed in function Z"
(The content of report A) "The system freezes when running process Y while using function X. …"	(The content of report B) "When function X is in process Y, *** will be started before function X is finished. …"	(The content of report C) "In function Z, when setting ***, an error occurs when trying to display the history. …"

The next section details how the item correspondences to be used for training the title and content models are determined.

C. *Selection of Item Correspondences for Training and Report Similarity Calculation Method*

Figure 6 overviews the flow for deciding the item correspondences for training the title and content models with inquiry and failure report descriptions (the title item correspondence combination, the content item correspondence combination) and the method for calculating report similarity from item similarities. First, candidates for title item correspondence combinations and those for content item correspondence combinations are selected according to the sequential selection method. Second, each combination of a title item correspondence combination candidate, a content item correspondence combination candidate, and a report similarity calculation method is evaluated. The system adopts the combination with the highest evaluation. The details of the procedure are as follows:

1) *Preparation*

a. Label each report pair in the dataset (previous reports) as a duplicate or unrelated.

b. Designate some inquiry reports with one or more duplicate reports in the dataset as the test data.

c. Extract the description of all the reports in the dataset separately for each item. Extracted items from the inquiry reports of the test data are the titles and the contents. Those from the inquiry reports that are not the test data are all the single items and the content + answers, which are the concatenations of the contents and the answers. Those from the failure reports are all the single items, all the single subitems, and the test environment + details, which are the concatenations of the test environments and the details.

2) Model building

a. Select item correspondences to be used for training the title model, and prepare the report triplet set from the reports other than the test data using the procedure explained above. Item correspondences of a pair of inquiry reports and a pair of failure reports are the same items. Those of a pair of an inquiry report and a failure report are the title of the inquiry report and the title of the failure report, the content + answer of the inquiry report and the test environment + details of the failure report. TABLE I shows the triplets for the case where inquiry reports A and B are duplicates, inquiry report C is not related to reports A and B, and the item correspondences between the titles and between the contents of the inquiry reports are selected.

b. Build a model using the report triplet set and the training method explained above.

3) Evaluation of the item correspondence combination

a. Vectorize the title of each report in the dataset with the built model and treat it as the title vector of the report.

b. For each report of the test data, calculate the cosine similarity of the title vector with each of the other reports (title similarity). For example, suppose the dataset is reports A, B, C, and D, and reports A and B are the test data. In this case, for report A, calculate the title similarity between reports A and B, between reports A and C, and between reports A and D. For report B, calculate the title similarity between reports B and A, between reports B and C, and between reports B and D.

c. For each report of the test data, rank the other reports according to the title similarities in descending order. Using the example in step 3b, for report A, create a ranking of reports B, C, and D according to their title similarities to A. For report B, create a ranking of reports A, C, and D according to their title similarities to B.

d. Evaluate each ranking with a ranking evaluation metric. In the system building for the experiment, AP was used as the ranking evaluation metric, as explained in Section IV.

e. Calculate the average evaluation scores of all the rankings as the evaluation score of the item correspondence combination selected in step 1b.

4) Selection of title item correspondence combination candidates

Repeat steps 2 and 3 but change the selection of item correspondences training, and select multiple item correspondence combinations whose evaluation scores are relatively high as title item correspondence combination candidates.

For the experiment, select the correspondence between the titles of inquiry or failure reports, and then change the item correspondences to be selected according to the forward-backward stepwise selection method. It should be noted that an item and its sub-item were not selected simultaneously in the experiment.

5) Selection of content item correspondence combination candidates

Select the content item correspondence combination candidates in the same way as the title item correspondence combination candidates (steps 2 – 4).

For each inquiry report, vectorize the content and treat it as the content vector of the report. For each failure report, vectorize the test environment + details and treat it as the content vector of the report.

Also, the first item correspondences to be selected are the correspondences between the contents of inquiry reports, between the content + answer of an inquiry report and the test environment + details of a failure report, and between the details of failure reports.

6) Evaluation of combinations of a title item correspondence combination, content item correspondence combination, and report similarity calculation method

a. Select a title item correspondence combination candidate and generate the title vector of each report in the dataset by the model trained with the combination as in step 3a. Similarly, select one of the content item correspondence combination candidates and generate the content vector of each report by the model trained with the combination in the dataset as explained in step 5. For each report of the test data, calculate the title similarities and the content similarities with each of the other reports similar to step 3b.

b. Select a method to calculate the report similarity from item similarities. For each report in the test data, calculate the report similarity to the other reports and rank the other reports according to the similarity in descending order similar to step 3c. Evaluate each ranking using the ranking evaluation metric.

c. Calculate the average of the evaluation scores of all the rankings as the evaluation score of the combination of the title item correspondence combination selected in step 6a, the content item correspondence combination selected in step 6a, and the report similarity calculation method selected in step 6b.

7) Selection of the best combination of a title item correspondence combination, a content item correspondence combination, and a report similarity calculation method

Repeat step 6, but change the item correspondence combinations selected in step 6a and the report similarity calculation method selected in step 6b to cover all combinations of a title item correspondence combination candidate, a content item correspondence candidate, and a report calculation method. In the experiment, we used the following seven report similarity calculation methods: i) Report similarity = title similarity, ii) content similarity, iii) mean(title similarity, content similarity), iv) 0.75 * title similarity + 0.25 * content similarity, v) 0.25 * title similarity + 0.75 * content similarity, vi) max(title similarity, content similarity), and vii) min(title similarity, content similarity).

The combination of a title item correspondence combination, a content item correspondence combination, and a report similarity calculation method with the highest evaluation score is adopted in the system. In the experiment, we employed the correspondences between the titles of inquiry or failure reports, between the contents of inquiry reports, between the answers of inquiry reports, between the test environments of failure reports, and between the feedback of failure reports as the title item correspondence combination. For the content item correspondence combination, we adopted the correspondences between the contents of inquiry reports, between the answers of inquiry reports, between the content + answer of an inquiry report and the test environment + details of a failure report, between the details of failure reports, and between the measures of failure reports. For the report similarity calculation method, we utilized report similarity = max(title similarity, content similarity).

IV. Industrial Experiment

We built three systems: the proposed system and two baselines. Then we compared their performances.

A. Baseline Systems

The baseline systems calculate the similarities of vectors generated from a document (a corpus) created from all items in reports using the term frequency-inverse document frequency (TF-IDF) or Latent Dirichlet Allocation (LDA) [20]. The content of a new report affects the vector generation in both new and previous reports. Hence, the baseline systems require that both the new and all previous reports are entered as inputs for each new detection. The output is a ranking in descending order of similarity to the new report. This is the same as the output of the proposed system. TF-IDF adds weights to words in a document. The weight of a word is calculated by multiplying the term frequency (tf) by the inverse document frequency (idf). The tf is a measure of the word frequency in the document, while idf is the degree of how rare a document containing the word is. LDA is a topical and generative statistical model for documents. Each document is expressed as a mixture of potential topics based on the concept that each topic can be characterized by its word distribution.

Figure 7 shows the flow for detecting the duplicate reports in the baseline systems. First, the contents of all items shown in Section II are concatenated for new and previous reports. Second, morphological analysis is performed on the texts of new and previous reports, and a set of all words in the original form, excluding symbols, is treated as a document for a report. Morphological analysis breaks a sentence into words and identifies the part of speech and the original form of the words. We used MeCab [21] to run morphological analysis. Third, vectors are generated from all documents, which are treated as report vectors. Fourth, cosine similarities are calculated between a new report vector and all previous report vectors. The output is a ranking of previous reports according to the similarity in descending order.

In TF-IDF, a word corresponds to the vector component. Consequently, the vector represents the number of dimensions, which is equal to the number of unique words in the document. The value of each vector component is the frequency of the corresponding word in the document multiplied by the weight of TF-IDF. We used scikit-learn [22] to generate vectors in TF-IDF.

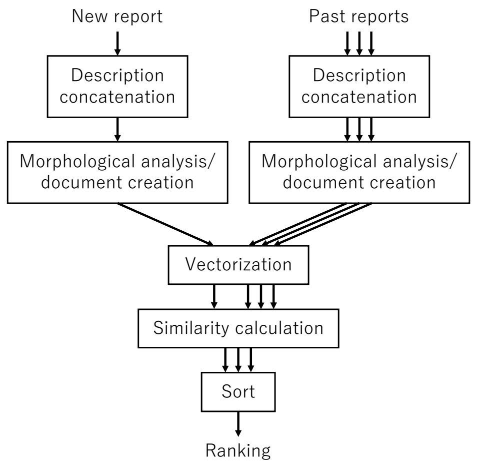
New report
Past reports
Description concatenation
Description concatenation
Morphological analysis/ document creation
Morphological analysis/ document creation
Vectorization
Similarity calculation
Sort
Ranking

Fig. 7. Flow to detect duplicate reports in the baseline systems.

TABLE II. Number of Triplets in a Report Triplet Set and the Mean Evaluation

	Title model	Content model
Number of triplets in a report triplet set	1200 – 1500	1100 – 1300
Mean evaluation score	0.985	0.987

In contrast, the LDA model is built from all the documents. We set the number of topics to ten. The vector represents the per-document topic distribution analyzed by the LDA model. We used gensim [23] to implement the LDA-based document topic analysis.

B. Dataset

The dataset used for the performance evaluation is the inquiry and failure reports for software product X, which has been developed, maintained, and operated by NTT Corporation since 2004. Product X employs the maintenance method that this study targets. Each report has 0 to 2 duplicate reports in the dataset.

TABLE II shows the details of the dataset. We selected all inquiry reports and corresponding failure reports issued in 2014. We chose them because the operation started ten years prior and the corresponding failure reports were likely already given in advance when inquiry reports were newly issued. Such a situation motivates us to avoid having duplicate reports.

C. Performance Evaluation

We evaluated the performance of the proposed system using the following procedure:

1. Divide all reports with one or more duplicate reports into five groups.
2. Designate one group as the test data. Use the rest as training data.
3. Train a title model and a content model on the reports, excluding the test data, to build the proposed system. Use the report triplet to evaluate models. Employ 20% of the

TABLE III. DATASET

	Inquiry reports	Failure reports
Total number of reports	25	26
Number of reports without duplicates	2	3
Number of reports with one duplicate	14	14
Number of reports with two duplicates	9	9

triplets as a set of evaluation report triplets and the remainder as a set of training report triplets. Title vectors and content vectors of all reports in the dataset are stored as previous reports.

4. For each report in the test data, remove the title vector and content vector of the report from those of the previous data, and input the report in the system. The output is the rank of all reports except for the input report.

5. From the output obtained from step 4, evaluate the system built in step 3 using an evaluation metric.

6. Repeat steps 2 – 5 using each of the five groups as a test dataset.

7. The performance value of the proposed system is the mean evaluation score.

The evaluation metric of a title or content model is derived as the proportion of triplets in the set of evaluation report triplets such that the distance between the anchor and positive vectors is smaller than that between the anchor and negative vectors where these vectors are generated by the evaluating model.

The evaluator in SBERT is used to evaluate a model's performance. Because the system is built five times, five sets of title and content models are created. TABLE III shows the range of the number of triplets in the report triplet set used for training and evaluating an instance of the model along with the mean evaluation score of five model instances.

We used the following procedure to evaluate the baseline systems:

1. Use an inquiry report with one or more duplicate reports in the dataset as a new report. Remove the content of the answer in the report and input the report into the system.

2. Input other reports as previous reports in the system. The system outputs the ranking.

3. For all inquiry reports with one or more duplicate reports, repeat steps 1 and 2 to input each one into the system as a new report.

4. From all the rankings, evaluate the performance of a baseline system using an evaluation metric.

In this study, the mean average precision (MAP) is used as the system evaluation metric. This metric evaluates the output ranking performance by a search system. Outputs obtained from multiple search queries are evaluated by the average precision (AP), a ranking evaluation metric. Then MAP is obtained by taking the mean value of multiple evaluation scores of the outputs. AP is expressed using precision of the top r ranks (Precision@r) as

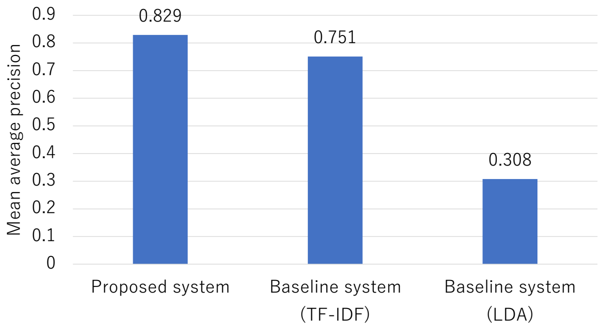

Fig. 8. Performance evaluation scores.

$$\mathrm{AP} = \frac{1}{n}\sum_{r} \mathrm{Precision@r} \cdot I(r) \tag{2}$$

where n is the number of matches for a search query (i.e., the number of duplicate reports of the input report in our research)

$$I(r) = \begin{cases} 1 \text{ (the rth ranked item is a match (duplicate))} \\ 0 \text{ (the rth ranked item is not a match (unrelated))} \end{cases} \tag{3}$$

A member of the maintenance team explained that if they employed the proposed system in their workflow, they would likely review the reports with the highest rankings to search for duplicate or similar reports. Therefore, we determined that MAP, which reflects the ranking of the duplicate reports, is a suitable evaluation metric. Figure 8 shows the performance evaluation scores of the proposed and two baseline systems.

D. Discussion

Based on the results of the experiment, we discuss answers for RQ1 and RQ2 below.

1) RQ1. For the target type of bug reports, can duplicate reports be identified?

The flow of the proposed system worked well for detecting duplicate reports (Fig. 5). SBERT was trained on general texts followed by the contents of the inquiry and failure reports. We chose this strategy after adapting the solutions to the issues to automatically detect duplicate bug reports and considering the features of SBERT.

RQ1. For the target type of bug reports, can duplicate reports be identified? **The detection flow implemented in the proposed system can identify duplicate reports. In the flow, SBERT is trained on general texts followed by the contents of the inquiry and failure reports.**

2) RQ2. Does the proposed system outperform the baseline systems?

The proposed system outperformed the baseline systems based on TF-IDF and LDA. The proposed system, which processes each item in a report separately and incorporates SBERT fine-tuned with bug reports, showed a higher performance evaluation score than the systems utilizing basic natural language processing techniques (TF-IDF or LDA, Fig. 8). Thus, for the reports targeted by this research, a combination of processing separately by item and fine-tuning SBERT with reports improves the performance of duplicate bug report detection.

The achievement in the proposed system is attributed to incorporating the characteristics of reports targeted by this research and the features of SBERT. Titles have short content and provide succinct descriptions of the problem phenomena. In contrast, the contents of the inquiry reports and test environment + details of failure reports cover the environment, settings, and operations at the time of the incident. They may even identify possible causes. Due to such differences, some expressions may appear in contents of inquiry reports and test environment + details of failure reports but not titles. Duplicate reports should have similarities in expressions of phenomena and similarities in expressions of environment, settings, operations, and causes. Thus, calculating title similarities and content similarities separately seems to be an effective approach.

Furthermore, the item on the report used to train the title model differed from that used to train the content model. The title model focused on expressions to describe phenomena to generate appropriate vectors, whereas the content model covered a wide range of detailed descriptions of problems to generate appropriate vectors. The constructed SBERT effectively generated vectors (TABLE III). The models captured differences in wording and explanations in inquiry and failure reports as well as the appearance of abnormality of product X by comprehending the context using the long-distance dependencies for training.

RQ2. Does the proposed system outperform the baseline systems? **The proposed system achieved a higher performance evaluation score than the baseline systems, which utilize basic natural language processing techniques. In the proposed system, the combination of processing separately by item and fine-tuning SBERT with reports improves the duplicate bug report detection performance.**

E. *Threats to Validity*

Most reports in the dataset used in the experiment had duplicate reports because the data was deliberately selected to contain duplicates. Therefore, it is likely that the reports stored in a real-world maintenance project have fewer duplicate reports than the dataset used in this study. This difference is a threat to external validity. In the future, we would like to verify this effect by conducting a performance evaluation using a larger dataset that contains more reports but fewer duplicates.

Another threat to external validity is that the evaluation was conducted using only a dataset of inquiry and failure reports for product X, which was the target of this research. However, the proposed system was built based on the characteristics of the report descriptions in the dataset to improve its performance by building models and selecting calculation methods. The proposed method may also be effective for inquiry and failure reports of other products. In the future, we would like to validate the proposed system by conducting performance evaluations using datasets of reports of other products, which employ the same maintenance method.

Another threat is that the format of bug reports depends on the maintenance method, which makes it difficult to directly apply the proposed system. In the future, we would like to generalize the proposed detection technique and validate its independence of the maintenance method by evaluating its performance using datasets of bug reports obtained from different maintenance methods.

V. Related Work

A. *Duplicate bug report detection research*

We introduced a system to detect duplicate bug reports using vector similarities, which were generated separately by items in multi-item bug reports. Other studies have used similar systems. Rodrigues et al. proposed a technique to output the duplication probability by consolidating vectors generated for fragmented elements of bug reports (product information and priority) and sentence parts [9]. Words in the sentence parts were categorized as either belonging to the summary or the description. Then vectors were generated according to these categories. They utilized the soft-attention alignment mechanism to generate vectors for sentence parts from word vectors. Using different vectors to compare different reports drastically differs from our proposed system.

Deshmukh et al. proposed a technique to encode structured information, short descriptions, and long descriptions using a different model for each element [10]. Then these models were joined to generate a vector for a report and duplicates were determined from similarities. The models were trained such that the similarities between the target report and duplicate reports became larger than those between the target report and unrelated reports. Although this approach is similar to ours, the technique proposed by Deshmukh et al. generates vectors using bi-LSTM for short descriptions and CNN for long descriptions. On the other hand, our proposed system uses an attentive neural network, which enables training using long-distance dependencies to generate vectors. Thus, our proposed system can generate more context reflective vectors.

B. *Studies Applying BERT to Bug Reports*

Studies have actively investigated BERT on documents related to software development and maintenance because fine-tuning based on the domain knowledge can process domain-specific expressions. Among these, we introduce a few examples that analyze bug reports using BERT. Ardimento et al. proposed a technique to input texts from bug reports into BERT with a classifier in the output layer to predict the time length to fix the bug [24]. Hirakawa et al. looked at bug reports for a target system, which was built from a combination of OSS (Open Source Software). They proposed a technique to classify reports by the OSS that caused the bug. Their technique employed report texts as inputs and BERT with a classifier in the output layer [25]. To the best of our knowledge, our research is the first to use BERT to detect duplicate bug reports.

VI. Conclusion and Future Work

We propose a deep-learning-based duplicate bug detection system for a specific maintenance method. The proposed system generates vectors from each item in a bug report using SBERT, calculates the similarities of reports from vector similarities, and outputs a ranking of previous reports in descending order of their similarities to a target report. SBERT was initially trained on general texts but was refined on texts from the reports. To evaluate the performance, we built baseline systems, which generate vectors from all items in a report using TF-IDF or LDA and output the ranking of previous reports. We evaluated the performance of both the proposed and baseline systems through industrial experiments. The proposed system attained a MAP evaluation score of .829 while the baseline systems with TF-IDF and LDA attained MAP evaluation scores of .751 and .308, respectively. The proposed system outperformed the baseline systems, demonstrating that the combination of processing separately by item and fine-tuning SBERT with reports can effectively detect duplicate bug reports. A preliminary concept of our method has been mentioned in a 2-page abstract presented at [26]; we presented its concrete mechanism and technical details with a comparative evaluation experiment in this paper.

In the future, it is necessary to evaluate our proposed system by comparing with: 1) a system that generates vectors from separate items in a report using a different technique, 2) a system that generates vectors from all items in a report using SBERT fine-tuned with texts in the reports, and 3) a system that generates vectors from separate items in a report using SBERT before fine-tuning with texts in the reports. This comparison will reveal the individual degree of contribution by item-wise processing and fine-tuning SBERT with the report texts on the performance. In addition, we would like to generalize the duplicate bug report detection technique in the proposed system by making it independent of the itemization in a report format and compare its performance with the state-of-the-art duplicate bug report detection techniques. A long-term goal is to incorporate the structure of products to improve performance in duplicate bug detection. Since the relationships of modules provide additional information about the source of a problem, devising a technique to identify similarities of issues from modules in the description of bug reports should be highly effective. Furthermore, findings in our duplicate bug report detection method would contribute to further analysis of bug reports, such as the analysis of typical patterns in the discussion of bug reports [27], and the deep-learning-based bug fixing time prediction [28] and severity prediction [29].

Acknowledgment

The authors would like to thank Yuta Koizumi, Zhenyu Xu, Akira Kanamaru, and Tatsuka Muramoto for their supports.

References

[1] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence embeddings using siamese BERT-networks," 2019 Conference on Empirical Methods in Natural Language Processing and 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), 2019, pp. 3982–3992.

[2] M. Zhang, Z. Li, G. Fu, and M. Zhang, "Dependency-based syntax-aware word representations," Artificial Intelligence, vol. 292, 103427, 2021.

[3] K. Ghosh, et al., "Retrieval of prior court cases using witness testimonies," 33rd International Conference on Legal Knowledge and Information Systems (JURIX), 2020, pp. 43–51.

[4] T. J. -J. Li, et al., "Multi-modal repairs of conversational breakdowns in task-oriented dialogs," 33rd Annual ACM Symposium on User Interface Software and Technology (UIST), 2020, pp. 1094–1107.

[5] J. Bromley, et al., "Signature verification using a "Siamese" time delay neural network," 6th International Conference on Neural Information Processing Systems (NIPS), 1993, pp. 737–744.

[6] F. Schroff, D. Kalenichenko, and J. Philbin, "FaceNet: A unified embedding for face recognition and clustering," 2015 Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp. 815–823.

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

[8] A. Vaswani, et al., "Attention is all you need," 31st Annual Conference on Neural Information Processing Systems (NIPS), 2017, pp. 5999–6009.

[9] I. M. Rodrigues, D. Aloise, E. R. Fernandes, and M. Dagenais, "A soft alignment model for bug deduplication," 17th International Conference on Mining Software Repositories (MSR), 2020, pp. 43–53.

[10] J. Deshmukh, K. M. Annervaz, S. Podde, S. Sengupta, and N. Dubash, "Towards accurate duplicate bug retrieval using deep learning techniques," 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2017, pp. 115–124.

[11] A. Kukkar, R. Mohana, Y. Kumar, A. Nayyar, M. Bilal, and K. -S. Kwak, "Duplicate bug report detection and classification system based on deep learning technique," IEEE Access, vol. 8, 2020, pp. 200749–200763.

[12] G. Xiao, X. Du, Y. Sui, and T. Yue, "HINDBR: Heterogeneous information network based duplicate bug report prediction," 31st IEEE International Symposium on Software Reliability Engineering (ISSRE), 2020, pp. 195–206.

[13] T. Akilan, D. Shah, N. Patel, and R. Mehta, "Fast detection of duplicate bug reports using LDA-based topic modeling and classification," 2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC), 2020, pp. 1622–1629.

[14] S. S. Sehra, T. Abdou, A. Başar, and S. K. Sehra, "Amalgamated models for detecting duplicate bug reports," 33rd Canadian Conference on Artificial Intelligence (Canadian AI), 2020, pp. 470–482.

[15] K. Babić, S. Martinčić-Ipšić, and A. Meštrović, "Survey of neural text representation models," Information, vol. 11, no. 11, 2020, pp. 1–32.

[16] https://github.com/cl-tohoku/bert-japanese

[17] Y. Yoshikawa, Y. Shigeto, and A. Takeuchi, "STAIR captions: Constructing a large-scale Japanese image caption dataset," 55th Annual Meeting of the Association for Computational Linguistics (ACL), 2017, pp. 417–421.

[18] K. Uno (June 23, 2020). "Natural language processing for beginners part 9: verification of similar sentence retrieval using Sentence BERT" https://www.ogis-ri.co.jp/otc/hiroba/technical/similar-document-search/part9.html (accessed January 18, 2021) (in Japanese)

[19] https://megagonlabs.github.io/ginza/

[20] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent Dirichlet allocation," Journal of Machine Learning Research, vol. 3, 2003, pp. 993–1022.

[21] https://taku910.github.io/mecab/

[22] F. Pedregosa, et al., "Scikit-learn: Machine learning in Python," Journal of Machine Learning Research, vol. 12, 2011, pp. 2825–2830.

[23] R. Řehůřek and P. Sojka, "Software framework for topic modelling with large corpora," LREC 2010 Workshop on New Challenges for NLP Frameworks, 2010, pp. 45–50.

[24] P. Ardimento and C. Mele, "Using BERT to predict bug-fixing time," 2020 IEEE Conference on Evolving and Adaptive Intelligent Systems (EAIS), 2020, 9122781.

[25] R. Hirakawa, K. Tominaga, and Y. Nakatoh, "Study on automatic defect report classification system with self attention visualization," 2020 IEEE International Conference on Consumer Electronics, 2020, 9043062.

[26] H. Isotani, et al., "Detecting duplicate bug reports through Sentence embedding and fine tuning," Software Engineering Symposium, 2021, pp. 1-2. (in Japanese)

[27] Y. Noyori, et al., "What are the features of good discussions for shortening bug fixing time?", IEICE Transactions on Information and Systems, Vol. E104-D, No. 1, 2021, pp.106-116.

[28] Y. Noyori, et al., "Extracting features related to bug fixing time of bug reports by deep learning and gradient-based visualization," IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA), 2021, pp. 402-407.

[29] Q. Liu, H. Washizaki, and Y. Fukazawa, "Adversarial Multi-Task Learning-Based Bug Fixing Time and Severity Prediction," IEEE Global Conference on Consumer Electronics (GCCE), 2021, pp.1-2.