TYPE Original Research
PUBLISHED 19 January 2023
DOI 10.3389/fcomp.2022.1032452

Sentence embedding and fine-tuning to automatically identify duplicate bugs

Haruna Isotani[1], Hironori Washizaki[1]*, Yoshiaki Fukazawa[1], Tsutomu Nomoto[2], Saori Ouji[3] and Shinobu Saito[3]

[1]Department of Computer Science and Engineering, Waseda University, Tokyo, Japan, [2]Software Innovation Center, NTT CORPORATION, Tokyo, Japan, [3]Computer & Data Science Laboratories, NTT CORPORATION, Tokyo, Japan

Check for updates

OPEN ACCESS

EDITED BY
Yingxu Wang,
University of Calgary, Canada

REVIEWED BY
Heitor Costa,
Universidade Federal de Lavras, Brazil
Muhammad Aamir,
Huanggang Normal University, China

*CORRESPONDENCE
Hironori Washizaki
✉ washizaki@waseda.jp

SPECIALTY SECTION
This article was submitted to Software, a section of the journal Frontiers in Computer Science

RECEIVED 30 August 2022
ACCEPTED 30 December 2022
PUBLISHED 19 January 2023

CITATION
Isotani H, Washizaki H, Fukazawa Y, Nomoto T, Ouji S and Saito S (2023) Sentence embedding and fine-tuning to automatically identify duplicate bugs. *Front. Comput. Sci.* 4:1032452. doi: 10.3389/fcomp.2022.1032452

Industrial software maintenance is critical but burdensome. Activities such as detecting duplicate bug reports are often performed manually. Herein an automated duplicate bug report detection system improves maintenance efficiency using vectorization of the contents and deep learning–based sentence embedding to calculate the similarity of the whole report from vectors of individual elements. Specifically, sentence embedding is realized using Sentence-BERT fine tuning. Additionally, its performance is experimentally compared to baseline methods to validate the proposed system. The proposed system detects duplicate bug reports more effectively than existing methods.

KEYWORDS

bug reports, duplicate detection, BERT, sentence embedding, natural language processing, information retrieval

1. Introduction

Software often contains bugs, which are issues with the code. Potential bugs identified by the user are submitted *via* a bug report. Bug reports assist with software maintenance, which is necessary to ensure the usefulness and functionality of industrial software. However, triaging bug reports can be time-consuming and burdensome for maintenance engineers. When a bug report is received, the first step is determining whether the report is for a new issue or a duplicate of an existing issue. Herein an automated duplicate bug report detection system is described.[1] This system aims to assist in detecting duplicate bug reports for software developed and maintained by the NTT Corporation.

In our system, sentence embedding (distributed representations of semantics) generates vectors from the description content for each element in a bug report. Subsequently, the semantic similarities of reports are calculated from vector representation similarities. Duplicate reports are identified by the degree of similarity. The higher the degree of similarity, the higher the likelihood a report is a duplicate. Our system employs Sentence-BERT (SBERT) (Reimers and Gurevych, 2019) for sentence embedding. Specifically, SBERT is trained on general texts and fine-tuned with texts in bug reports. Adjusting the text in bug reports used for tuning realizes a specialized SBERT for each element found in a report.

To demonstrate the usefulness of our system, it is experimentally compared with two baselines, which were built with vectors generated using all elements in bug reports and natural

1 This paper substantially extends our preliminary conference paper presented at the IEEE International Conference on Software Maintenance and Evolution, ICSME 2021 (Isotani et al., 2021). Explanations of the proposed method, experiment, and related works are considerably revised and expanded in a well-structured paper format.

language processing techniques. The results confirm that the proposed system detects more duplicate bug reports than the baselines.

This paper addresses the following research questions:

RQ1. **Is it possible to identify duplicate reports?** Directly comparing bug reports is challenging because each report is composed of many elements. Moreover, the type of bug report may affect the ability to detect duplicate reports, as well as the optimal performance. RQ1 examines the elements impacting the ability to detect duplicate bug reports and the optimal technique to detect duplicate bug reports.

RQ2. **Does the proposed system outperform the baselines?** The proposed system processes elements separately and subsequently fine-tunes Sentence-BERT (SBERT) with bug reports. Determining the benefits of separating processing by element and fine-tuning SBERT may improve our system so that it is applicable in practical situations. To address RQ2, the performance of the proposed system is compared to baselines built using basic natural language processing techniques to generate vectors from all elements.

This study has two main contributions:

- An automatic duplicate bug report detection system is proposed.
- The proposed system outperforms the baselines.

The rest of this paper is organized as follows. The background is detailed in Section 2. The proposed system is overviewed and evaluated in Sections 3, 4, respectively. Related works are introduced in Section 5. Finally, the conclusions and future work are presented in Section 6.

2. Background

The objective of our proposed system is to support software maintenance processes by detecting duplicate bug reports employing SBERT. In this section, we explain the target maintenance process and details of SBERT.

2.1. Target maintenance process

Our research targets a maintenance flow that begins when a user finds a problem (Figure 1). The details from the time a problem is identified until the user receives a response of the findings are shown in Figure 2. In the target system, maintenance involves three types of bug reports. The first is generated when the user reports a problem to the reception center (user report). The second occurs when the reception center sends a request to the maintenance team to investigate the problem (inquiry report). The final report contains the investigation result and solution proposed by the maintenance team (failure report). All reports are written in Japanese.

When the reception center receives a user report, it examines manuals, screen displays, and logs to resolve the issue. If successful, the center responds to the user. If unsuccessful, the center initiates an inquiry report for the maintenance team. The inquiry report describes the problem and the opinions of the reception center. After reviewing the design document and code, the maintenance team reports the findings back to the reception center if the issue has been previously identified. If the issue is due to a new bug, the maintenance team generates a failure report. The failure report includes details about the problem. Additionally, once the issue is resolved, its origin and how to prevent it are added to the failure report. Then the failure report is shared with the reception center and the user. Finally, the development vendor fixes the bug.

As part of the investigation, the maintenance team determines whether the bug is due to the problem identified in the inquiry report. Not only can this be time-consuming and burdensome, but it also requires highly skilled maintenance engineers because each previous inquiry and failure report is analyzed individually. As such, skills and knowledge regarding a specific bug may not be transferred adequately. To overcome this limitation, this study devises a system to automatically detect whether an inquiry report is for a new issue or a duplicate.

Inquiry reports are composed of the following elements:

- Title: Summary of the reported problem
- Content: Description of the issue, including the phenomena, environment, settings, and operations at the time of the incident
- Commercial impact (optional): Impact of the problem on the service and users
- Answer: Results of the maintenance team's investigation.

The first three are completed by the reception center. The maintenance team logs the last one by either providing a solution if the issue is due to a previously reported bug or by indicating that a failure report was generated because the issue is due to a new bug.

Failure reports contain the following elements:

- Title: Summary of the problem described in the failure report, which differs from the title of the inquiry report
- Test environment: Environment in which the problem was observed
- Details: Details of the problem from the inquiry report and the initial opinions of the maintenance team.
- Origin of failure: This is further divided into:
  - Cause: Root issue
  - Conditions: Conditions in which the problem occurs
  - Effect: Abnormal behavior when the problem occurs and troubles for users
  - Workaround: A temporary solution until the bug is fixed
  - Affected version: Versions impacted by the problem.
- Measures: This is further divided into:
  - Measure policy: How to fix the bug
  - Source: A simple description of where and how to modify the source code, which may refer to a different file
  - Document: A simple description of where and how to modify the document, which often refers to a different file.
- Feedback: This is further divided into:
  - Injection cause: Cause of the bug, including background and history
  - Reason for delayed removal: Explanation of why the bug was not found before the release

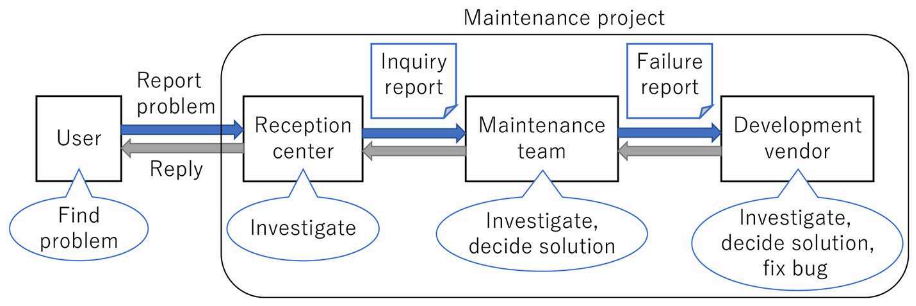

FIGURE 1
Overview of the maintenance process.

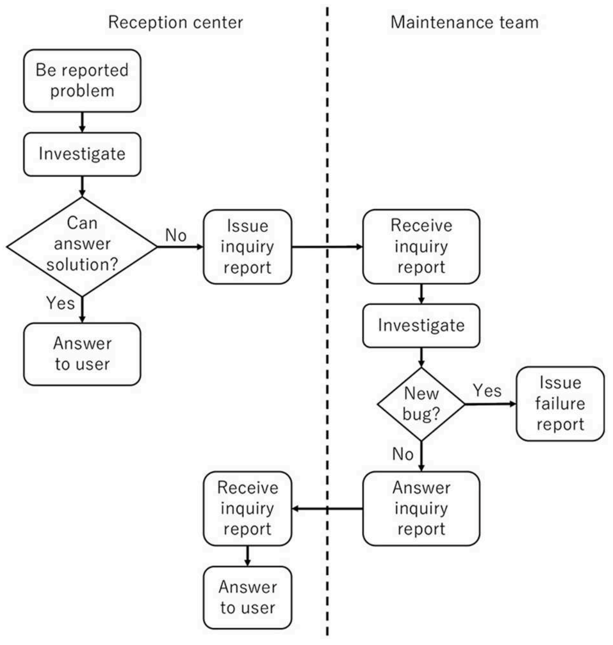

FIGURE 2
Maintenance process of the reception center and the maintenance team.

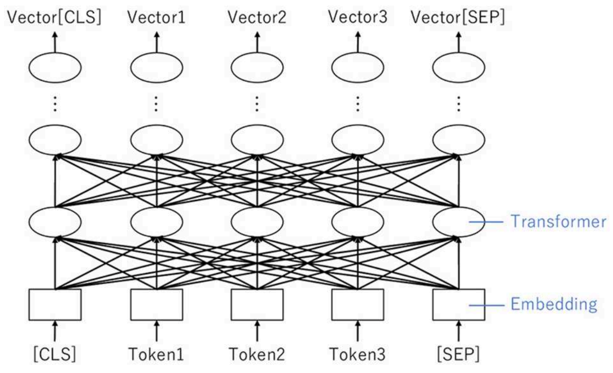

FIGURE 3
Structure of BERT.

- Inspection: The inspection results for similar bugs in parts of the product with a comparable structure
- Measures to prevent recurrence: Prevention measures to prevent similar bugs in the future.

All elements are documented by the maintenance time. The test environment and the details are documented through detailed investigation by the maintenance team with reference to the details from the inquiry report. Vendor developers may contribute to the failure cause, the measures, and the feedback. In our proposed system, the contents of these elements are used to detect duplicate reports.

2.2. Sentence-BERT

Sentence-BERT (SBERT) is an extension of BERT (Devlin et al., 2019). BERT is a deep learning–based language representation mode based on Transformer (Vaswani et al., 2017) (Figure 3). Unlike Transformer, which is a language representation model that connects encoders and decoders through an attention mechanism, BERT pre-trains deep bidirectional representations with unlabeled texts. Pre-training provides context and sentence relationships. The advantage of BERT is that a high-performance model for diverse natural language processing tasks can be realized by simply fine-tuning. The drawback is that BERT is inefficient for large-scale semantic similarity comparisons because converting sentences into vectors comparable to cosine similarity is time-consuming.

SBERT extends BERT to include sentence embedding (Reimers and Gurevych, 2019). It embodies the content features by converting sentences into vectors. Since being published in 2019, SBERT has been a state-of-the-art method (Ghosh et al., 2020; Li et al., 2020; Zhang et al., 2021). The vector output is similar to the cosine similarity. As a model, SBERT adds pooling operations to the output of a pre-trained BERT (Figure 4). Training involves fine-tuning using either a Siamese or triplet network. A Siamese network is a type of neural network that employs two subnetworks to extract features from two inputs and then measures the distance between the feature vectors (Bromley et al., 1993). In contrast, the triplet network is a type of embedding network trained by triplet loss (Schroff et al., 2015). The additional pooling operation calculates a fixed-length vector of the entire sentence from each token vector.

Our proposed system adopts a triplet network. The triplet network uses the triplet loss function and three types of sentences

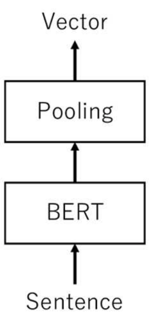

FIGURE 4
Structure of sentence BERT.

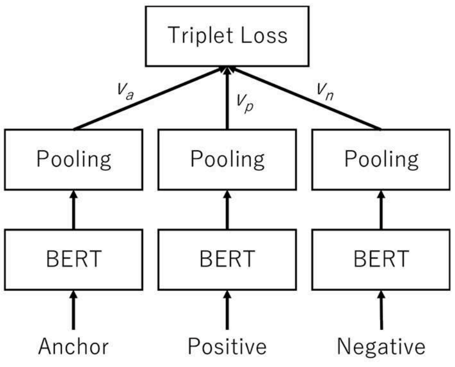

FIGURE 5
Structure of the triplet network.

as inputs (Figure 5). The first is an anchor. The second is a positive, which is similar to the anchor. The third is a negative, which differs from the anchor. SBERT is trained to ensure that the distance between the anchor vector v_a and the positive vector v_p is a constant ϵ (margin), which is smaller than that between the anchor vector v_a and the negative vector v_n. The loss function is expressed as:

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

where ‖ ... ‖ is a distance metric.

3. Proposed duplicate bug detection system

Here, a system is developed for the maintenance process described in Section 2.1. Our system automatically detects whether an inquiry report is new or a duplicate of a previous one.

3.1. Duplicate bug report detection

A system must overcome three challenges to assist maintenance engineers in identifying duplicate bug reports. The first is devising a process to evaluate whether a bug report is new or a duplicate. Consistent with best practices (Deshmukh et al., 2017; Akilan et al., 2020; Kukkar et al., 2020; Rodrigues et al., 2020; Sehra et al., 2020; Xiao et al., 2020), our proposed system uses the components of the bug reports for this assessment. Because our research targets an organization with a standard format for the bug reporting process, the elements in bug reports follow a general formula with explicit or implicit rules. Although each report has minor differences, this barrier can be overcome by converting elements or reports into vectors.

The second is converting bug reports into vectors that capture the characteristics of the reported problems. To address this challenge, our system vectorizes descriptions by element type in the report using sentence embedding. This approach should be suitable to clarify the characteristics of the problem because target reports are written in a natural language. In our system, reports are vectorized by element because each report contains unique information. Here, sentence embedding is both a technique and a model. It converts sentences into vectors that capture the characteristics of the content.

The third is selecting an appropriate sentence embedding model. Our method uses supervised learning trained with general texts and fine-tuned with texts from bug reports. Supervised learning uses general texts because the data available for training using only bug reports is limited. However, fine-tuning with bug reports captures the vocabulary of the domain-specific and product-specific expressions found in the bug reports. This approach should ensure that the vectors reflect expressions in the bug reports.

To overcome the above challenges, our proposed system adopts SBERT for sentence embedding and a triplet network. We chose SBERT because it should absorb the variations in the reports. Not only does the target maintenance system have three different types of reports but reports on the same bug may also differ because the wording and explanations depend on the person completing the report. Additionally, as an attentive neural network, SBERT can learn long-distance dependencies and is appropriate for tasks that include the context of long sentences (Babic et al., 2020). We chose a triplet network as it should incorporate differences specific to reports for an organization or a product because it trains the model using examples of duplicate reports.

Figure 6 shows the input and output of our system, where A is the new report and B–D are the outputs. Inputs are the title and content of an inquiry report. It should be noted that optional content (e.g., commercial impact) may be omitted. The output is a list of previous inquiry and failure reports in descending order of similarity. In this example, report B has the highest similarity score to A followed by reports C and D. The maintenance engineer uses the results of our proposed system to evaluate if A is a duplicate report. The engineer reviews the content of the outputted reports starting with the one with the highest similarity. The search is more efficient because the list is a ranking of the likelihood of being a duplicate.

Our proposed system has four steps (Figure 7):

- Step 1. Title and content extraction in Figure 7: Our system receives a new report as input and extracts the title and contents of the input report.
- Step 2. Title SBERT and content SBERT in Figure 7: Our system vectorizes each title and content element using two SBERT models to generate title and content vectors independently.
- Step 3. Similarity calculation and report similarity calculation in Figure 7: Our system calculates title similarities between the title vector of the input report and the title vectors of past reports. In

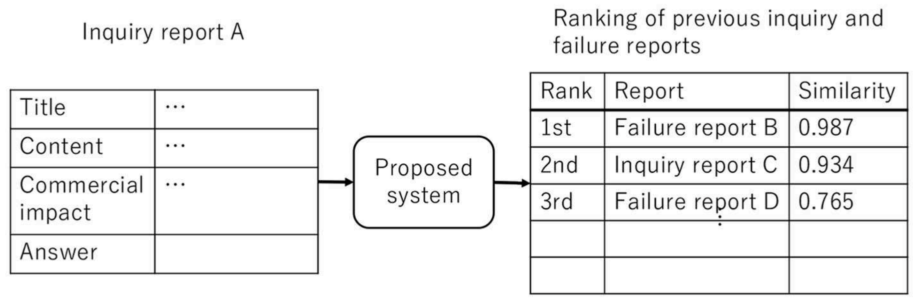
Inquiry report A
Title …
Content …
Commercial impact …
Answer
Proposed system
Ranking of previous inquiry and failure reports
Rank | Report | Similarity
1st | Failure report B | 0.987
2nd | Inquiry report C | 0.934
3rd | Failure report D | 0.765

FIGURE 6
Inputs and outputs of the proposed system.

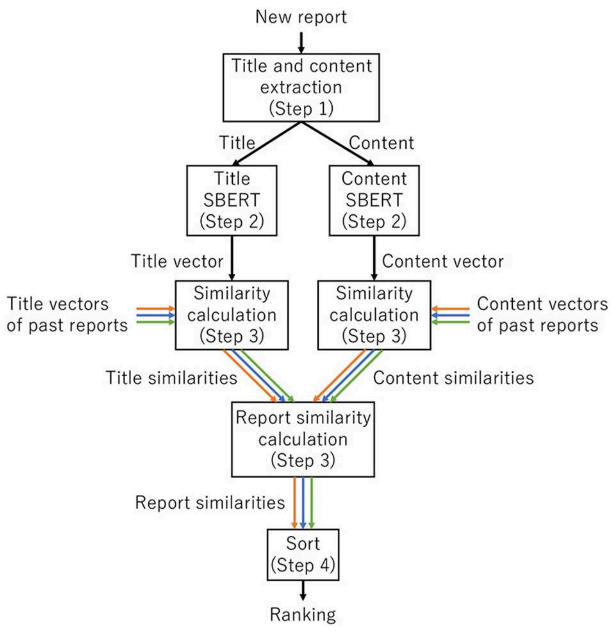
New report
Title and content extraction (Step 1)
Title
Content
Title SBERT (Step 2)
Content SBERT (Step 2)
Title vector
Content vector
Title vectors of past reports
Similarity calculation (Step 3)
Similarity calculation (Step 3)
Content vectors of past reports
Title similarities
Content similarities
Report similarity calculation (Step 3)
Report similarities
Sort (Step 4)
Ranking

FIGURE 7
Overview of the flow in proposed system.

parallel, our system calculates content similarities between the input report's content vector and past reports' content vectors. After that, our system combines the title similarity and the content similarity by calculating various weighted similarities and the max and min of two similarities and selecting the best one.

- Step 4. Sort in Figure 7: Our system outputs a list of past reports in descending order of similarity.

Our system contains two SBERT models: one for the title and another for the content. These models are initially built by training with large amounts of Japanese text. Then they are fine-tuned using descriptions from actual existing reports. After fine-tuning and deployment of the system, additional learning can be conducted continuously using newly created reports.

The title and the content vectors of previous reports are generated by vectorizing their title and contents. The title and the content vectors of all previous reports are stored so they can be recalled later. Since SBERT outputs vectors comparable to the cosine similarity, our system calculates the similarity scores between the input and previous reports and outputs a list of previous inquiry and failure reports in descending order of similarity. As the report similarity, our system combines the title similarity and the content similarity by calculating various weighted similarities (such as 0.75 * title similarity + 0.25 * content similarity) as well as the max and min of two similarities and selects the best one.

3.2. Model training

A pre-trained Japanese BERT model[2] by Inui and Suzuki Laboratory, Tohoku University, serves as the base of our system. SBERT is built by a large triplet set of Japanese sentences using a triplet network.

The experimental models use a large-scale Japanese image caption dataset, STAIR Captions (Yoshikawa et al., 2017). By referring to the literature (Uno, 2020), 100,000 triplets were prepared. Each captioned image was vectorized with GiNZA.[3] The threshold for the cosine similarity was set. In the experiment, the margin ϵ was set to 1, the same as that set in the literature of the Sentence BERT proposal (Reimers and Gurevych, 2019). The other hyperparameters were also set by referring to the literature and manual of SBERT's code (Reimers and Gurevych, 2019). In the future, we plan to conduct more experiments using different thresholds and hyperparameters to confirm their impacts on the result.

Caption pairs with values above the threshold are selected as the anchor and the positive. In contrast, a pair with a value below the threshold is selected as the negative. Once the anchor, positive, and negative are set, the system is trained on the report triplet set. The distance metric for triplet loss is the cosine distance (1-cosine similarity).

Each report triplet set is prepared using four steps:

- Step 1: Extract the contents for the training from reports.
- Step 2: Select a report with at least one duplicate report and randomly set the content of one element as the anchor.
- Step 3: Select a duplicate report of the one selected in Step 2. Set the element corresponding to the anchor as the positive.

2 https://github.com/cl-tohoku/bert-japanese

3 https://megagonlabs.github.io/ginza/

- Step 4: Select a non-duplicate report of the one selected in Step 2. Set the element corresponding to the anchor as the negative.

Triplets are prepared with all possible combinations by repeating the above steps. Note that triplets whose anchor and positive are swapped but have the same negative are identical, and only one should be included in the report triplet set.

3.3. Training elements and report similarity calculations

Element correspondence for training the title and content models involves three steps (Figure 8):

- Step 1: Prepare data and extract elements.
- Step 2: Prepare a report triplet set from all reports and build a model.
- Step 3: Use a sequential selection method to determine candidates for combinations of corresponding title elements.
- Step 4: Evaluate each title element correspondence combination.
- Step 5: Use a sequential selection method to determine candidates for combinations of corresponding content elements and evaluate each combination.
- Steps 6 and 7: Calculate the similarity score. Our proposed system adopts the combination with the highest evaluation. Our system combines the title similarity and the content similarity as the report similarity by calculating various weighted similarities, max, and min of two similarities and selecting the best one.

The details of the procedure are as follows.

3.3.1. Data preparation

Data is treated in three steps:

- Step 1a: Label each report pair in the dataset as a duplicate or unrelated.
- Step 1b: Identify test data as inquiry reports with at least one duplicate report in the dataset.
- Step 1c: Separately extract the description for each element. Extracted elements include the titles and the contents from the inquiry reports. Additionally, single elements and the content + answers, which are the concatenations of the contents and the answers, are extracted. Extracted elements from the failure reports are all single elements, single sub-elements, and the test environment + details, which are the concatenations of the test environments and the details.

3.3.2. Model building

The model is built using three steps:

- Step 2a: Identify the element correspondences to train the title or content model.
- Step 2b: Prepare a report triplet set from all reports previously identified as training data.
- Step 2c: Build a model using the report triplet set and the above training method.

Note that the correspondences of a pair of inquiry reports and a pair of failure reports are the same elements. Elements from an inquiry report and a failure report are their titles, contents, answers to inquiries, test environment, and failure details. For example, Table 1 shows the triplets when inquiry reports A and B are duplicates, inquiry report C is unrelated, and the element correspondences between the titles and between the contents of the inquiry reports are selected.

3.3.3. Evaluation of title element correspondence combinations

Title element correspondence is determined using four steps:

- Step 3a: For each report, vectorize the title and treat it as the title vector of the report.
- Step 3b: Calculate the cosine similarity of the test vector for each report in the test data. For example, if there are reports A–D, where A is the test data, calculate the similarity between A & B, A & C, and A & D.
- Step 3c: List the results from Step 3b in descending order.
- Step 3d: Assess each ranking using a ranking evaluation metric. This includes calculating the average evaluation scores for all the rankings of the element correspondence.

The experiment used average precision (AP) as an evaluation metric to build the system.

3.3.4. Selection of title element correspondence combination candidates

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

In the experiment, the correspondence between the titles of inquiry or failure reports is selected. Thus, the element correspondences are determined according to the forward-backward stepwise selection method. The experiment excludes the simultaneous selection of an element and its sub-element.

3.3.5. Selection of content element correspondence combination candidates

This is determined in the same way as selecting title element correspondence combination candidates (Section 3.3.3), except content elements and not report titles are used. The vectorized content in the inquiry report is treated as the content vector. The vectorized test environment + details of the failure report are treated as the content vector. When selecting the element correspondence, start with the correspondences between the contents of inquiry reports followed by that between the content + answer described in an inquiry report, the test environment + details described in a failure report, and the details described in failure reports.

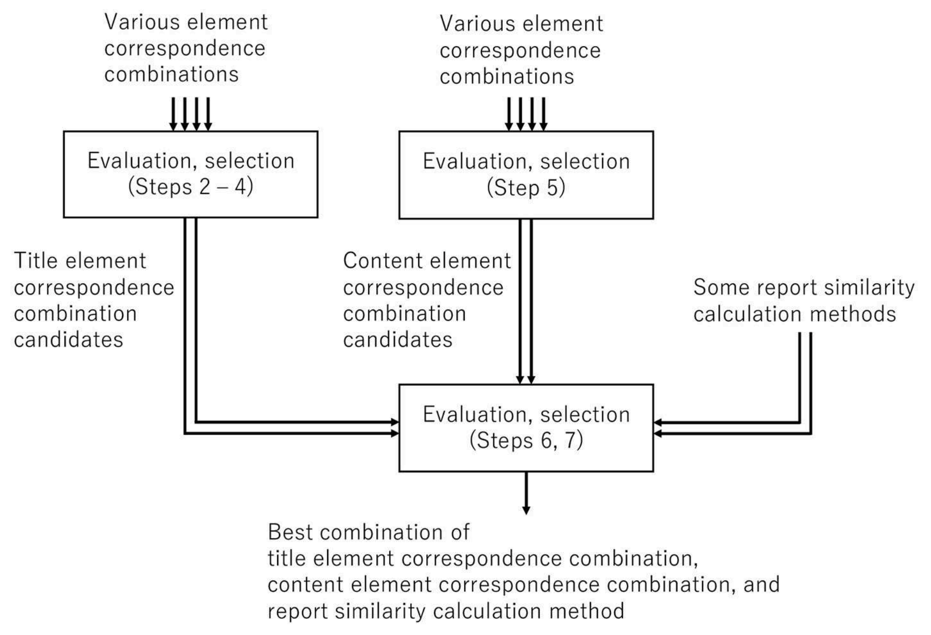
Various element correspondence combinations
Evaluation, selection (Steps 2 – 4)
Various element correspondence combinations
Evaluation, selection (Step 5)
Title element correspondence combination candidates
Content element correspondence combination candidates
Some report similarity calculation methods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

FIGURE 8
Determination of element correspondences for training models and the report similarity calculation method.

TABLE 1 Example of a triplet.

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

3.3.6. Combinations of a title element and content element

Combinations of title and content elements are evaluated using five steps:

- Step 6a: Select a title element correspondence combination candidate and generate the title vector for each report in the dataset using the model.
- Step 6b: Select a content element correspondence combination candidate and generate the content vector for each report using the model trained with the combination in the dataset.
- Step 6c: Calculate the title similarities and the content similarities with each of the other reports.
- Step 6d: Calculate the report similarity and list the results in descending order.
- Step 6e: Evaluate the rankings using the evaluation metric. Calculate the average evaluation score.

3.3.7. Best combinations and similarity calculations

Repeat step 6 (Section 3.3.6), but change the element correspondence combinations selected in step 6a and the report similarity calculation method chosen in step 6b to cover all combinations of a title element correspondence combination candidate, a content element correspondence candidate, and a report similarity calculation method.

The experiment uses the following report similarity calculations:

- Weighted similarities
  - 1.0 * title similarity
  - 1.0 * content similarity
  - 0.5 * title similarity + 0.5 * content similarity
  - 0.75 * title similarity + 0.25 * content similarity
  - 0.25 * title similarity + 0.75 * content similarity
- Max (title similarity, content similarity)
- Min (title similarity, content similarity).

The system adopts the highest evaluation score using the title element correspondence combination, content element correspondence combination, and the report similarity calculation method.

4. Experiment

The experiment compares our proposed system to two baselines.

4.1. Baseline systems

Vector similarities in the baseline systems are calculated from a document (corpus) created from all elements in a report. Two baselines were prepared. One used the term frequency-inverse document frequency (TF-IDF). The other used Latent Dirichlet Allocation (LDA) (Blei et al., 2003). We selected these approaches as the baselines since these are well-accepted fundamental ones and are often adopted as comparative baselines in text and document similarity measurements and other text processing tasks (Dai et al., 2015; Shahmirzadi et al., 2019). Furthermore, these still work well in certain areas, even against state-of-the-art sentence embedding approaches; for example, we have confirmed and reported that TF-IDF worked better than approaches based on Universal Sentence Encoder (USE) (Cer et al., 2018) and SBERT in linking security-related documents based on similarity measurements (Kanakogi et al., 2022). Thus, it is worth comparing our system with these baselines. In the future, we plan to compare our system with approaches based on other existing state-of-the-art embedding models (such as USE, Cer et al., 2018; InferSent, Conneau et al., 2017; and GenSen, Subramanian et al., 2018) as additional baselines to confirm the model performance.

Each new input of the baselines requires data from not only the new report but also all previous reports because the content of the new report affects the vector generation of all reports. Similar to our proposed system, the baselines output a list of similarities to the new report in descending order.

Each baseline has four steps to detect duplicate reports (Figure 9):

- Step 1: Concatenate the contents for all elements in the new and previous reports.
- Step 2: Analyze the texts of new and previous reports morphologically.
- Step 3: Generate vectors from all documents.
- Step 4: Calculate the cosine similarity and output list in descending order.

In Step 2, a set of all words in the original form, excluding symbols, is treated as a document. Morphological analysis divides a sentence into words and identifies the part of speech. In the experiment, morphological analysis employs MeCab.[4]

TF-IDF weights the words in a document. The weight is calculated by multiplying the term frequency (TF) by the inverse document frequency (IDF). 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. In TF-IDF, the vector 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. In the experiment, scikit-learn (Pedregosa et al., 2011) generates the vectors in TF-IDF.

LDA is a topical and generative statistical model for documents. Each document is a mixture of potential topics based on the concept that each topic can be characterized by its word distribution. The LDA model is built from all the documents. In the experiment, the number of topics was set to ten. The vector represents the per-document topic distribution analyzed by the LDA model. The LDA-based document topic analysis uses gensim (Řehůřek and Sojka, 2010).

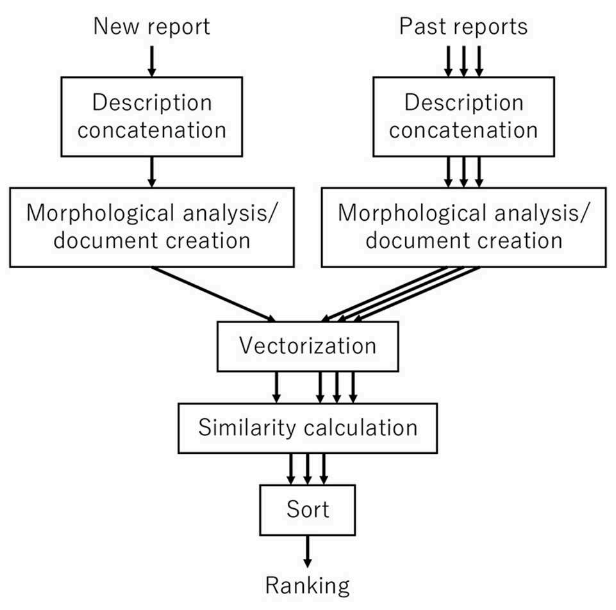
New report
Past reports
Description concatenation
Description concatenation
Morphological analysis/ document creation
Morphological analysis/ document creation
Vectorization
Similarity calculation
Sort
Ranking

FIGURE 9
Steps to detect duplicate reports in the baseline systems.

TABLE 2 Dataset used in the experiment.

	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

4.2. Dataset

Our system is functional and was applied to detect duplicate bug reports for software product X developed and maintained by the NTT Corporation. The NTT Corporation has maintained product X since 2004 using the target maintenance method. Hence, the dataset in the experiment is the inquiry and failure reports for software product X. The dataset is curated so that each report has between 0 and 2 duplicate reports (Table 2). The experiment focuses on the inquiry reports issued in 2014 and their corresponding failure reports. This year was selected because the product was mature by this time, and the likelihood of a corresponding failure report is high.

4.3. Evaluation

4.3.1. Proposed system

The performance evaluation has five steps:

- Step 1: Divide all reports into five groups to implement the 5-fold cross-validation to avoid overfitting by calculating the average of the results. As the number of N-fold cross-validation, we chose N = 5 for our experiment since often the 5-fold cross-validation has been adopted for handling bug reports such as their severity

4 https://taku910.github.io/mecab/

TABLE 3 Number of triplets in a report triplet set and the mean evaluation score.

	Title model	Content model
Number of triplets in a report triplet set	1,200–1,500	1,100–1,300
Mean evaluation score	0.985	0.987

prediction (Chaturvedi and Singh, 2012; Sharma et al., 2015). In the future, we plan to conduct more experiments using different values (such as 2- and 10-fold) to confirm their impacts on the result.

- Step 2: Designate one group as the test data. The remaining are training data.
- Step 3: Train the title and content models using the training data. Use the report triplet set to evaluate models, and then store the title vectors and content vectors of all reports as previous reports.
- Step 4: For each report in the test data, remove the title vector and content vector. Use each report as input.
- Step 5: The system outputs a list of similarities in descending order.

These steps are repeated for every element and report until all data is used as the test step. Then the performance is assessed using an evaluation metric. The performance value is the mean score of the evaluation metric.

The evaluation metric of a title or content model is the proportion of triplets where the distance between the anchor and positive vectors is smaller than that between the anchor and negative vectors.

The evaluator in SBERT is used to evaluate a model's performance. Five sets of models are prepared because the system is built five times. Table 3 shows the range of the number of triplets in the report triplet set used for training and evaluating as well as the mean evaluation score of the five different model sets.

Furthermore, we conducted the same experiment by preparing our proposed system without fine-tuning (i.e., without the training step 3) to clarify the effectiveness of fine-tuning using reports.

In the experiment, we identified and employed the following good correspondence combination and the calculation method. For the title element correspondence combination, we identified and adopted 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. For the content item correspondence combination, we identified and 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 identified and utilized report similarity = max (title similarity, content similarity).

4.3.2. Baselines

The baselines are evaluated using three steps:

- Step 1: Select an inquiry report with at least one duplicate report in the dataset as a new report. Remove the content of the answer and input the report into the system.
- Step 2: Input the other reports as previous reports.
- Step 3: The system outputs a list of similarities in descending order.

These steps are repeated for all reports with at least one duplicate. Then the performance is assessed using an evaluation metric.

4.3.3. Evaluation metric

A conversation with a maintenance engineer revealed that if they were to adopt our proposed system to detect duplicate reports, they would first review the results with the highest rankings. Consequently, we selected a metric that reflects the ranking of the duplicate reports.

The mean average precision (MAP) evaluates the output ranking performance by a search system. First, the average precision (AP) of the outputs from multiple search queries is ranked. Then MAP is determined using the mean value of multiple evaluation scores of the outputs. AP is expressed using the precision of the top r ranks (Precision@r) as:

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

where n is the number of duplicate reports of the input report in our research.

$$I(r) = \begin{cases} 1 & \text{(the } r\text{th ranked element is a match (duplicate))} \\ 0 & \text{(otherwise)} \end{cases} \tag{3}$$

4.4. Results

The MAP score of our proposed system with fine-tuning is 0.829 compared to a score of 0.751 using the baseline with TD-IDF and 0.308 for the baseline with LDA (Figure 10). Hence, our proposed system with fine-tuning outperforms the baselines. Furthermore, our proposed system with fine-tuning outperformed our system without fine-tuning.

4.4.1. RQ1. Is it possible to identify duplicate reports?

The results show that it is possible to detect duplicate bug reports automatically. Not only the proposed system but also the baselines can detect duplicate reports. The results suggest that it is possible to design an automated bug detection system for the maintenance method targeted in this study. The results also show that the training and fine-tuning process influence the ability to identify duplicate reports.

4.4.2. RQ2. Does the proposed system outperform the baselines?

The proposed system with fine-tuning outperformed the baselines. Moreover, fine-tuning positively impacted the detection

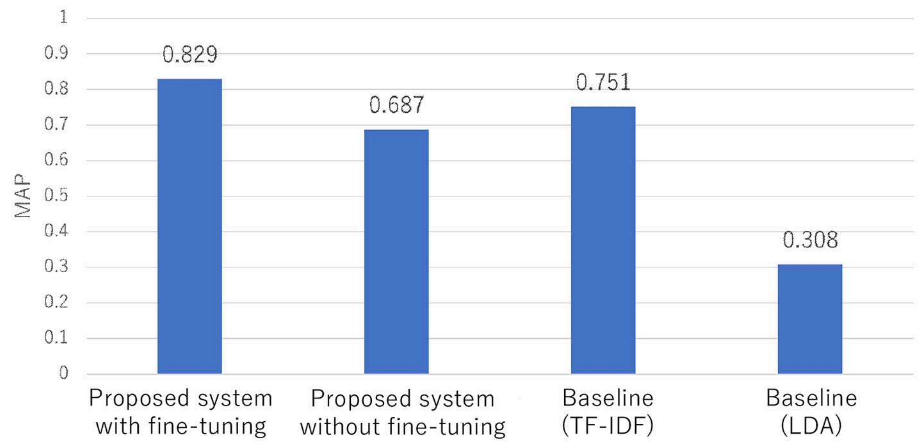

FIGURE 10
Performance evaluation scores by system.

performance by confirming that the system with fine-tuning outperformed that without fine-tuning. The proposed system processes each element in a report individually and fine-tunes SBERT with actual bug reports. This approach is more effective than the baselines, which use TF-IDF or LDA to train with all the data in a report. For the reports targeted in this research, a combination of processing separately by element and fine-tuning SBERT with reports improves the performance of duplicate bug report detection.

Our proposed system has two main advantages. First, fine-tuning with SBERT allows the characteristics of the target reports to be incorporated. Second, training the title model, independent of the content model, allows more appropriate vectors to be generated. The title model focuses on expressions. Although titles have limited content, they provide succinct descriptions of the problem. In contrast, the content model covers a wide range of problem descriptions. Because content contains more details (e.g., the test environment, setting, failure description, operations at the time of the incident, and possibly the root cause), critical expressions to identify duplicate reports may appear in the elements but not the title. By combining title and content similarities in various ways, such as weighted values and the max and min of two similarities, and selecting the best one, our system can effectively address the nature of each different domain dataset to detect duplicate reports. For example, we identified and utilized report similarity = max(title similarity, content similarity) in the experiment.

As shown in Table 3, the constructed SBERT effectively generates vectors. Not only does using two models identify differences in wording and explanations, but this approach can also detect abnormalities in product X because training uses long-distance dependencies.

4.5. Threats to validity

The data was skewed to ensure that the experiment contained reports with duplicates. A real-world maintenance project likely has fewer duplicate reports. This is a threat to external validity. In the future, this should be assessed by evaluating the performance using real-world datasets with more reports and fewer duplicates.

Another threat to external validity is that the dataset was for a specific product. The proposed system was designed based on the characteristics of the report descriptions in the dataset, but it may be effective for other products. In the future, the universality of the proposed system should be validated using datasets of reports for other products that employ the same maintenance method.

The format of the bug report is another threat. It is difficult to directly apply the proposed system because the format varies by maintenance method. In the future, a generalized detection technique, which is independent of the maintenance method, should be developed and evaluated using bug reports obtained from different maintenance methods.

5. Related works

Previous studies have analyzed bug reports and detected duplicates. The similarities and differences between our proposed system and existing works are summarized below. Furthermore, we also summarize recent works on fundamental sentence embedding techniques.

5.1. Duplicate bug report detection

Our system detects duplicate bug reports using vector similarities. Similar systems have been used in other research. One study developed a technique to output the duplication probability by consolidating vectors. They generated fragmented elements of bug reports (product information and priority) and the sentence parts (Rodrigues et al., 2020). In the sentence parts, vectors were generated by categorizing words into two groups: belonging to the summary or belonging to a description. The approach of using different vectors to compare reports drastically differs from our proposed system.

Another study proposed a technique to encode structured information, short descriptions, and long descriptions (Deshmukh et al., 2017). Their technique employed a different model for each element, which was subsequently joined to generate a vector for a report. Duplicates were subsequently determined from similarities.

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. Their study generated vectors using bi-LSTM for short descriptions and CNN for long ones. In contrast, our proposed system uses an attentive neural network to train using long-distance dependencies to generate vectors. As such, our system can generate more context-reflective vectors.

5.2. Bug report techniques using BERT

Other studies have applied BERT to software development and maintenance documents. For example, one technique predicted the time length to fix the bug by inputting texts from bug reports into BERT (Ardimento and Mele, 2020). Another developed a technique to identify duplicate bug reports by Open Source Software that caused bugs using report texts and BERT (Hirakawa et al., 2020). Unlike these studies, our system uses BERT to detect duplicate bug reports.

Furthermore, several studies (Rocha and da Costa Carvalho, 2021; Messaoud et al., 2022) have adopted BERT for encoding textual features of bug titles and descriptions to detect duplicate bug reports. Although our system also adopts BERT to detect duplicates, our system examines various different correspondence combinations at the element level and various different calculation methods. In contrast, these studies haven't explored such element-level combinations. In the future, these models should be compared with our system in detail by targeting the same datasets.

5.3. Sentence embedding

Sentence embedding techniques for extracting a numerical representation of a sentence to encapsulate its meanings have been well-studied in these years. Sentence embedding methods can be categorized into non-parameterized and parameterized models (Wang and Kuo, 2020). Furthermore, some methods combine the advantage of both parameterized and non-parameterized methods, such as SBERT-WK, which is computed by subspace analysis of the manifold learned by the parameterized BERT-based models (Wang and Kuo, 2020).

Non-Parameterized models are usually based on pre-trained word embedding methods and aggregation of each word's embedding into a sentence. Weighted averaging techniques such as TF-IDF and Smooth Inverse Frequency (SIF) (Arora et al., 2017) based on the random walk to model the sentence generation process fall into this group. From non-parameterized models, we selected the simplest TF-IDF as one of the comparative baselines.

Parameterized models are more complex, and they usually perform better than non-parameterized models (Wang and Kuo, 2020). USE based on multiple objectives and transformer architecture (Cer et al., 2018), InferSent employing bi-directional LSTM (BiLSTM) with supervised training (Conneau et al., 2017), and the SBERT method as one of the state-of-the-art sentence embedding models by training the Siamese network over BERT fall into this group. BERT, SBERT, and LSTM are often employed to achieve complicated NLP tasks such as natural language understanding (NLU). For example, the NLU framework for argumentative dialogue has used both BiLSTM and a pre-trained BERT model for recognizing the user intent and identifying system arguments (Abro et al., 2022). A pre-trained BERT model, CNN, and RNN have been adopted together in a multi-task model for multi-turn intent determination and slot-filling tasks (Abro et al., 2020). From parameterized models, we selected the SBERT method for the core of our system.

In the future, we plan to compare our system with approaches based on other non-parameterized and parameterized sentence embedding models as different baselines to confirm the model performance.

6. Conclusion and future work

Detecting duplicate bug reports can reduce the burden on maintenance engineers. Herein a duplicate bug detection based on deep learning using SBERT is reported. In our system, SBERT generates vectors for each element in a bug report, which are subsequently used to determine the similarities between reports and output rankings in descending order.

The proposed system was designed for a specific maintenance method. Its performance was assessed experimentally by comparing it with baseline systems. Unlike the proposed system, which generates vectors by each element, the baseline systems generate vectors using all elements in the report. And the proposed system outperformed the baseline systems. The results suggest that a multiple-step process in which the elements in a report are separately processed and fine-tuned with SBERT can effectively detect bug reports.

To enhance the effectiveness of our system, achieve duplicate bug report detection in various actual development projects, and explore new applications to achieve effective and practical handling of bug reports, there are multiple directions for future research roughly classified into five types: an examination of our system's design and settings in detail, evaluation of universality resulting in generalization of our system, comparison with other existing state-of-the-art approaches and identification of potential improvements, incorporation other features including the structure of target software, and exploring applications to other bug report analysis tasks such as identifying typical report discussion patterns.

The first is to examine the individual degree of contribution by item-wise processing and fine-tuning SBERT with the report texts on the performance. We plan to evaluate our proposed system by comparing it 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. In relation to that, it is also necessary to examine better settings of our system in terms of the model's hyperparameters. To clarify that, we plan to conduct more experiments using different thresholds and hyperparameters to confirm their impacts on the result.

The second is to evaluate the robustness and universality of our system under various environments with various maintenance methods and report formats. To consider them, we plan to conduct more N-fold cross-validation experiments using different N values to confirm their impact on the result. Furthermore, it is necessary to validate the universality of our system by evaluating the performance using other real-world datasets with more reports. In relation to that, we plan to generalize our system to be independent of the elementization of a report format. A generalized detection technique, independent of the maintenance method, including report formats, should be developed and evaluated using bug reports obtained from different maintenance methods.

The third is to compare our system with approaches based on other existing state-of-the-art models and identify potential improvements and extensions of the system. We plan to compare our method with approaches based on other non-parameterized and parameterized sentence embedding models, including USE and InferSent, as different baselines to confirm the model performance.

The fourth is to extend our system to incorporate other features, particularly the structure of target software 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.

Finally, not only will these endeavors realize our ultimate goal, but they should contribute to more effective analysis of bug reports, including identifying typical bug report discussion patterns (Noyori et al., 2021a), deep-learning of bug fixing time predictions (Noyori et al., 2021b), and severity predictions (Liu et al., 2021).

Data availability statement

The original contributions presented in the study are included in the article/supplementary material, further inquiries can be directed to the corresponding author.

Author contributions

HI has handled conceptualization and methodology. All authors have contributed to the literature review analysis, read, and agreed to the published version of the manuscript.

Acknowledgments

We thank Yuta Koizumi, Zhenyu Xu, Akira Kanamaru, Tatsuka Muramoto, and Naohiko Tsuda for their support. We also would like to thank the reviewers for their insightful comments and suggestions.

Conflict of interest

TN, SO, and SS were employed by NTT Corporation.

The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

References

Abro, W. A., Aicher, A., Rach, N., Ultes, S., Minker, W., and Qi, G. (2022). Natural language understanding for argumentative dialogue systems in the opinion building domain. *Knowl. Based Syst.* 242, 108318. doi: 10.1016/j.knosys.2022.108318

Abro, W. A., Qi, G., Ali, Z., Feng, Y., and Aamir, M. (2020). Multi-turn intent determination and slot filling with neural networks and regular expressions. *Knowl. Based Syst.* 208, 106428. doi: 10.1016/j.knosys.2020.106428

Akilan, T., Shah, D., Patel, N., and Mehta, R. (2020). "Fast detection of duplicate bug reports using lda-based topic modeling and classification," in *2020 IEEE International Conference on Systems, Man, and Cybernetics, SMC 2020, Toronto, ON, Canada, October 11–14, 2020* (Toronto, ON: IEEE), 1622–1629.

Ardimento, P., and Mele, C. (2020). "Using BERT to predict bug-fixing time," in *2020 IEEE Conference on Evolving and Adaptive Intelligent Systems, EAIS 2020, Bari, Italy, May 27–29, 2020* (Bari: IEEE), 1–7.

Arora, S., Liang, Y., and Ma, T. (2017). "A simple but tough-to-beat baseline for sentence embeddings," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24–26, 2017, Conference Track Proceedings* (Toulon: OpenReview.net).

Babic, K., Martincic-Ipsic, S., and Mestrovic, A. (2020). Survey of neural text representation models. *Information* 11, 511. doi: 10.3390/info11110511

Blei, D. M., Ng, A. Y., and Jordan, M. I. (2003). Latent dirichlet allocation. *J. Mach. Learn. Res.* 3, 993–1022. Microtome Publishing.

Bromley, J., Guyon, I., LeCun, Y., Säckinger, E., and Shah, R. (1993). "Signature verification using a siamese time delay neural network," in *Advances in Neural Information Processing Systems 6*, [*7th NIPS Conference, Denver, Colorado, USA, 1993*], eds J. D. Cowan, G. Tesauro, and J. Alspector (Denver, CO: Morgan Kaufmann), 737–744.

Cer, D., Yang, Y., Kong, S., Hua, N., Limtiaco, N., John, R. S., et al. (2018). "Universal sentence encoder for english," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, EMNLP 2018: System Demonstrations, Brussels, Belgium, October 31-November 4, 2018*, eds E. Blanco and W. Lu (Brussels: Association for Computational Linguistics), 169–174.

Chaturvedi, K. K., and Singh, V. B. (2012). "Determining bug severity using machine learning techniques," in *2012 CSI Sixth International Conference on Software Engineering (CONSEG)* (Piscataway: IEEE), 1–6. doi: 10.1109/CONSEG.2012.6349519

Conneau, A., Kiela, D., Schwenk, H., Barrault, L., and Bordes, A. (2017). "Supervised learning of universal sentence representations from natural language inference data," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9–11, 2017*, eds M. Palmer, R. Hwa, and S. Riedel (Copenhagen: Association for Computational Linguistics), 670–680.

Dai, A. M., Olah, C., and Le, Q. V. (2015). Document embedding with paragraph vectors. *CoRR*, abs/1507.07998. doi: 10.48550/arXiv.1507.07998

Deshmukh, J., Annervaz, K. M., Podder, S., Sengupta, S., and Dubash, N. (2017). "Towards accurate duplicate bug retrieval using deep learning techniques," in *2017 IEEE International Conference on Software Maintenance and Evolution, ICSME 2017* (Shanghai: IEEE Computer Society), 115–124.

Devlin, J., Chang, M., Lee, K., and Toutanova, K. (2019). "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, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, eds J. Burstein, C. Doran, and T. Solorio (Minneapolis, MN: Association for Computational Linguistics), 4171–4186.

Ghosh, K., Pawar, S., Palshikar, G. K., Bhattacharyya, P., and Varma, V. (2020). "Retrieval of prior court cases using witness testimonies," in *Legal Knowledge and Information Systems - JURIX 2020: The Thirty-third Annual Conference, Brno, Czechia, December 9-11, 2020, volume 334 of Frontiers in Artificial Intelligence and Applications*, eds S. Villata, J. Harasta, and P. Kremen (Brno: IOS Press), 43–51.

Hirakawa, R., Tominaga, K., and Nakatoh, Y. (2020). "Study on automatic defect report classification system with self attention visualization," in *2020 IEEE International Conference on Consumer Electronics (ICCE)* (Las Vegas, NV: IEEE), 1–2.

Isotani, H., Washizaki, H., Fukazawa, Y., Nomoto, T., Ouji, S., and Saito, S. (2021). "Duplicate bug report detection by using sentence embedding and fine-tuning," in *IEEE International Conference on Software Maintenance and Evolution, ICSME 2021* (Luxembourg: IEEE), 535–544.

Kanakogi, K., Washizaki, H., Fukazawa, Y., Ogata, S., Okubo, T., Kato, T., et al. (2022). Comparative evaluation of nlp-based approaches for linking capec attack patterns from cve vulnerability information. *Appl. Sci.* 12, 3400. doi: 10.3390/app12073400

Kukkar, A., Mohana, R., Kumar, Y., Nayyar, A., Bilal, M., and Kwak, K. (2020). Duplicate bug report detection and classification system based on deep learning technique. *IEEE Access* 8, 200749–200763. doi: 10.1109/ACCESS.2020.3033045

Li, T. J., Chen, J., Xia, H., Mitchell, T. M., and Myers, B. A. (2020). "Multi-modal repairs of conversational breakdowns in task-oriented dialogs," in *UIST '20: The 33rd Annual ACM Symposium on User Interface Software and Technology*, eds S. T. Iqbal, K. E. MacLean, F. Chevalier, and S. Mueller (Virtual Event, USA: ACM), 1094–1107.

Liu, Q., Washizaki, H., and Fukazawa, Y. (2021). "Adversarial multi-task learning-based bug fixing time and severity prediction," in *10th IEEE Global Conference on Consumer Electronics, GCCE 2021* (Kyoto: IEEE), 185–186.

Messaoud, M. B., Miladi, A., Jenhani, I., Mkaouer, M. W., and Ghadhab, L. (2022). Duplicate bug report detection using an attention-based neural language model. *IEEE Trans. Reliabil.* 1–13. doi: 10.1109/TR.2022.3193645

Noyori, Y., Washizaki, H., Fukazawa, Y., Kanuka, H., Ooshima, K., Nojiri, S., et al. (2021a). What are the features of good discussions for shortening bug fixing time? *IEICE Trans. Inf. Syst.* 104-D, 106–116. doi: 10.1587/transinf.2020MPP0007

Noyori, Y., Washizaki, H., Fukazawa, Y., Ooshima, K., Kanuka, H., Nojiri, S., et al. (2021b). "Extracting features related to bug fixing time of bug reports by deep learning and gradient-based visualization," in *Proceedings of the IEEE International Conference on Artificial Intelligence and Computer Applications, ICAICA* (Dalian: IEEE Computer Society), 402–407.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., et al. (2011). Scikit-learn: machine learning in python. *J. Mach. Learn. Res.* 12, 2825–2830.

Řehůřek, R., and Sojka, P. (2010). "Software Framework for Topic Modelling with Large Corpora," in *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks* (Malta: University of Malta), 45–50.

Reimers, N., and Gurevych, I. (2019). "Sentence-bert: sentence embeddings using siamese bert-networks," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019* eds K. Inui, J. Jiang, V. Ng, and X. Wan (Hong Kong: Association for Computational Linguistics), 3980–3990.

Rocha, T. M., and da Costa Carvalho, A. L. (2021). Siameseqat: A semantic context-based duplicate bug report detection using replicated cluster information. *IEEE Access* 9, 44610–44630. doi: 10.1109/ACCESS.2021.3066283

Rodrigues, I. M., Aloise, D., Fernandes, E. R., and Dagenais, M. R. (2020). "A soft alignment model for bug deduplication," in *MSR '20: 17th International Conference on Mining Software Repositories*, eds S. Kim, G. Gousios, S. Nadi, and J. Hejderup (Seoul: ACM), 43–53.

Schroff, F., Kalenichenko, D., and Philbin, J. (2015). "Facenet: a unified embedding for face recognition and clustering," in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015* (Boston, MA: IEEE Computer Society), 815–823.

Sehra, S. S., Abdou, T., Basar, A., and Sehra, S. K. (2020). "Amalgamated models for detecting duplicate bug reports," in *Advances in Artificial Intelligence-33rd Canadian Conference on Artificial Intelligence, Canadian AI 2020, Ottawa, ON, Canada, May 13–15, 2020, Proceedings, volume 12109 of Lecture Notes in Computer Science*, eds C. Goutte and X. Zhu (Ottawa, ON: Springer), 470–482.

Shahmirzadi, O., Lugowski, A., and Younge, K. (2019). "Text similarity in vector space models: a comparative study," in *18th IEEE International Conference On Machine Learning And Applications, ICMLA 2019*, eds M. A. Wani, T. M. Khoshgoftaar, D. Wang, H. Wang, and N. Seliya (Boca Raton, FL: IEEE), 659–666.

Sharma, G., Sharma, S., and Gujral, S. (2015). A novel way of assessing software bug severity using dictionary of critical terms. *Procedia Comput. Sci.* 70, 632–639. doi: 10.1016/j.procs.2015.10.059

Subramanian, S., Trischler, A., Bengio, Y., and Pal, C. J. (2018). "Learning general purpose distributed sentence representations via large scale multi-task learning," in *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30-May 3, 2018* (Vancouver, BC: Conference Track Proceedings. OpenReview.net).

Uno, K. (2020). "Natural language processing for beginners part 9: verification of similar sentence retrieval using sentence bert." Available online at: https://www.ogis-ri.co.jp/otc/hiroba/technical/similar-document-search/part9.html

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., et al. (2017). "Attention is all you need," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017*, eds I. Guyon, von U. Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett (Long Beach, CA), 5998–6008.

Wang, B., and Kuo, C. J. (2020). SBERT-WK: a sentence embedding method by dissecting bert-based word models. *IEEE ACM Trans. Audio Speech Lang. Process.* 28, 2146–2157. doi: 10.1109/TASLP.2020.3008390

Xiao, G., Du, X., Sui, Y., and Yue, T. (2020). "HINDBR: heterogeneous information network based duplicate bug report prediction," in *31st IEEE International Symposium on Software Reliability Engineering, ISSRE 2020*, eds M. Vieira, H. Madeira, N. Antunes, and Z. Zheng (Coimbra: IEEE), 195–206.

Yoshikawa, Y., Shigeto, Y., and Takeuchi, A. (2017). "STAIR captions: constructing a large-scale japanese image caption dataset," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30-August 4, Volume 2: Short Papers*, eds R. Barzilay and M. Kan (Vancouver, BC: Association for Computational Linguistics), 417–421.

Zhang, M., Li, Z., Fu, G., and Zhang, M. (2021). Dependency-based syntax-aware word representations. *Artif. Intell.* 292, 103427. doi: 10.1016/j.artint.2020.103427