Received February 15, 2021, accepted March 5, 2021, date of publication March 17, 2021, date of current version March 26, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3066283

SiameseQAT: A Semantic Context-Based Duplicate Bug Report Detection Using Replicated Cluster Information

THIAGO MARQUES ROCHA[1] AND ANDRÉ LUIZ DA COSTA CARVALHO[2]
[1]Sidia R&D Institute, Manaus 69055-035, Brazil
[2]Institute of Computing—IComp, Federal University of Amazonas, Manaus 69067-005, Brazil

Corresponding author: Thiago Marques Rocha (thiago@icomp.ufam.edu.br) and André Luiz Da Costa Carvalho (andre@icomp.ufam.edu.br)

This work was supported in part by the Samsung Eletrônica da Amazônia Ltd., through the Auspice of the Informatics Law, under Grant 8.387/91. In addition, the author would like to thank (i) the third-part Icomp/Federal University of Amazonas (UFAM), and (ii) Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

ABSTRACT In large-scale software development environments, defect reports are maintained through bug tracking systems (BTS) and analyzed by domain experts. Different users may create bug reports in a non-standard manner and may report a particular problem using a particular set of words due to stylistic choices and writing patterns. Therefore, the same defect can be reported with very different descriptions, generating non-trivial duplicates. To avoid redundant work for the development team, an expert needs to look at all new reports while trying to label possible duplicates. However, this approach is neither trivial nor scalable and directly impacts bug fix correction time. Recent efforts to find duplicate bug reports tend to focus on deep neural approaches that consider hybrid representations of bug reports, using both structured and unstructured information. Unfortunately, these approaches ignore that a single bug can have multiple previously identified duplicates and, therefore, multiple textual descriptions, titles, and categorical information. In this work, we propose SiameseQAT, a duplicate bug report detection method that considers information on individual bugs as well as information extracted from bug clusters. The SiameseQAT combines context and semantic learning on structured and unstructured features and corpus topic extraction-based features, with a novel loss function called *Quintet Loss*, which considers the centroid of duplicate clusters and their contextual information. We validated our approach on the well-known open-source software repositories Eclipse, NetBeans, and Open Office, comprised of more than 500 thousand bug reports. We evaluated both the retrieval and classification of duplicates, reporting a Recall@25 mean of 85% for retrieval and 84% AUROC for classification tasks, results that were significantly superior to previous works.

INDEX TERMS Duplicate bug report, deep learning, deep neural networks, semantic context-based, Siamese network, loss function, quintet, triplet, attention mechanism, BERT, MLP, LDA, topic modeling.

I. INTRODUCTION

Bug Tracking Systems (BTS) are tools used to coordinate and manage bug detection and fixing in large software development environments. Bugs may be reported by different sources, e.g., developers, testers, and users in general, with inconsistent levels of detail [1]. Typically, the bug triage process can be divided into three phases [2]: (i) understanding, (ii) selection, and (iii) correction. Understanding (i) and selecting (ii) a bug report are not trivial tasks, since different users may write reports using their own words, causing arduous work for an expert *triager*, who is liable to understand and separate these documents. Correction (iii) is the last phase when the development team fixes the reported bug. A *triager* is a developer who should be able to understand all reports published in a day, while screening for possible duplicated reports, avoiding rework on redundant tasks [3], [4].

This triage work is not easily scalable in a situation of rising reports pending analysis. Thus, solutions that automate triage processes can directly impact software management costs, aiding decision-making and avoiding development rework during the bug fix. Many researchers have explored automatic approaches for selecting duplicate reports, which can be roughly divided into information retrieval (IR) and machine learning/deep learning (DL) based methods. IR-based methods usually deploy variations of Vector Space

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

Models (VSM) [5]–[9] to represent bug reports based on features like word frequency or topic distribution, extracting syntax and context from the reports. On the other hand, the most recent machine learning-based methods tend to center on deep neural networks [10]–[17]. These models extract automatic latent features based on embedding representations from structured and unstructured content. This manifests a different way of extracting lingering patterns from bug reports, capturing semantic and syntactic context about the content of the documents.

While those previous works achieved satisfactory results, in this work, we believe that the duplicate bug detection task results found can be further improved by exploring three new approaches to enhance the bug report representation through embedding vectors: First, attention-based textual embeddings [18]. Second, bug report corpus context for embeddings using topic modeling, and third, using duplicate cluster information to improve bug representation:

Attention-based textual embedding. We propose to improve text representation of bug reports, where we use attention mechanisms instead of more traditional models used in previous works [10]–[17].

In previous works, the context of words is not weighted individually, resulting in representations that describe the sentences of the documents and not also the word context. Hence, these previous approaches dismiss specific challenges of the bug report documents (e.g., semantic variation and natural and non-natural language content, with stack traces, logs, and source code). Consequently, an attention-based network should understand the context of sequences via weights in the vocabulary, smoothing the detection of non-natural language, and context changes in the text. In the literature, the Bidirectional Encoder Representations from Transformers (BERT) is a language model considered a state-of-art for textual representation, a model that enables learning based on attention mechanisms. Thus, we pretend to use it to generate embedding of textual features, as [19] have employed in the context of commit messages in software engineering tasks. This way, we will deliver rich textual embedding for the bug report documents.

Textual context using topic modeling. We enhance our bug text representation by extracting latent topics from the report corpus and adding the extracted topic distribution to each bug report, thus adding information regarding the relation of a specific report to all others. This is done by applying topic modeling methods [20] to capture the shared topics between the reports, in hopes that it can increase the detection accuracy, similarly to what was demonstrated in [6].

Replicated cluster information using centroid. In order to improve bug representation, we introduce a new loss function that also considers information pertaining to duplicated bug report clusters. Our proposed function, named Quintet Loss, uses an optimization strategy that extends the use of the Triplet Loss function [21], including duplicate clusters information in its learning process, by also bringing the anchor instance closer to its positive cluster centroid while distancing it from a negative duplicate cluster centroid. In the literature, a Triplet Loss function can boost the learning of features on bug reports, and according to [21], this loss is used in problems to detect the similarity between objects. In this way, we first investigated the use of duplicates group information to train this set of triplets together with their centroid groups. On the other hand, Ge *et al.* [22], and Wu *et al.* [23] explain that Triplet loss treats all triplets as equally important, using of the same weights used for all triplets. This has a negative impact if the training set includes triplets too much deviant from the global data distribution due to artifacts in sampling. Therefore, we assume that using the information on whole clusters of duplicates may help the model to better organize the duplicates in the latent space, reducing the equally important triplet effect, minimizing duplicate set overlaps, and improving report discrimination tasks.

In this context, as a way to automate the selection of duplicates and reduce the time spent on the triage processes, we propose SiameseQAT, a Siamese approach to detect duplicate bug reports, using attention-based, topic-based context representation embeddings and the *Quintet Loss* function.

In summary, we present the significant research contributions of SiameseQAT:

(i) We propose a novel clustering loss function, Quintet Loss, which uses the centroids of duplicate clusters while generating its bug report representation to improve replicas' discrimination. The source code[1] is available for the research community enabling its use in any clustering problem, just as we employed in duplicated bug detection.

(ii) The deployment of semantic context-based learning behind topic modeling and BERT language modeling in tandem with a Siamese architecture to generate contextual relations in structured and unstructured information from bug reports.

(iii) We evaluated our approach on large popular open-source projects, comparing with two state-of-the-art [10], [12]. The experimental results show that we outperform the two works for duplicate bug report detection on tasks of detection, classification, and retrieval.

Our paper is organized as follows. In Section II, we illustrate the challenges in duplicate bug reports. Section III gives an overview of the related literature. In Section IV-A, we cover the minimal prerequisites to understand the key concepts of deep learning. In Section IV, we describe the SiameseQAT approach and its architecture. Section V describes the experiments and discusses the results. Then, we conclude the paper and describe future research opportunities in Section VI.

[1] https://github.com/thiagomarquesrocha/siameseQAT

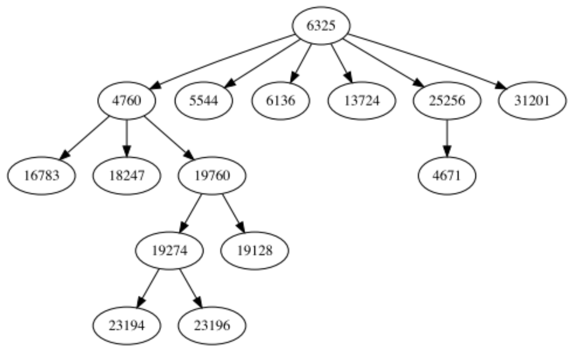

FIGURE 1. An example of duplicate reports extracted from [24]. Report #6325 is the master and all other are duplicated reports. Every report X is referenced by Y, denoted as $Y->X$.

II. MOTIVATION

Bug reports are documents created by bug tracker systems like Bugzilla[2] and Atlassian,[3] tools that maintain many reports describing software defects. For each bug report, we have structured and unstructured information filled by different users, such as developers, testers, or other users [1]. *Structured information is* all numeric or categorical attributes present in a bug report such as *product, component, priority, bug severity, resolution, bug status, and version*, attributes describing the software or the defect. *Unstructured information is* all attributes such as *short description* and *description* which contain textual descriptions about the defect.

Each duplicate bug report receives a manual label given by a specialist referencing the duplicate ID from the first bug report published related to the same problem [2]. *A duplicate bug report* means that two bug reports are describing the same defect using the same or different vocabulary. Thus, if a bug report references an older duplicate, the relationship between them could be visualized as a tree with the first report found (the root node) known as the master report. Which represents the set of duplicates (its child nodes), as shown in Figure 1.

To detect duplicate bug reports, an algorithm based on deep models need to determine bug embeddings that describe the duplicate content and the similarity between the documents. However, as our examples in Table 1 show, bug reports have five characteristics: *project uniqueness*; *natural language*; *non-natural language*; *semantic variation*; and *imbalanced data*. These characteristics are critical issues in duplicate bug reports detection and are detailed below:

1) *Project uniqueness*: Users of different projects often have different conventions when reporting software defects, using project domain keywords such as *word*, *doc*, *eclipse*, or *java*.

[2]https://www.bugzilla.org/
[3]https://br.atlassian.com/

2) *Natural language*: Bug reports are usually described using natural language. However, there is no guarantee that non-natural text may be present in the description.
3) *Non-natural language*: Moreover, it is not unusual for developers and testers usually attach stack traces, logs, steps to reproduce the problem, and other evidence to help understand the software defect. This data is pasted mixed with natural language.
4) *Semantic variation*. The same semantic concept can be described in many domain-related forms, such as *OO* or *OOo*, that means *Open Office*, or *NullPointerException* and *NP*, which means the null reference on runtime, and *FS* word that means *File System*, among many other examples.
5) *Imbalanced data*. The ratio of duplicate bug report pairs in a project can be low compared to non-duplicate bug reports, as Table 3 displays. This characterizes a classic problem of imbalanced data for classification and retrieval tasks.

To address the above challenges, we propose SiameseQAT, an approach for detecting duplicate bug reports in classification and retrieval tasks. Our approach learns various bug reports latent features from structured and unstructured inputs through embedding representations, thus removing the need for manual feature engineering, solving challenges 1, 2, and 3. Also, it extracts semantic and context meaning from bug features through embedding vectors, covering challenge 4. To overcome the imbalanced data issue, our approach uses the Siamese strategy to train always on balanced inputs, giving a random bug report, its positive, negative, and their centroids at the same time.

III. RELATED WORK

Duplicate bug report detection approaches can use structured and unstructured information while also being divided between information retrieval (IR) solutions and machine learning/deep learning (DL) techniques. Unstructured information is textual features, usually extracted from fields like short description and description to generate a text document. Structured information consists of categorical fields such as product, component, priority, bug severity, resolution, bug status, and version. Hybrid-structured information methods use textual features and categories together, and in some cases, include execution information like stack trace or logs.

A. INFORMATION RETRIEVAL FOR DUPLICATE REPORT DETECTION

Earlier work on duplicate bug report detection was introduced by [3], who used a document feature vector based on a Bag of Words (BOW) approach to describe bug reports and applied cosine similarity between report vectors to find candidate duplicates through its content similarity. It achieved an accuracy of up to 40% in detecting duplicates in a private dataset from Sony Ericsson Mobile Communications. The main disadvantage of applying BOW in this method is to

TABLE 1. Examples of duplicate bug reports from three open-source software projects.

Project	Product	Duplicate descriptions
Eclipse	WTP Common Tools	**Title** : Deleting a validation-disabled resource can lead to NPE. **Description** : If a resource is disabled from validation and then removed, a NPE will be triggered the next time a resource is enabled or disabled. It most likely occurs because IContainer#findMember() can return a null, and that return result is stored in the disabled list regardless in org.eclipse.wst.validation.internal.DisabledResourceManager#load(IProject) line 91...
		Title : NPE in DisabledResourceManager.save(). **Description** : In the context of 424340 we faced a NPE in DisabledResourceManager.save() java.lang.NullPointerException at org.eclipse.wst.validation.internal.DisabledResourceManager.save(DisabledResourceManager.java:67) at...
NetBeans	Platform	**Title** : Filesystem's state could be changed to read-only when I have file lock. **Description** : How to reproduce: Let have a read-write FS. Open some file from it into the editor. Edit file but do not save it. Set FS state to read-only. ERROR - FS state could be changed and file could be saved although FS is read- only.
		Title : Saving (CTRL-S or File/Save) a file on read-only filesystem causes uncaught IOException. Save All action correctly reports that it can't save file. **Description** : java.io.IOException: File Outer/Inner/Pokus.java is readonly. at org.openide.text.EditorSupport$5.run(EditorSupport.java:422) at org.openide.filesystems.FileSystem.runAtomicAction(FileSystem.java:357) at org.openide.text.EditorSupport.saveDocument(EditorSupport.java:412) at com.netbeans.developer.modules.loaders.java.JavaEditor.saveDocument(JavaEditor.java:294)...
Open Office	Writer	**Title** : File saved as M.Word 97/2000/XP (.doc) corrupted. **Description**: The file came to me as a one page .doc form (NOT .dot) and included 3 tables. After completing I saved it back as Word 97/200/XP. The resulting file does not open in Word 2000 and when opened in OO Writer the second and third tables are missing and replaced by a lot of space forcing the final text on to the next page...
		Title : WW8: table with center alignment disappears when exported. **Description** : Hi My configuration : OOo 3.0.1 FR under Ubuntu 8.04. Step to reproduce : 1/ open new text document ; 2/ insert a table, for example 2 rows and 3 columns ; 3/ menu Table > Table properties > Tab Table 4/ choose Alignment Center and click Ok ; 5/ save the file as odt 6/ menu File > Save as and choose file format MS-Word 97/2000/XP (.doc) 7/ close the file .doc 8/ reopen the doc file with OOo 3.0.1 => the table has disappeared...

ignore the word placement context. It also only considers word frequency in its vector representation. Nevertheless, it remains a simple and computationally inexpensive method.

Sun *et al.* [25] improve the previous approach by using feature vectors based on Inverse Document Frequency (IDF) to learn the weights of all words in a bug report. Candidate duplicates are also retrieved by cosine similarity. The corpus in this paper considered fields such as short_description, description, and the concatenation of both fields. Then, a Support Vector Machine (SVM) is applied to a pair of reports to classify whether they are duplicates. Their results show accuracy levels between 50% and 70% on three datasets, Eclipse, Firefox, and Open Office.

The proposal of [26] employs both natural and non-natural texts extracted from description fields, stack traces, and logs. The idea is to calculate two similarity scores combining natural and non-natural information from bug reports, vectorizing them using TFIDF, and retrieving candidates by cosine similarity. They have reached an accuracy of 71% and 82% to detect the duplicates in Eclipse and Firefox datasets, respectively. In future works, we intend to study techniques to process non-natural text separately for duplicate retrieval.

Jalbert and Weimer [27] developed a classifier mixing bug severity, date fields, a similarity score from textual fields, and graph information with links between the bugs to discriminate duplicates. Their strategy reached 43% of accuracy in the Mozilla dataset.

Sun *et al.* [5] have proposed REP, a retrieval method based on BM25F*ext* to vectorize textual and categorical fields. This method supports long text queries and learns weights through stochastic gradient descent. BM25F was developed to address the lack of normalization and standardization in all fields of BM25. The proposal achieved from 68% to 72% of accuracy in the Eclipse, Open Office, and Mozilla datasets.

Nguyen *et al.* [6] have used the BM25F*ext* combined with Latent Dirichlet Allocation (LDA), a topic model describing the duplicates based on shared topics, calculating the similarity score between the bugs. The feature vector generated by LDA is the probability distribution of topics in a bug report, one strategy that worked well in Eclipse, Open Office, and Mozilla datasets, with an accuracy of around 80%.

Aggarwal *et al.* [28] have created a contextual method using a specific vocabulary extracted from software engineering books and technical documents. This approach produces a richer corpus, using BM25F to calculate the similarity score between bugs, detecting 90% of duplicates in Android, Open Office, Mozilla, and Eclipse.

Neysiani and Babamir [16] introduced a study focused on evaluating the best automatic bug report detection considering IR-based or ML-based solutions. The study separated some classifier as KNN and Logistic Regression (LR) combining with algorithms like Support Vector Machine (SVM), Information Gain Ratio (IGR), Chi-Square(CS), Gini Index (GI), Principal Component Analysis (PCA) to represent the documents.

It demonstrated that the ML-based approaches are more efficient than IR-based, using hybrid-structured information in all experiments. For classification tasks, these methods were able to predict the duplication up to 97% accuracy, while for retrieval tasks, 51% of replicas were correctly retrieved. The study was evaluated only in the Android dataset.

B. DEEP LEARNING FOR DUPLICATE REPORT DETECTION

More recently, deep learning-based approaches have successfully been deployed to detect duplicate bug reports.

Kukkar *et al.* [17] proposed a system based on a deep learning model for relevant feature extraction using a Convolution Neural Network (CNN) in a Siamese architecture to capture local features and examine all vocabulary in a bug report from multiple perspectives. The approach used structured and unstructured information from bug reports calculating the similarity between the reports. The words were encoded via Word2Vec with 300 dimensions. The study reported an accuracy average between 85% to 99% for classification tasks while retrieval tasks took a recall@20 rate between 79% and 94%. The dataset evaluated considered Mozilla, Eclipse, NetBeans, Gnome, Open Office, Firefox, and the combined base.

He *et al.* [15] extended the use of Convolution Neural Network (CNN) with a Dual-Channel CNN (DC-CNN), a method to combine two matrices of report features, extracting them from reports provided in a deep learning architecture. The approach uses hybrid-structured information as input, demonstrating the impact of unstructured information (e.g., product, component) to detect the replicas. To represent the document words, they have used Word2vec Continuous Bag of Words (CBOW) vectors [29] with 300 dimensions. Their study achieved in Open Office, Eclipse, Net Beans, and combined datasets an accuracy average of about 95% to discriminate when a pair of reports are replicas.

Poddar *et al.* [14] introduced a neural architecture able to detect bug report replicas and aggregate them into latent topics, a model that uses semantic notion and attention mechanism to perform the two tasks of duplicate detection and topic-based clustering. Through a neural model for multitask learning, the method proposes a custom loss function that can run both tasks of supervised duplicate classification while performing topic clustering in an unsupervised fashion. The study considered uses shared Bi-gated recurrent neural network (Bi-GRU), Word Embedding, and self-attention mechanisms as well as conditional attention in their components. Also, it used the title and product feature to detect when a pair is a replica achieving an accuracy average of 70%, 87%, 88%, and 95% on Snap S2R, Eclipse Platform, Eclipse JDT and Mozilla Firefox datasets.

Xie *et al.* [13] propose a deep framework model based on hybrid-structured information combining Convolution Neural Network (CNN) feature with domain fields. The strategy was to extract semantic and syntactic patterns from bug reports in a deep architecture to acquire various information. The words for a document are represented by word embedding (Random, Glove, Word2vec [29], [30]) and concatenated with domain fields (e.g., component, bug severity, issue time) outputting the final bug representation. To discriminate a pair of reports the model classifies in the last layer as replicate or not. This classification happened within many hidden CNN layers that extracted the latent features from the reports. The study achieved an accuracy between 82% and 94% in four datasets, Hadoop, Hdfs, MapReduce, and Spark from Jira bug tracking tool.

Budhiraja *et al.* [10], [11] introduced a pair-based deep classification model for detecting whether a pair of reports are duplicated or not named Deep Word Embedding Neural Network (DWEN). The model uses the title and description of bug reports, with words being represented by CBOW and Skip-Gram [29] vectors. Classification is done using an MLP layer and a Sigmoid output layer to discriminate whether the pair is replicate or not. It reported retrieving 77% of duplicates on retrieval task and 94% on classification in Open Office and Firefox dataset.

Deshmukh *et al.* [12] uses a deep Siamese architecture, receiving a hybrid input, with textual (title, description) and categorical (version, component, product) features. The Siamese Network is composed of three kinds of Networks: MLP, Bidirectional Long-Short Term Memory (BiLSTM), and Convolution Neural Network (CNN). The output of processing these three kinds of features is combined in an embedding vector representing the report. The network is trained using a Triplet loss function, which in each instance uses a candidate, a negative and a positive example, aiming to maximize the similarity between duplicates and minimize between non-duplicates. It reached accuracy between 50% and 81% in retrieval tasks while achieved between 72% and 82% classification accuracy in the Eclipse, NetBeans, and Open Office datasets.

C. DISCUSSION BETWEEN THE DUPLICATE REPORT DETECTION APPROACHES

For many years IR-based solutions were the state-of-art methods to detect duplicate bug reports. Unlike these approaches, which represented bug reports as bags-of-words, our approach does not discard the order of words and sentences missing the whole text meaning. Moreover, our method extracts automatic latent features that best describe the bug reports considering syntactic and semantic sense, without manual techniques to select features.

Deep Learning-based solutions became the state-of-art for detect duplicate bug reports, bringing a plethora of new techniques. SiameseQAT differs from these approaches introducing a new framework system that combines semantic context-based learning behind LDA topic modeling and BERT language, modeled in a Siamese architecture to make automatic representation for bug reports while also considering information extracted from duplicated clusters. Our proposal performs duplicate detection based on multitask learning to classify and retrieval the bug report replicas. Based on a cross-entropy loss to classify and a novel loss function to aggregate the bug replicas as clusters, we extract automatically bug features to calculate the similarity and discriminate the replicas. Furthermore, we validate the experiments in popular open-source projects that demonstrate the model effectiveness.

Aiming to easily visualize the core differences between all related works, we summarize in Table 2 describing the approach used, authors, information used, dataset, methods, task evaluated, and performance of detection. We can observe that all more recent works are based on DL solutions started a new trend of solutions to detect replicas based on deep architectures. While the IR-based solutions presented a high detection accuracy, it is unfair to directly compare all solutions based on their detection accuracy because of all distinct datasets and different heuristics to divide the pair of replicas, and previous works comparing them with DL based methods showed that the latter vastly outperformed them [10], [11], [17].

IV. SiameseQAT: A SEMANTIC CONTEXT-BASED DUPLICATE BUG REPORT DETECTION METHOD USING REPLICATED CLUSTER INFORMATION

In this section, we present our proposed method, SiameseQAT. First, we cover preliminary knowledge about DL models, the different layers used in our model, and Siamese topologies for neural networks. We then introduce our general framework for duplicate bug report detection, describing the SiameseQAT model to encode bug reports and outputting an embedding vector. Finally, we present the customized loss function proposed in this paper, Quintet Loss, as an improvement for the triplet loss function.

A. DEEP LEARNING PREREQUISITES

1) DEEP LEARNING MODELS WITH MULTILAYER PERCEPTION (MLP)

A classic structure used in neural networks is the perceptron, and the simplest deep learning models use stacks of layers of densely connected perceptrons, called Multilayer Perceptron (MLP) [31]. When processing any input X, this model extracts semantic knowledge as output, a vector Y comprised by the combination of input X with a weight W and any bias b denoted by $Y = X.W + b$. When stacking perceptron layers, the output Y of a layer is used as the input X of the next layer, creating a chain of multiple layers. With a few MLP layers, it is possible to extract semantic relationships from input data. Its output is a latent vector of size D that describes the knowledge acquired by the model and is usually applied in classification, clustering, or other tasks. MLPs were used to detect whether a given bug report is a defect or not, based on textual content, in the work of Terdchanakul *et al.* [32], which also used *N-Gram Inverse document frequencies (IDF)*. Budhiraja *et al.* [10] used MLPs to encode bug reports and then used these encoding to detect duplicates.

2) DEEP LEARNING MODELS WITH BIDIRECTIONAL ENCODER REPRESENTATIONS FROM TRANSFORMERS (BERT)

Attention mechanisms are a way to weigh the input features based on their importance for a given task [33]. Researchers such as [12] cite the possibility of using attention in a customized deep architecture but did not experiment with them. Since the introduction of attention mechanisms by [34] and [35], hard problems involving a long sequence of texts like machine translation showed significant improvements in quality. Previous solutions usually deployed RNNs and CNNs are thus becoming less usual in textual processing contexts, giving rise to new concepts such as transformations, which were introduced by Vaswani *et al.* [33].

The most popular natural language tasks have seen significant improvements using attention techniques [35]–[38], with models based on the Bidirectional Encoder Representations from Transformers (BERT) [39] being particularly successful. An approach used for general-purpose natural language processing deploys attention mechanisms to learn contextual relations between words in a text [18]. Its attention vectors, which form the central core of BERT, works as weighted features. For example, imagine a sequence X formed by n words $\{X_1, X_2, .., \text{and } X_n\}$, with each word identified by a continuous vector (word embedding) of size m representing its learned latent features. A high-level view of the *attention* output generated by BERT is one matrix of n lines and m columns to represent each weighted word that the model generated, transformed by a positional function f. For each line in the *attention* matrix, a new vector of size m is calculated from an average of all X input dot with weights, as defined in (1).

$$attention[_{n,m}] = \frac{1}{n}\sum_{i=1}^{n} f(X_i).w \tag{1}$$

BERT can be useful in many tasks such as question answering, natural language inference, and paraphrase detection [40], [41]. The problem studied can be seen as similar to paraphrase detection, since we are trying to detect whether two texts (along with their categorical features) of duplicate bug reports are describing the same subject using a different vocabulary.

3) SIAMESE MODELS TO COMPARE BUG REPORTS VIA TRIPLET LOSS

Siamese models are neural networks modeled to compare pairs of objects, which has a high capacity to generalize representations about the compared objects and to learn latent similarities between the instances. This architecture is modeled to accept all kinds of entries through any number of networks [42]. In the NLP community, several works predominantly employ Siamese architecture [43]–[45]. In this architecture, the inputs can be images, texts, or another object. Moreover, while these entries are transformed into latent vectors in the same resource space, the model shares weights between their

TABLE 2. Comparison between related works for duplicate bug report detection.

Approach	Author	Information	Dataset	Method	Task	Detection
ML / DL	[12]	hybrid-structured	Eclipse NetBeans Open Office Combined	Siamese MLP Bi-LSTM CNN Word Embedding (Glove)	Classification Retrieval	72-82% 50-81%
	[10], [11]	unstructured	Open Office Firefox	MLP Word Embedding (CBOW, SkipGram)	Classification Retrieval	60-94% 21-77%
	[17]	hybrid-structured	Mozilla Eclipse NetBeans Gnome Open Office Firefox Combined	Siamese CNN Word Embedding (Word2Vec)	Classification Retrieval	85-99% 79-94%
	[15]	hybrid-structured	Eclipse NetBeans Open Office Combined	DC-CNN Word Embedding (Word2Vec)	Classification	94-95%
	[14]	hybrid-structured	Snap S2R Eclipse Platform Eclipse JDT Mozilla Firefox	Siamese Bi-GRU Word Embedding Self-Attention Conditional Attention	Classification	70-88%
	[13]	hybrid-structured	Hadoop Hdfs MapReduce Spark	CNN Word Embedding (Random, Glove, Word2Vec) Context	Classification	82-94%
IR	[3]	unstructured	Private	TF/IDF	Retrieval	31-42%
	[26]	hybrid-structured	Eclipse Firefox	TF/IDF	Retrieval	42-71%
	[27]	hybrid-structured	Mozilla	TF/IDF Graph	Retrieval	27-43%
	[25]	unstructured	Open Office Firefox Eclipse	TF/IDF SVM	Retrieval	31-69%
	[5]	hybrid-structured	Open Office Mozilla Eclipse Large Eclipse	$BM25F_{ext}$	Retrieval	38-72%
	[6]	hybrid-structured	Open Office Mozilla Eclipse	BM25F LDA	Retrieval	40-82%
	[28]	hybrid-structured	Android Open Office Mozilla Eclipse	BM25F Context	Classification	84-92%
	[16]	hybrid-structured	Android	KNN LR IGR CS GI PCA	Classification Retrieval	97% 34-51%

networks [46]. Aiming to illustrate the approach, Figure 2 details a case of two networks, *Network A* and *Network B*, where they learn to encode two entries, *Input A* and *Input B*, to space Y, reducing the vector distance between similar inputs while increasing the distance between distinct through its similarity function. The main idea of this strategy is to use grouped neural networks that share weights W, ensuring that latent embeddings are learned in the same feature space. Moreover, the A and B networks can be neural networks with any configuration of layers.

The shared weights can be seen as an advantage since this means the network uses a comparably smaller number of parameters (while representing two objects, only one set of parameters is used). Moreover, they also lead to advantages in training, since the weights are adjusted for both examples at each backpropagation gradient computation, while also producing a single embedding spacing for visualization, grouping, and other purposes [47]. In the context of duplicate bugs, this strategy was used by [12], aiming to reduce the distance of the duplicates in the embedding space produced

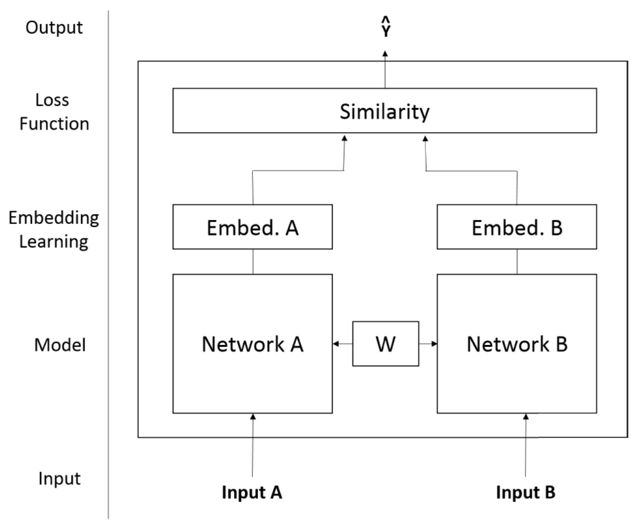

FIGURE 2. Siamese model to compare two objects.

by the model while increasing the distance between non-duplicates. To do so, it uses the Triplet Loss (TL) as its loss function, which was originally introduced by [21] for the face recognition problem.

The TL is a loss function defined as a measure to orient how well the model learns and gives directions to improve its learning [44], [48], aiming to maximize a similarity metric between similar instances and minimize between dissimilar. In this context, the Siamese model receives three instances, an input anchor, a duplicate of the anchor (the positive example), and a non-duplicate (the negative example). Thus, the loss value has a continuous range output as (2), which is the variation of TL used [12] demonstrates.

$$Max\{0, C + Cosine(F_A, F_P) - Cosine(F_A, F_N)\} \qquad (2)$$

The C constant applies a distance margin between the similarity values, and is usually set to 1 but can be freely set according to model needs. The instance A is the candidate, called anchor, P is a positive example of a duplicate of A, N is a non-duplicate or negative example regarding A. The instances (A, P, N) received a function F transforming in (F_A, F_P, F_N) that generates an associated representation for each instance. The function TL then aims to maximize the cosine similarity [4] between the embedding vector of F_A and F_P, while minimizing for F_A and F_N. In a duplicate detection scenario, this function brings duplicates closer while pushing away non-duplicates, and was used by [12] for the duplicated bug report task. However, Triplet Loss will cluster two duplicated instances in the latent space, disregarding information on the whole cluster of duplicates of the instances during this training step. In SiameseQAT, we expand on this idea by introducing information regarding clusters of duplicated reports during training, as we will better describe in Section IV.

[4] While [12] using cosine similarity, any distance metric could be used

B. DUPLICATE DETECTION FRAMEWORK

Duplicate bug report detection can be formulated as either a duplicate retrieval or a binary classification task, and the framework we propose in this paper can be applied for both tasks. Figure 3 shows an overview of our framework. Our proposal is to generate bug report embeddings and to use these representations on retrieval and classification tasks, training our model using quintet instances, containing a random report (the anchor), a duplicate (the positive example), and a non-duplicate report (a negative example), as well as two extra elements: an embedding of the group of all duplicates of the positive example (the positive centroid) and the same for the group of all duplicates of the negative example (the negative centroid).

Our training is driven by the Quintet Loss function if training for generating bug report representations for the retrieval task or binary cross-entropy for the duplicate pair classification task. Our framework can be divided into three main stages: (1) *Model Training*, (2) *Duplicate Retrieval*, and (3) *Duplicate Classification*. Those three stages are further explained below:

1) **Model training**: In this phase, we train SiameseQAT using bug report examples. First, each bug report is preprocessed, extracting textual tokens, topic distributions extracted using Latent Dirichlet Allocation [20] applied to the whole training dataset, and one-hot-encoding representations for categorical information. This dataset is used to generate example quintets for training our models (the details of the models are better explained in section IV-C). The retrieval model is trained first in order to generate the bug report embeddings. Then the classification model is trained using the same embeddings generated by the retrieval model, i.e., the weights of the SiameseQAT model trained for retrieval are used to generate the binary classifier model's input. The stage's output is two neural network models, one for generating bug report embeddings and one for classifying two reports. Besides these models, embedding representations of all reports used in training are also stored in a dataset for future retrieval tasks.
2) **Duplicate retrieval**: In this phase, the framework receives as input a new bug report and retrieves the top-k reports most likely to be a duplicate of the input. To do so, it must first generate its embedding representation by going through the same preprocessing process and using SiameseQAT. This embedding representation is then used to retrieve the duplicate bug reports with a higher probability of being a duplicate to the query, and a *triager* (or a user, or an automatic classification method) can identify whether these candidates are indeed replicas.
3) **Duplicate classification**: Our framework can also be used to automatically determine whether a pair of reports are duplicates or not. The reports must also be preprocessed and have their embeddings generated by

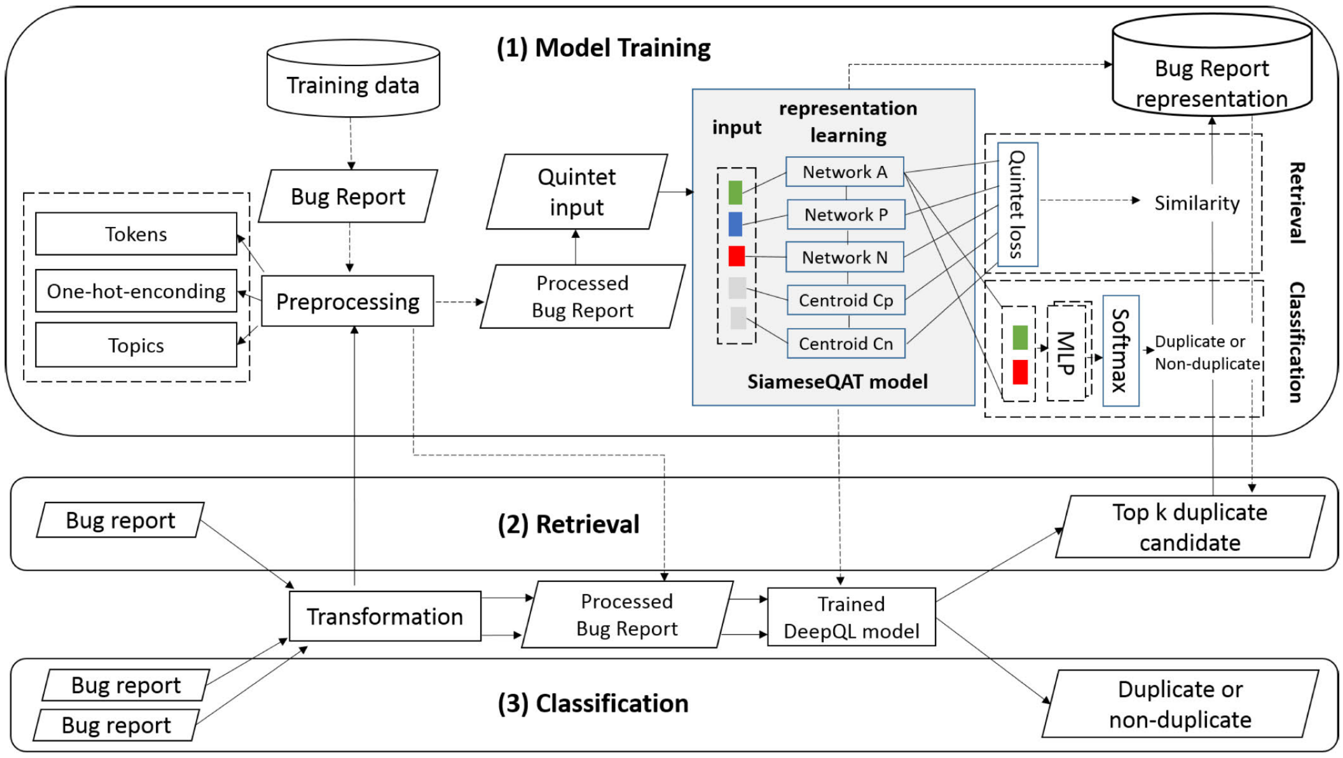

FIGURE 3. Framework to detect duplicate bug reports split into three phases: (1) Model training; (2) Retrieval; and (3) Classification.

the SiameseQAT model. Then, the previously trained binary classifier can be used to evaluate whether they are indeed duplicates. By using the same model to generate the embedding for the retrieval and classification tasks, the embeddings of previously-stored reports can be used without repeating the whole process, making it simpler to compare a larger number of reports.

C. SIAMESEQAT: A SIAMESE ATTENTION-BASED AND SEMANTIC CONTEXT-AWARE NEURAL NETWORK FOR DETECTING DUPLICATED BUG REPORTS USING DUPLICATED CLUSTER INFORMATION

Figure 4 presents a high-level representation of SiameseQAT, a deep Siamese neural network for representing bug reports through embeddings. Our approach builds and improves the initial idea by [12] on three fronts: (1) An improved loss function that also takes into consideration the information on clusters of duplicated reports; (2) attention-based embedding of textual features to better represent bug report titles and descriptions and (3) inclusion of topic modeling-based information to boost the textual embedding of the report, which, according to [6] can also improve the detection of duplicates and add global information to the reports.

Thus, the objective of our model is to learn bug report embeddings by extracting semantic context-based features from the title, description, and categorical features of not only the anchor report and the positive and negative examples, but also an aggregate embedding of their same groups/clusters (both positive and negative). In Figure 4, this information is shown as centroids C_P and C_N, respectively the aggregate embedding of the duplicates of the anchor/positive examples, and the duplicates of the randomly picked negative example duplicates. Our model is composed of five components: (i) input, (ii) model, (iii) embedding learning, (iv) loss function, and (v) output:

(i) **Input**: Each instance in the input layer receives three bug reports: the anchor, a positive, and a negative example, represented by their titles, descriptions, LDA topic distributions extracted from the concatenation of those two textual fields, and categorical information (encoded as one-hot vectors). In this component, SiameseQAT receives as input a quintet (A, P, N, C_P, C_N), where A is an anchor bug report, P a positive example of a duplicated report, and N a negative example. Besides these three reports, it also receives two embedding vectors representing duplicated report clusters C_P and C_N, where C_P is the embedding *centroid* of the cluster of positives examples (the group of reports to which A and P belong), C_N is the centroid of the cluster of the negative example N. A *centroid* is the average of the embedding vectors of all reports in a given duplicate cluster. Negative example instances are selected using the semi-hard triplet method suggested by [21], where $distance(A, P) < distance(A, N) < distance(A, P) + margin$.

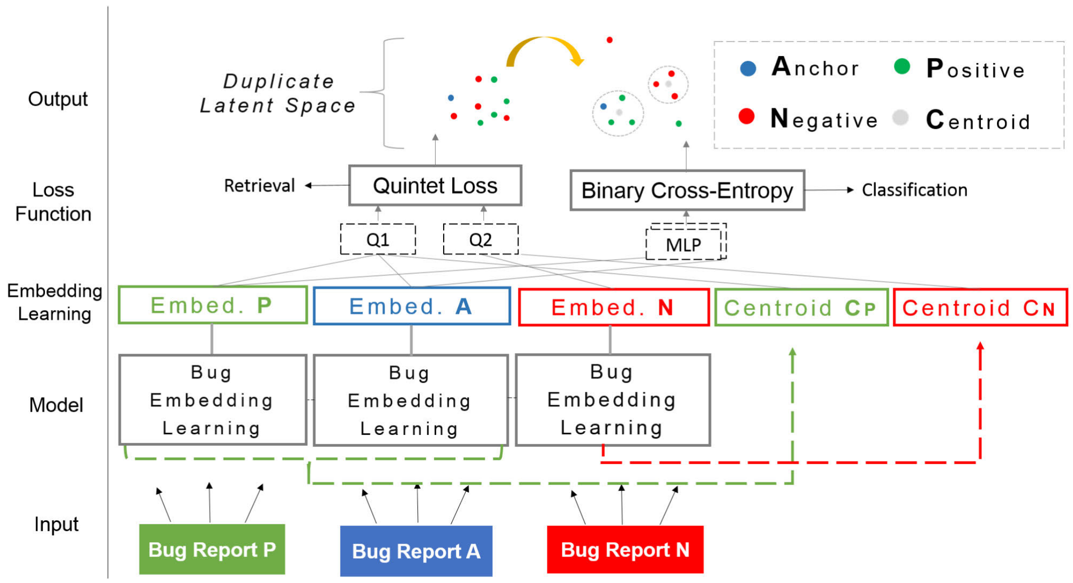

FIGURE 4. SiameseQAT: Deep Siamese model to learn duplicates bug report embeddings and matches quintet examples using an anchor, a positive, a negative, and their centroids.

(ii) **Model**: This component is responsible for generating semantic context-based embedding for bug report documents, building learning patterns, and it receives its input from component (i). The three boxes tagged as *bug embedding learning* in Figure 4, are Siamese deep learning models formed by two types of neural network layers, BERT and MLP. Two sets of BERT layers are used for processing textual features, short description and description separately, similar to what was done in [12], except that we replace LSTM and CNN layers by attention-based BERT layers. The MLP layers are used for processing two kinds of information: categorical features (product, component, priority, bug severity, resolution, bug status, and version) and LDA topic distribution. Thus, as illustrated in Figure 5, bug embedding learning is composed of four sets of layers: BERT layers for the title, BERT layers for description, MLP to process topics, and MLP to process categorical information. The output of those four sets is then combined by a concatenation layer, which is the *embedding representation* of the bug report. The architecture is shown in Figure 4 is a Siamese network, where a single set of weights is used to learn the latent feature of the 3 bug reports in the input, which is a common approach for duplicate detection using deep neural networks [12].

(iii) **Embedding Learning**: This component is a concatenation of the four embedding representations learned by the model described in Figure 5. In SiameseQAT, each component embedding has the same size D, and thus the final embedding representation of a bug report is a vector with a size of $4D$. This vector can then be used for retrieval and classification tasks.

(iv) **Loss Function**: This component may apply one of two mathematical functions, depending on the task at hand: Quintet Loss and binary cross-entropy. The Quintet Loss aims to model the similarity between anchor bug report's embedding vectors, the positive/negative examples, and their respective centroids generated by component (iii). The objective is to minimize the distance between the anchor report and its positive examples while maximizing the negative example distance, which can then be used to generate report embedding datasets for retrieval tasks. Its output is an (weighted or not) average of these distances. The details of our motivations and the effect of using Quintet Loss are explained in section IV-D. The binary cross-entropy aims to minimize the error between the predicted value and the real value for the binary classification of pairs of duplicate reports.

(v) **Output**: The output of SiameseQAT is embedding bug report representations as vectors in a latent feature space. As the training progresses, we expect that duplicated reports end closer than non-duplicated reports.

D. QUINTET LOSS (QL): A LOSS FUNCTION BASED ON REPLICATED CLUSTER INFORMATION

The Triplet Loss (TL) function was introduced by [49] and [21], aiming to make the latent space generated by a deep learning model cluster similar instances [50]. This loss does not consider a global data distribution since training over all possible triplets is expensive and usually is employed via random sampling [22]. Furthermore, all triplets are treated equally with a constant that violates the margin distance between the instances, which motivates approaches to investigate re-weighting distribution in training samples [23]. Inspired by these issues, we decided to improve on the TL

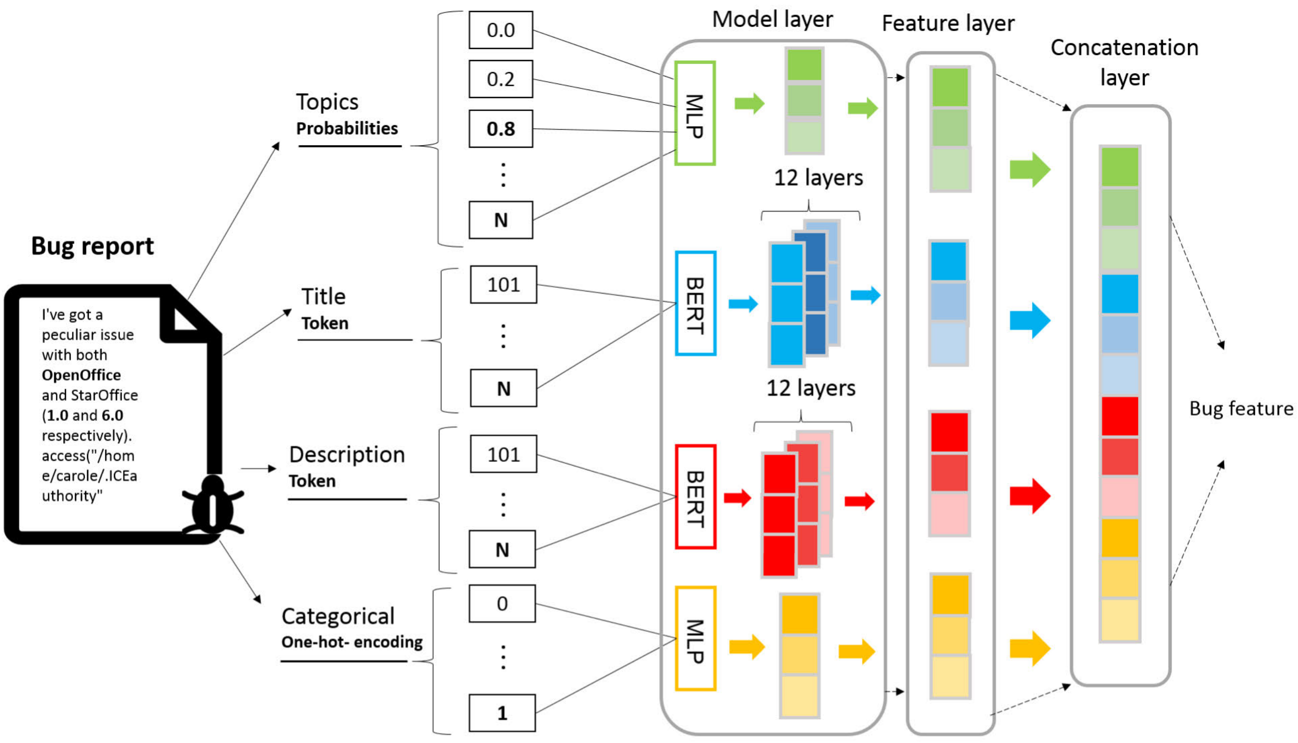

FIGURE 5. Bug embedding learning proposed to extract a semantic context-based features.

function, introducing a novel loss, Quintet, which also adds information about duplicates groups alongside individual examples, by including duplicate *cluster* centroid information in tandem with positive and negative examples.

Thus, the idea behind Quintet Loss is twofold: (i) to avoid the equal importance issue of the original triplet training by adding centroids and new loss function weights, also (ii) to make the model generate more well-defined and discriminate clusters, by adding information regarding all replicas of the anchor present in the batch (the centroid).

In doing so, we expect to improve the learning of training instances aiming to avoid examples with equal influence on the training phase described by [23], through added weights and a compound equation system. Thus, in Quintet Loss we compute two components, (3) and (4), which are combined in the new loss function (5).

$$Q_1 = Max\{0, C + Cos(F_A, F_P) - Cos(F_A, F_N)\} \quad (3)$$

$$Q_2 = Max\{0, C + Cos(F_A, C_P) - Cos(F_A, C_N)\} \quad (4)$$

$$L_{quintet} = \frac{(Q_1.W_1 + Q_2.W_2)}{(W_1 + W_2)} \quad (5)$$

On (3), (4), (5), the C constant is a constant value (set to 1 in our experiments), applying the distance margin between the cosine values for positives and negatives instances. This value, however, can be freely chosen according to the task at hand. The instance F_A is an embedding representation of a bug report chosen as an anchor, F_P is the embedding of a duplicate of F_A, F_N is the embedding of a bug report that is not a duplicate of F_A. Then, C_P is the *centroid* vector of all duplicates of F_P (including itself). This *centroid* is the average of those vectors. Similarly, C_N is a *centroid* vector of all duplicates of F_N in the dataset. The Q_1 component is the same as the original Triplet loss for F_A, F_P, and F_N, while the Q_2 equation minimizes the distance between the anchor F_A and the *centroid* of the positive cluster C_P while maximizing its distance to the *centroid* of the negative C_N. In our proposed method, we use the cosine between vectors as the similarity metric.

The *cosine similarity* in the Quintet Loss formula is defined by (7), where Q is the query vector, and J is the vector that has been compared. The $Cos_{Distance}$ function in (6), measures the cosine distance between vectors Q and J. Thus, the cosine distance finds the similarity using the Euclidean dot product, resulting in ranges of -1 when the vectors are opposite, and 1 when they are identical. A zero value indicates orthogonality or no correlation.

$$Cos_{Distance}(Q, J) = \frac{\sum_{i=1}^{N} Q_i J_i}{\sqrt{\sum_{i=1}^{N} Q_i^2}\sqrt{\sum_{i=1}^{N} J_i^2}} \quad (6)$$

$$Cos(Q, J) = 1 - Cos_{Distance}(Q, J) \quad (7)$$

The overall loss function in (5) is the weighted average of Q_1 and Q_2. In this paper we propose two alternatives to set these weights. Our first approach is simply to use the same weights for both components, turning the Quintet

TABLE 3. Dataset statistics over all duplicates and features from bug reports.

		Eclipse	NetBeans	Open Office
Duplicates	Number of duplicates	233.415	248.666	132.518
	Number of unique reports	271.901	162.921	52.440
Duplicate Clusters	Number of clusters	23.812	19.050	6.132
	The smallest duplicates cluster	2	2	2
	First quartile duplicate cluster size	10	12	11
	Median duplicate cluster size	19	23	20
	Third quartile duplicate cluster size	28	36	29
	The largest duplicate cluster	50	56	99
Textual	Number of distinct words*	20.000	19.061	18.562
	Average length of title	15	16	13
	Average length of description	326	796	207
Categorical	Number of distinct bug_severity values	7	7	6
	Number of distinct bug_status values	3	3	3
	Number of distinct component values	927	473	137
	Number of distinct priority values	5	4	5
	Number of distinct product values	189	39	41
	Number of distinct version values	547	18	537

* Size of the vocabulary restricted to the top-20.000 most frequent words due to memory restrictions.

Loss function into a simple average of the two components. The second approach is to add those two weights as trainable parameters in SiameseQAT and let the training help chose the fittest weights for the training dataset.

V. EXPERIMENTS AND RESULTS

In this section, we present the experiments we performed to evaluate the impact of using SiameseQAT in the retrieval and classification of duplicated bug report tasks compared to our baselines. We also present a qualitative analysis of the embedding representations of reports generated by the evaluated methods.

The model and baselines were implemented using Keras [51], a high-level open-source deep-learning library written in Python. All experiments were carried on an Intel i7-7700 server with 64GB RAM and two Nvidia GeForce GTX 1080 Ti video cards. All source code and datasets are available on a public repository[5] for reproducibility purposes.

A. DATASETS

In our experiments, we used three bug report repositories of large open-source projects that are popular for this detection task in the literature: Eclipse, NetBeans, and Open Office, collected and published by [52]. These datasets contain a large number of reports and the ground truth regarding their duplicates. We present many auxiliary statistics of those datasets in Table 3. To evaluate our approach, we split the datasets into train and test sets of 90% and 10% of the total duplicate clusters, respectively. Reports without duplicates were considered as clusters containing a single report for the split. Table 4 shows the number of duplicate pairs used for each division. These splits are used to train and test the retrieval and classification model.

TABLE 4. Train and test split.

Dataset	Train pairs	Test pairs	Bug reports
Eclipse	76.638	8.809	334.422
NetBeans	87.532	9.904	216.715
Open Office	52.246	4.116	72.234
Total	**216.416**	**22.829**	**623.371**

The attributes used for training the retrieval and classification models were textual (short description and description) and categorical (product, component, priority, bug severity, resolution, bug status, and version). Attributes such as date (delta_ts and creation_ts) or bug_id were ignored.

B. PREPROCESSING BUG REPORTS

We truncate each bug report textual field, such as short description and description, at the first 100 tokens. We also performed preliminary experiments truncating at 20, 50, 150, and 200, and found 100 achieved the best results. Then, we applied LDA to extract topics of the whole training dataset, where a document was comprised of a concatenation of its short description and description. We chose to extract 30 topics from the datasets, after evaluating topic sizes 10, 20, 30, 40, and 50. Those configurations are exhaustively tested via Grid Search.[6] We also represented all categorical features as one-hot encoded vectors.

C. BASELINES

We compared SiameseQAT with two state-of-the-art baselines to detect duplicate bug reports. We focused our evaluation on Deep Learning-based approaches since [10] demonstrated they achieved significantly superior results compared to previous IR-based methods. It is important to note that none of the two baselines have presented a direct

[5]https://github.com/thiagomarquesrocha/siameseQAT

[6]https://scikit-learn.org/

comparison between them using the same datasets, and neither cites the other. Thus, this paper will also present this comparison.

The two chosen baselines are:

- **DWEN**: Deep Word Embedding Neural Network (DWEN) proposed by [10] and [11]. The approach was modeled to classify a pair of bugs as duplicated or not, considering only textual features on a model based on the Skip-Gram method. The model outputs an embedding vector for the bug and the candidate, then encoding the pair using a Sigmoid function to generate as output the probability of them being duplicated. The original approach [10] does not evaluate the model for classification tasks. However, in [11] examination, the authors have extended the research to include classification experiments. Thus, in our experiments, we use the Sigmoid layer from DWEN for binary classification. Then, the layer before the Sigmoid layer was used to encode the bug reports for all datasets and employed for retrieval tasks. In comparison with SiameseQAT, DWEN does not extract nor use text topics from the datasets, does not employ attention mechanisms, and disregards bug report challenges with stack traces, logs, and source code among textual content. Also, they have not used categorical features.
- **DMS**: A Deep Siamese Model (DMS) proposed by [12] creates a single vector made by combining the short description, description, and categorical (product, component, priority, bug severity, resolution, bug status, and version) of bug reports. This baseline used three kinds of neural network layers: Multilayer Perceptron (MLP), Convolution Neural Network (CNN), and Bidirectional Long-Short Term Memory (BiLSTM) to generate bug report embeddings. DMS is similar to our approach since it also uses a Siamese topology, textual and categorical features, and is evaluated both in classification and retrieval tasks. However, it uses the traditional Triplet Loss, does not use attention mechanisms, and also does not include topics extracted from the bug report representation dataset.

We employed the same settings and parameters described in the respective papers in all baseline implementations.

D. EVALUATION METRICS

1) RECALL@K

Previous works, such as [10] and [12], use recall rate as a metric to evaluate the quality of their methods in retrieving duplicate bugs given a new report as a query. The output of this query is a ranked list of possible duplicated bug reports, ordered by likelihood of being duplicated. The recall rate is the ratio of reports with at least one duplicated report in their top-k lists over the total number of reports evaluated. The measure is expressed by (8).

$$Recall@K = \frac{1}{k}\sum_{t=1}^{k}\frac{duplicate}{1}$$
$$duplicate = \begin{cases} 1, & \text{If found at least one duplicate in K bugs} \\ 0, & \text{otherwise} \end{cases} \tag{8}$$

In equation 8, the K value is the number of reports considered in the evaluated queries, and *duplicate* is the conditional equation to detect when at least one duplicate bug is returned.

2) ACCURACY

In the classification task, we used the traditional accuracy measure. For each pair of reports evaluated, the accuracy is the percentage of pairs that are correctly classified.

3) AREA UNDER THE RECEIVER OPERATING CHARACTERISTIC CURVE (AUROC)

We also use AUROC as a classification metric, used to distinguish the separability between two classes. It measures the relationship between the True Positive Rate (TPR) and False Positive Rate (FPR). Given that TP is the number of true positives, TN the number of true negatives, FP the number of false positives, and FN the number of false negatives. The TPR is expressed in (9), while the FPR is formulated in (10), and Specificity in (11).

$$TPR = \frac{TP}{(TP+FN)} \tag{9}$$
$$FPR = 1 - Specificity \tag{10}$$
$$Specificity = \frac{FP}{(TN+FP)} \tag{11}$$

The best performance is a score of 1, with 0.5 being the score of a random model that classifies 50% of the time as positive and 50% as negative. We used this metric to evaluate the probability of discovering duplicates considering the sensibility over the false positives.

4) SILHOUETTE SCORE

One of the expected effects of representing bug reports as embedding vectors in latent feature space using SiameseQAT is that it should lead to a clustering effect, i.e., duplicated reports should be represented close to each other in this space. Thus, it should be interesting to measure the separability of those duplicate clusters, and for this, we use the *Silhouette* metric. It assumes that there exist K clusters on data. Then the Silhouette coefficient can be defined as equation (12).

$$Silhouette = (b-a)/max(a,b) \tag{12}$$

where a is the mean distance between a sample of the same cluster and b is the distance between a sample and the nearest cluster that the sample is not a part of. The Silhouette ranges from -1 to $+1$, where a high value indicates well-discriminated clusters, and a negative value can indicate low separability and overlaps.

TABLE 5. Proposed variations.

Method	Components
SiameseTA	Triplet Loss + Attention
SiameseTAT	Triplet Loss + Attention + Topics
SiameseQA-A	Quintet Loss (averaged weights) + Attention
SiameseQA-W	Quintet Loss (learned weights) + Attention
SiameseQAT-A	Quintet Loss (averaged weights) + Attention + Topics
SiamseseQAT-W	Quintet Loss (learned weights) + Attention + Topics

E. TRAINING AND HYPERPARAMETERS

To better understand the individual contributions of our proposal, we performed experiments with some variations of SiameseQAT. The variations are based on using or not either topic information or Quintet Loss, and in the case of using Quintet Loss whether the weights are averaged or with learned weights. For easier visualization in the results, we added letters in the method name suffix to better identify which components are used, as shown in Table 5.

Neural Network Setup: On our SiameseQAT method and its variations, we have two sets of BERT layers (for encoding textual information) and two sets of MLP layers (to encode categorical information and topics), with the final bug report embedding vector having 1.200 neurons, where every 300 neurons are for each feature set, as shown in figure 4. Similarly, variations without topic distribution features had 900 neurons. We used 100 words as sentence length for each textual feature, and 30 topics for each bug report document. The BERT layers were pre-trained with the model 'uncased_L-12_H-768_A-12', which has weights for a billion tokens pre-trained on a large corpus.[7] The MLP layers have 600 fully connected units using hyperbolic tangent as its activation function.

For BERT, we used 12 layers present in the architecture of [18], thawing eight layers for training. This adopted architecture is inspired by [39] recommendations, which clarify the use of BERT as a feature extractor to encode textual sequences and output semantic embeddings.

We used the *model* layer in architecture Figure 4 to encode incoming bug reports for the duplicate retrieval task. Also, we used the same layers present in the retrieval model and transferred their pre-trained weights while freezing them from further modification for the classification task. Then we added two MLP layers with 64 dimensions, using softmax as its output and binary cross-entropy as the loss function. All models and baselines have been trained using ADAM [53] as the optimizer.

F. RESULTS

In this section, we present the results achieved by SiameseQAT and the baselines. We divided our evaluation into retrieval, classification, and qualitative analysis. Deshmukh *et al.* [12] used the Eclipse, NetBeans, and Open Office datasets, while [10] used Open Office and Firefox. Despite using the same datasets, there is no information around the instances chosen by these authors in the holdout split. Thus, the numbers presented in our experiments are not a perfect match to those presented in those papers. In these datasets, given a duplicate set, all possible pairwise combinations of duplicated elements are considered a positive example.

[7]https://github.com/google-research/bert

1) PERFORMANCE UNDER DUPLICATE REPORT RETRIEVAL

Figure 6 shows the results obtained by the best variations of our methods, SiameseQAT and SiameseTAT compared to the baselines in terms of *Recall@k*. As can be seen, the proposed method achieves consistently superior results even when considering very few candidates, revealing recall rate levels between 8% and 18% better. In the Open Office dataset, the recall improvement was even steeper, with an increase of around 18% in recall rate when compared with the best baseline, DMS. This is a strong indication that the contributions proposed in this paper had a significant impact on separating distinct bug reports while keeping duplicates close. When comparing SiameseQAT and SiameseTAT, they achieved similar results in all datasets, with a slight advantage for SiameseTAT, albeit not statistically significant, as is presented in table 11. Thus, we believe that it is also important to understand the contribution of each component SiameseQAT in this visible increase in recall compared to the baselines.

Tables 6, 7 and 8 show the results achieved by all variations of the components of SiameseQAT, and the baselines, for the three evaluated datasets.

Effectiveness of BERT: When examining the attention/BERT component, we could see an improvement of 19% to 25% in top@k at level 1 and 8% to 18% in top@k at level 25 in recall levels compared to the baselines for all variations proposed, including variations that do not include topic modeling information (SiameseQA and SiameseTA). Overall, the addition of attention did indeed show gains as expected from its use in other NLP tasks.

Effectiveness of Topic Modeling: In the evaluation of the topic probabilities feature, we can see that in all datasets, their inclusion leads to consistent increases in recall for all values of K considered in comparison to the same variation without topics, with absolute increases ranging from 0.01 and 0.04 on the Eclipse and NetBeans. On the Open Office, this increase was slightly smaller, ranging from no increase to 0.03.

Effectiveness of Quintet Loss: When considering the impact of using Quintet Loss, in the Eclipse and NetBeans datasets, using SiameseQAT-W, SiameseQAT-A, and SiameseQA-A lead to superior or equal results when compared to using Triplet Loss (SiameseTA and SiameseTAT), with gains around none to 0.01 for all recall levels, an improvement that, while small is consistent for all recall levels with statistically relevant gains, as will be better explained below. However, using Triplet Loss leads to better results on Open Office, even though for R@1 and R@25 SiameseQAT-W achieved better results. We believe that this difference in recall levels can be explained due to this dataset having a much smaller number of clusters and duplicates than the other two. This may indicate that Quintet Loss is more suited for datasets with a large number of duplicates. Nevertheless, the proposed improvements all lead to quality improvements in numerous scenarios.

Regarding the comparison between using the same weights versus trainable weights in Quintet Loss, as can be seen in tables 6, 7, and 8, the retrieval levels achieved were very similar, with slight advantages for using the same weights in Eclipse and NetBeans, and for trainable weights in Open Office. However, none of these differences were statistically significant, as will be more explored below.

Statistical Comparison Between All Methods: We summarize in Tables 9, 10, and 11 a comparison of the relative gain levels between all methods, with the value being the relative gain/loss percentage of the line method over that of the column in terms of R@5 for the three data sets, where we compared if the difference between them was statistically significant through the student's t-test considering $p < 0.05$ as rejecting the null hypothesis, represented by the value being underlined. Although we are showing the results for R@5, the results for other recall levels lead to similar conclusions, especially as the k value increases, and will be suppressed for brevity.

Something visible in Tables 9, 10, and 11 is that the gains obtained by the variations proposed in this work were substantial and statistically significant compared to baselines. When analyzing the result for each dataset, we can see that in the Eclipse and NetBeans datasets, although the gain values are different, the relationship between the methods and the statistical significance is similar in most comparisons.

The tables show that, regarding the addition of topics, it leads to statistically significant gains when comparing a method to its similar variation without topics in all datasets except for Open Office. The same effect can be seen regarding Quintet Loss, with its use leading to statistically significant gains in Eclipse and NetBeans while having a statistically significant loss in Open Office when compared with using Triplet Loss in a similar configuration. Nevertheless, as is presented in Section V-F2, using Quintet Loss leads to substantial gains in classification tasks in all datasets, including Open Office, which might offset this small perceived loss in retrieval levels.

Performance of Our Approach: In general, we can state that the use of the proposed improvements in our approach presented itself as a more robust alternative in terms of retrieval levels in all datasets used. The recall levels observed are up to 80%, as observed in Figure 6. This fact shows that in the majority case we can recover duplicates correctly with very few tries, considering all datasets.

2) PERFORMANCE UNDER DUPLICATE REPORT CLASSIFICATION

To evaluate the classification task, we used two metrics, accuracy and AUROC, to confirm how many times the models correctly classify a pair of duplicate reports. Table 12 shows the results for all methods, with the best performance in bold.

As can be seen, SiameseQAT has reached the overall best results with both metrics for all datasets with varying margins of difference between the approaches using BERT, topics, Quintet Loss (SiameseQAT-W and SiameseQAT-A), being technically tied in some scenarios and is the best result in the Open Office dataset using just BERT and Quintet Loss. Regarding DMS, its results were much more competitive. However, SiameseQAT was superior in terms of accuracy in all datasets, with its largest improvement being in the Open Office dataset (7.0%). In terms of AUROC, the increase was more modest, from 0.75 to 0.82 in this same dataset, with improvement on the other two datasets of 3%, a small margin of difference.

When comparing the components of SiameseQAT, the combination of all components leads to consistently superior results for Eclipse and NetBeans, however in the Open Office the use of Quintet Loss and BERT without topics leads the best result.

Effectiveness of BERT: Unlike the retrieval results, using just the BERT component with Triplet Loss and topics via SiameseTA and SiameseTAT did not achieve better results than DMS. This is a clear indication that components that aid the retrieval of duplicates may not be enough to create separation hyperplanes for classification tasks. The best result was obtained by combining BERT with Quintet Loss which presented substantial enhancements to accuracy levels, demonstrating the impacting of our proposed loss function for this task.

Effectiveness of Topic Modeling: When comparing the use of topics, we could see that it lead to consistent small improvements in the results, both when using Triplet Loss or Quintet Loss with BERT, except for the Open Office dataset.

Effectiveness of Quintet Loss: In the classification task, even when using our proposed method with BERT and Quintet Loss without topics (SiameseQA) it is better in all datasets in comparison to methods using Triplet Loss and BERT (with and without topics), demonstrating that the proposed Quintet Loss heavily impacts the classification task and can improve the accuracy of results. The results showed a strong indication that the use of Quintet Loss can lead to expressive improvements to classification results while also achieving competitive results in retrieval tasks, even when using all other components proposed in this paper. Regarding using average or learned weights in Quintet Loss, again unlike the retrieval task, the results obtained by using trainable weights were similar to simply averaging weights.

Performance of Our Approach: From the experimental results and bar chart presented in Figure 7, we could see that the variations of our methods that included the use of Quintet Loss achieved superior or at least tied results compared to the best baseline, DMS, especially in the Open Office dataset, which is a strong indication that the use of Quintet Loss lead to more well-defined clusters in the embedding space, aiding classification efforts. Our method using attention layers,

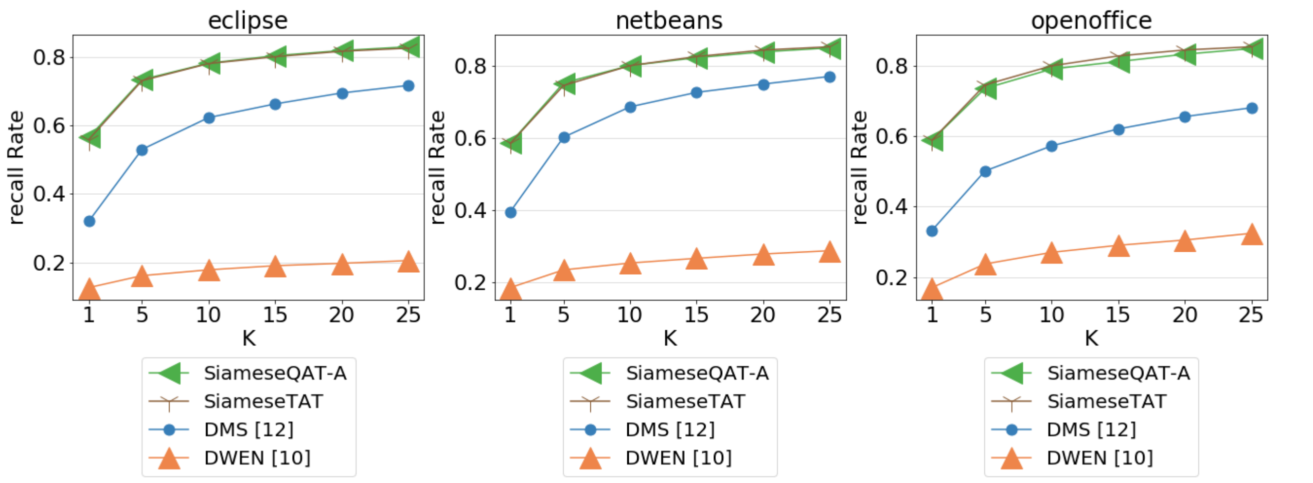

FIGURE 6. Recall rate of retrieval model for various values of K, and all baselines in comparison to the SiameseQAT and SiameseTAT.

TABLE 6. Recall@K rate for Eclipse and all approaches.

Eclipse						
	Recall@1	Recall@5	Recall@10	Recall@15	Recall@20	Recall@25
DWEN [10]	0.13	0.16	0.18	0.19	0.20	0.21
DMS [12]	0.32	0.53	0.62	0.66	0.70	0.72
SiameseTA	0.53	0.70	0.76	0.79	0.80	0.81
SiameseTAT	0.56	**0.73**	**0.78**	**0.80**	**0.82**	**0.83**
SiameseQA-W	0.53	0.70	0.74	0.77	0.78	0.79
SiameseQA-A	0.54	0.70	0.76	0.78	0.79	0.81
SiameseQAT-W	0.56	**0.73**	0.77	**0.80**	0.81	0.82
SiameseQAT-A	**0.57**	**0.73**	**0.78**	**0.80**	**0.82**	**0.83**

TABLE 7. Recall@K rate for NetBeans and all approaches.

NetBeans						
	Recall@1	Recall@5	Recall@10	Recall@15	Recall@20	Recall@25
DWEN [10]	0.19	0.24	0.25	0.27	0.28	0.29
DMS [12]	0.40	0.60	0.69	0.73	0.75	0.77
SiameseTA	0.57	0.72	0.78	0.80	0.82	0.83
SiameseTAT	**0.59**	**0.75**	**0.80**	**0.83**	**0.84**	**0.85**
SiameseQA-W	0.56	0.72	0.78	0.80	0.82	0.83
SiameseQA-A	0.56	0.73	0.78	0.81	0.82	0.83
SiameseQAT-W	**0.59**	0.74	0.79	0.82	0.83	0.84
SiameseQAT-A	**0.59**	**0.75**	**0.80**	0.82	**0.84**	**0.85**

topic distributions, and Quintet Loss achieved classification accuracy levels above 80% in all datasets.

3) PERFORMANCE OF CLUSTERING THE DUPLICATE BUG REPORTS

We explore how Quintet Loss and semantic context-based features through topic modeling and BERT organized the reports in the embedding space. To do so, we evaluate two aspects: (1) qualitative measures and (2) quantifiable measures, evaluating how well are given the duplicate clusters.

(1) Qualitative Measures of Duplicate Clustering: We selected a few duplicate clusters randomly to verify how the different methods arrange them in the space. For each method, we plotted the embedding space of 16 randomly sampled bug report duplicate clusters in a bi-dimensional space using t-SNE,[8] to better visualize how this space is organized. For brevity, we present this visualization only for the Open Office dataset, but similar behavior was found in all three datasets.

[8]https://scikit-learn.org/

TABLE 8. Recall@K rate for Open Office and all approaches.

Open Office						
	Recall@1	Recall@5	Recall@10	Recall@15	Recall@20	Recall@25
DWEN [10]	0.17	0.24	0.27	0.29	0.31	0.32
DMS [12]	0.33	0.50	0.57	0.62	0.66	0.68
SiameseTA	**0.60**	0.74	0.79	0.81	0.83	0.84
SiameseTAT	0.59	**0.75**	**0.80**	**0.83**	**0.84**	0.85
SiameseQA-W	0.57	0.72	0.76	0.79	0.81	0.83
SiameseQA-A	0.57	0.72	0.76	0.78	0.80	0.82
SiameseQAT-W	**0.60**	0.74	0.79	0.82	**0.84**	**0.86**
SiameseQAT-A	0.59	0.74	0.79	0.81	0.83	0.85

TABLE 9. Evaluation of the relative statistical gain among the recall@k values in the top 5 using the t-test for Eclipse in all approaches.

Eclipse								
	DWEN	DMS	SiameseTA	SiameseTAT	SiameseQA-W	SiameseQA-A	SiameseQAT-W	SiameseQAT-A
SiameseTA	337.5%	32.1%	-	-4.1%	0%	0%	-4.1%	-4.1%
SiameseTAT	**356.25%**	**37.7%**	**4.3%**	-	**4.3%**	**4.3%**	0%	0%
SiameseQA-W	337.5%	32.1%	0%	-4.1%	-	0%	-4.1%	-4.1%
SiameseQA-A	337.5%	32.1%	0%	-4.1%	0%	-	-4.1%	-4.1%
SiameseQAT-W	**356.3%**	**37.7%**	**4.3%**	0%	**4.3%**	**4.3%**	-	0%
SiameseQAT-A	**356.3%**	**37.7%**	**4.3%**	0%	**4.3%**	**4.3%**	0%	-

In bold the biggest gains.
Underline represents statistically significant results.

TABLE 10. Evaluation of the relative statistical gain among the recall@k values in the top 5 using the t-test for NetBeans in all approaches.

NetBeans								
	DWEN	DMS	SiameseTA	SiameseTAT	SiameseQA-W	SiameseQA-A	SiameseQAT-W	SiameseQAT-A
SiameseTA	200.0%	20.0%	-	-4.0%	0%	1.4%	-2.7%	-4.0%
SiameseTAT	**212.5%**	**25.0%**	4.1%	-	**4.2%**	**2.7%**	**1.4%**	0%
SiameseQA-W	200.0%	20.0%	1.4%	-4.0%	-	-1.4%	-2.7%	-4.0%
SiameseQA-A	204.2%	21.7%	1.4%	-2.7%	1.4%	-	-1.4%	-2.7%
SiameseQAT-W	208.3%	23.3%	2.8%	-1.3%	2.8%	1.4%	-	-1.3%
SiameseQAT-A	**212.5%**	**25.0%**	**4.2%**	0%	**4.2%**	**2.7%**	**1.4%**	-

In bold the biggest gains.
Underline represents statistically significant results.

TABLE 11. Evaluation of the relative statistical gain among the recall@k values in the top 5 using the t-test for Open Office in all approaches.

Open Office								
	DWEN	DMS	SiameseTA	SiameseTAT	SiameseQA-W	SiameseQA-A	SiameseQAT-W	SiameseQAT-A
SiameseTA	208.4%	48.0%	-	-1.4%	2.8%	2.8%	0%	0%
SiameseTAT	**212.5%**	**50.0%**	**1.4%**	-	**4.2%**	**4.2%**	**1.4%**	**1.4%**
SiameseQA-W	200.0%	44.0%	-2.7%	-4.0%	-	0%	-2.7%	-2.7%
SiameseQA-A	200.0%	44.0%	-2.7%	-4.0%	0%	-	-2.7%	-2.7%
SiameseQAT-W	208.3%	48.0%	0%	-1.3%	2.8%	2.8%	-	0%
SiameseQAT-A	208.3%	48.0%	0%	-1.3%	2.8%	2.8%	0%	-

In bold the biggest gains.
Underline represents statistically significant results.

In Figure 8(c), we can see the latent embedding space for SiameseQAT, where after training the selected duplicate clusters are well-formed, with almost no intersection between clusters. Figure 8(b) shows the plot for SiameseTAT where it can be seen the existence of some clusters with some mixing, such as the duplicates of 76895 and 23263. Finally, Figure 8(a) shows the same for DMS where not only the clusters are much more mixed, but there is also the presence of an outlier as a duplicate report of 93239 next to the duplicates of 23263. This fact is observed in SiameseTAT and SiameseQAT latent space, where there are fewer examples with the same behavior. These results may help explain the classification accuracy of SiameseQAT compared to the others since it has more well-formed clusters, which would make finding a separation hyperplane between a report and its non-duplicates easier. Moreover, those outlier's presence may harm recall tasks, especially if the query report is one of those outliers, where regardless of the k used, it would be tough to retrieve

TABLE 12. Accuracy and AUROC for binary classification, label duplicate or non-duplicate, 1 or 0.

	Eclipse		NetBeans		Open Office	
	Accuracy	AUROC	Accuracy	AUROC	Accuracy	AUROC
DWEN [10]	0.67	0.67	0.71	0.71	0.69	0.69
DMS [12]	0.85	0.85	0.82	0.82	0.75	0.75
SiameseTA	0.79	0.79	0.63	0.63	0.73	0.73
SiameseTAT	0.81	0.81	0.77	0.77	0.77	0.77
SiameseQA-W	0.86	0.86	0.83	0.83	**0.82**	**0.82**
SiameseQA-A	0.85	0.85	0.84	0.84	**0.82**	**0.82**
SiameseQAT-W	0.86	0.86	**0.85**	**0.85**	0.75	0.75
SiameseQAT-A	**0.87**	**0.87**	0.84	0.84	0.80	0.80

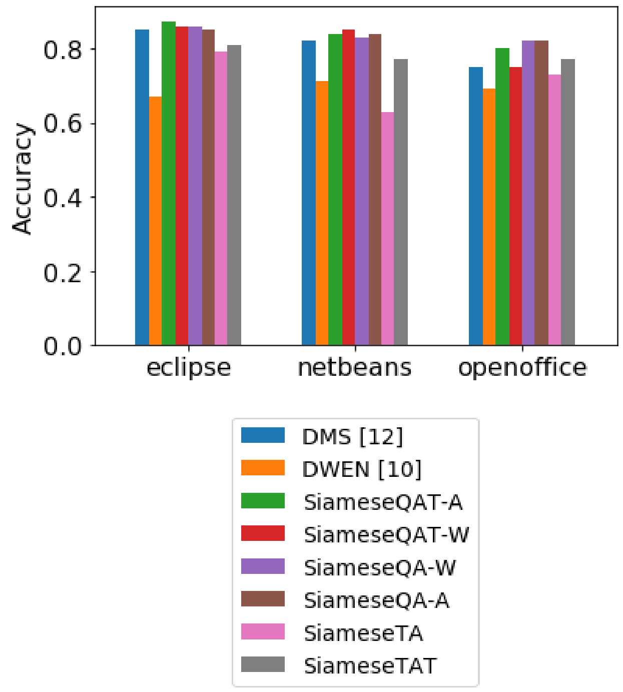

FIGURE 7. Classification comparison for all baselines and approaches using Accuracy metric.

a duplicate using the nearest neighbor approach. Thus, those outliers demand to be verified in future experiments to review which cases are true outliers, since that they can be a wrongly labeled duplicate or a bug report with a description lacking in detail, transforming them into an outlier, as suggested by [24].

(2) Quantifiable Measures of How Well Clustered Are the Duplicate Groups: We measured the separability of clusters using the Silhouette score for all methods in all datasets. The results are summarized in Table 13 and Figure 9, and different from the visualization presented previously for randomly duplicate clusters, here we measure how all clusters present for each dataset are well separated by measuring their Silhouette score. The results validate that the clusters proposed by our proposed method variations produce more separable clusters and a well-defined latent space according to the Silhouette score, with the SiameseQAT and SiameseTAT obtaining the best results in all datasets. When compared to the best baseline, the absolute gains observed in our methods

TABLE 13. Cluster results on test set using silhoutte score for all datasets.

	Eclipse	NetBeans	Open Office
	Silhouette	Silhouette	Silhouette
DWEN [10]	-0.31	-0.34	-0.37
DMS [12]	-0.03	-0.02	-0.03
SiameseTA	0.07	0.09	**0.07**
SiameseTAT	**0.08**	**0.10**	**0.07**
SiameseQA-W	**0.08**	0.09	0.06
SiameseQA-A	**0.08**	0.09	0.06
SiameseQAT-W	**0.08**	0.09	0.06
SiameseQAT-A	**0.08**	0.09	**0.07**

were of around 0.01 in all datasets, according to the measure, a small variation reflected in the more than 95 thousand clusters evaluated in all datasets.

Performance of Our Approach: Overall, these results confirm that using semantic context-based learning in tandem with Quintet Loss and topic distributions can produce more well-formed duplicate clusters, impacting both the duplicate retrieval and classification tasks for the three datasets evaluated.

4) PERFORMANCE OF SEMANTIC TEXTUAL ATTENTION

As an alternative to understanding how BERT is trying to recognize textual features and use attention mechanism to detect duplicate reports, we selected two examples of duplicated report sentences:

(a) Frequent workspace crashes;
(b) Out of memory error with Mylar?;

The two sentences describe an uncertain problem of "error" frequently causing application crashes. Then, we expect that the BERT model focuses on "error" and "crashes" words to determine the two sentences in a similar context. To allow us to visualize this learning, we used the BertViz visualization tool created by [54], which relates each word to each other for each layer and head present in the BERT model. Then, the final plot shows a word relation learned by the model. In Figure 10, sharper lines between two words mean more attention. It has the two sentences dispose of side by side, relating each word from layer #10 in head #10 of BERT.

In Figure 10, we can see that the model gives attention exactly to "memory," "workspace," "error," and "crashes,"

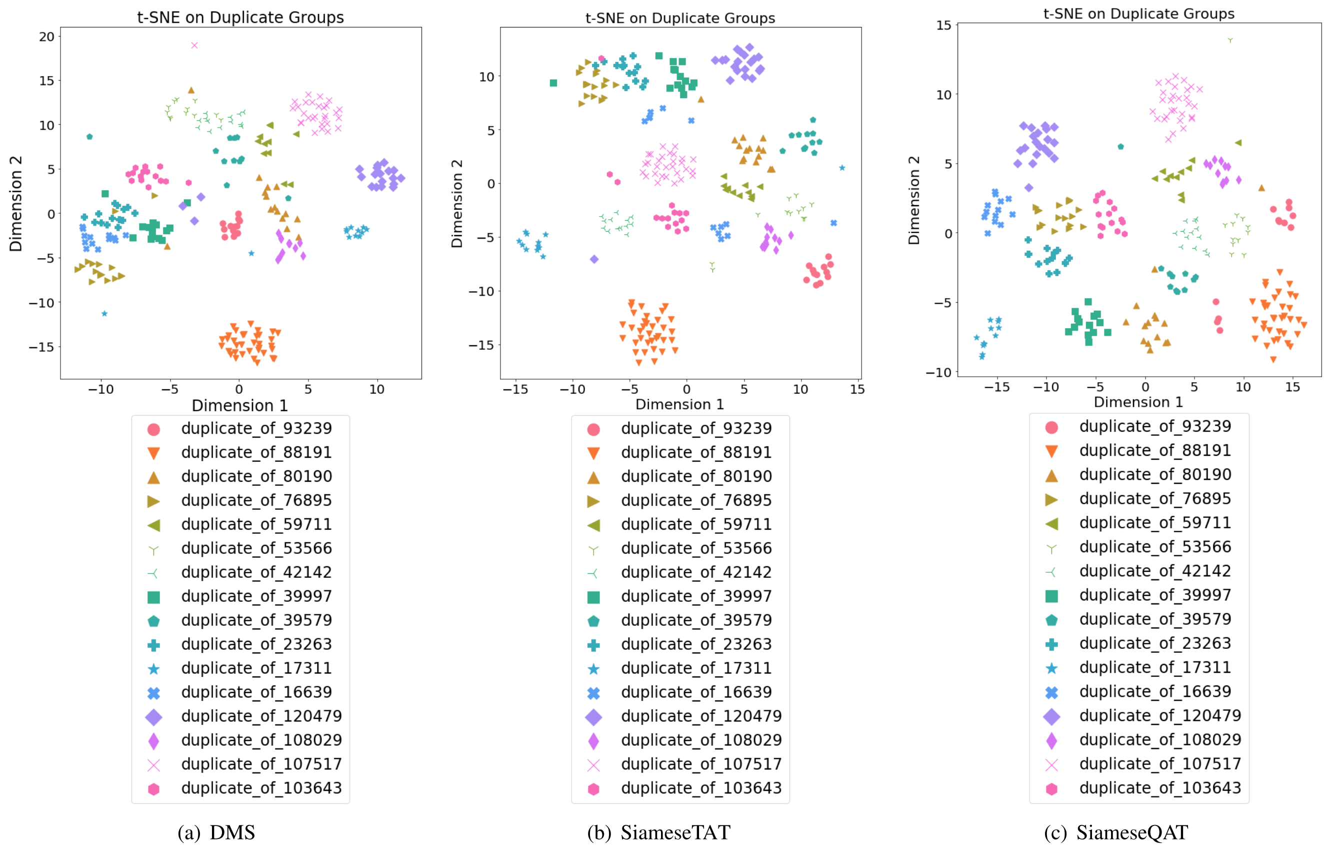

FIGURE 8. **Open Office duplicate latent space for 16 duplicate sets.**

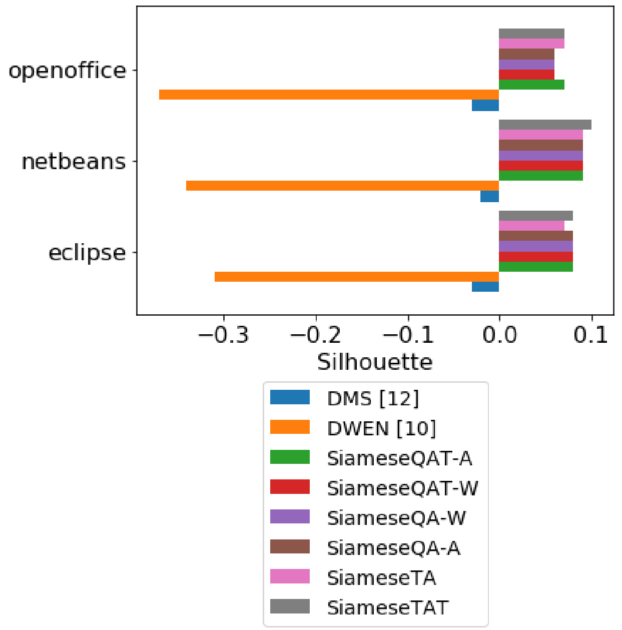

FIGURE 9. **Cluster quality evaluation using Silhoutte score for all datasets.**

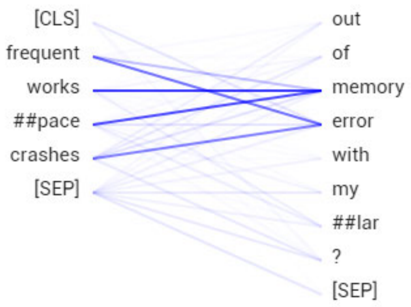

FIGURE 10. **Visualizing the layer #10, head #10 from pre-trained BERT for two-sentence duplicates titles, using BertViz visualization tool created by [54].**

also correlating the "error" with "frequent" and "crashes," getting a global sense for the problem, ignoring all other auxiliary words. This knowledge is useful to help discriminate between the two sentences, to match them as duplicates.

VI. CONCLUSION AND FUTURE WORK

We have introduced a novel duplicate detection framework using semantic context-based learning and a customized loss function, Quintet Loss. It has shown to improve significantly retrieval and classification accuracy, and the quality of the duplicate clusters on three public datasets, Eclipse, NetBeans and Open Office for more than 500 thousand bug reports. SiameseQAT is the first attempt at presenting a customized loss function specifically for deep learning-based detection of duplicate bug reports while also being the first attempt to merge semantic context-based within BERT and topic features to detect duplicate bug reports. We also provide the source code of our methods as well as the experimental data for the scientific community to employ and extend the use of the components proposed in this work.

Our reported retrieval results were well above the ones presented in previous works, with increases between 8% and 18% in recall rate, while accuracy and AUROC overcame the best baseline with an average accuracy of 84%. The results presented show that SiameseQAT produces more well-defined clusters of duplicate sets, impacting both classification and retrieval tasks. Nevertheless, there are many future works that we believe could be done to improve this work further. We highlight the following contributions:

(i) **Incorporate lightweight models**: While using BERT layers lead to gains in retrieval and classification, training BERT-based neural networks is more computationally expensive than most methods, due to its many parameters and numerous layers, which could compromise training in more massive datasets. Thus, as an alternative, we intend to deploy ALBERT, a lite version of BERT presented by [55], which could also lead to more freedom in the organization of layers in the model in comparison with BERT-based networks.

(ii) **Cross-domain and more data**: There are other public datasets[9] like Firefox, Thunderbird, Cassandra, and JDT available in the literature, which opens an opportunity to run experiments combining all datasets in a large, unified dataset [56]. This experiment would evaluate the model capacity to generalize an independent project representation of duplicate reports, which could be deployed in new projects with minimal effort.

(iii) **Process non-natural text separately**: We believe that, as done in [26], processing non-natural text, such as stack traces, logs, and code snippets separately from a natural text can lead to further improvements in detection accuracy and retrieval.

(iv) **Quintet Loss centroids**: Although we have strong evidence that this loss function leads to more well-defined clusters of duplicates in the latent space, we identified some centroids identical to negative examples in the training batch since no other duplicates of that same negative example appear in this batch. Therefore, the cluster centroid of this example is always itself, since no replica of the same report exists in the training batch. In this case, another strategy for selecting negative examples is needed, to evaluate the impact of optimal centroid that guarantees the construction of more representative embedding for duplicate instances, and consequently, impacts learning on SiameseQAT.

ACKNOWLEDGMENT

The authors would like to thank the Sidia R&D Institute and the Federal University of Amazonas for supporting the progress of this research.

[9]https://github.com/logpai/bugrepo

REFERENCES

[1] K. Aggarwal, F. Timbers, T. Rutgers, A. Hindle, E. Stroulia, and R. Greiner, "Detecting duplicate bug reports with software engineering domain knowledge," *J. Softw., Evol. Process*, vol. 29, no. 3, p. e1821, Mar. 2017.

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

[3] P. Runeson, M. Alexandersson, and O. Nyholm, "Detection of duplicate defect reports using natural language processing," in *Proc. 29th Int. Conf. Softw. Eng.*, May 2007, pp. 499–510.

[4] J. Uddin, R. Ghazali, M. M. Deris, R. Naseem, and H. Shah, "A survey on bug prioritization," *Artif. Intell. Rev.*, vol. 47, no. 2, pp. 145–180, Feb. 2017.

[5] C. Sun, D. Lo, S.-C. Khoo, and J. Jiang, "Towards more accurate retrieval of duplicate bug reports," in *Proc. 26th IEEE/ACM Int. Conf. Autom. Softw. Eng.*, Nov. 2011, pp. 253–262.

[6] A. T. Nguyen, T. T. Nguyen, T. N. Nguyen, D. Lo, and C. Sun, "Duplicate bug report detection with a combination of information retrieval and topic modeling," in *Proc. 27th IEEE/ACM Int. Conf. Autom. Softw. Eng.*, Sep. 2012, pp. 70–79.

[7] N. Klein, C. S. Corley, and N. A. Kraft, "New features for duplicate bug detection," in *Proc. 11th Work. Conf. Mining Softw. Repositories*, 2014, pp. 324–327.

[8] H. Rocha, M. T. Valente, H. Marques-Neto, and G. C. Murphy, "An empirical study on recommendations of similar bugs," in *Proc. IEEE 23rd Int. Conf. Softw. Anal., Evol., Reeng. (SANER)*, vol. 1, Mar. 2016, pp. 46–56.

[9] J. Xuan, H. Jiang, Z. Ren, J. Yan, and Z. Luo, "Automatic bug triage using semi-supervised text classification," 2017, *arXiv:1704.04769*. [Online]. Available: http://arxiv.org/abs/1704.04769

[10] A. Budhiraja, K. Dutta, R. Reddy, and M. Shrivastava, "DWEN: Deep word embedding network for duplicate bug report detection in software repositories," in *Proc. 40th Int. Conf. Softw. Eng., Companion*, May 2018, pp. 193–194.

[11] A. Budhiraja, K. Dutta, M. Shrivastava, and R. Reddy, "Towards word embeddings for improved duplicate bug report retrieval in software repositories," in *Proc. ACM SIGIR Int. Conf. Theory Inf. Retr.*, Sep. 2018, pp. 167–170.

[12] J. Deshmukh, K. M. Annervaz, S. Podder, S. Sengupta, and N. Dubash, "Towards accurate duplicate bug retrieval using deep learning techniques," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol. (ICSME)*, Sep. 2017, pp. 115–124.

[13] Q. Xie, Z. Wen, J. Zhu, C. Gao, and Z. Zheng, "Detecting duplicate bug reports with convolutional neural networks," in *Proc. 25th Asia–Pacific Softw. Eng. Conf. (APSEC)*, Dec. 2018, pp. 416–425.

[14] L. Poddar, L. Neves, W. Brendel, L. Marujo, S. Tulyakov, and P. Karuturi, "Train one get one free: Partially supervised neural network for bug report duplicate detection and clustering," 2019, *arXiv:1903.12431*. [Online]. Available: http://arxiv.org/abs/1903.12431

[15] J. He, L. Xu, M. Yan, X. Xia, and Y. Lei, "Duplicate bug report detection using dual-channel convolutional neural networks," in *Proc. 28th Int. Conf. Program Comprehension*, Jul. 2020, pp. 117–127.

[16] B. S. Neysiani and S. M. Babamir, "Automatic duplicate bug report detection using information retrieval-based versus machine learning-based approaches," in *Proc. 6th Int. Conf. Web Res. (ICWR)*, Apr. 2020, pp. 288–293.

[17] 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, pp. 200749–200763, 2020.

[18] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," 2018, *arXiv:1810.04805*. [Online]. Available: http://arxiv.org/abs/1810.04805

[19] S. Zafar, M. Z. Malik, and G. S. Walia, "Towards standardizing and improving classification of bug-fix commits," in *Proc. ACM/IEEE Int. Symp. Empirical Softw. Eng. Meas. (ESEM)*, Sep. 2019, pp. 1–6.

[20] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent Dirichlet allocation," *J. Mach. Learn. Res.*, vol. 3, pp. 993–1022, Jan. 2003.

[21] F. Schroff, D. Kalenichenko, and J. Philbin, "FaceNet: A unified embedding for face recognition and clustering," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2015, pp. 815–823.

[22] W. Ge, "Deep metric learning with hierarchical triplet loss," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 269–285.

[23] C.-Y. Wu, R. Manmatha, A. J. Smola, and P. Krahenbuhl, "Sampling matters in deep embedding learning," in *Proc. IEEE Int. Conf. Comput. Vis.*, Oct. 2017, pp. 2840–2848.

[24] M. Sadat, A. B. Bener, and A. Miranskyy, "Rediscovery datasets: Connecting duplicate reports," in *Proc. 14th Int. Conf. Mining Softw. Repositories*, May 2017, pp. 527–530.
[25] C. Sun, D. Lo, X. Wang, J. Jiang, and S.-C. Khoo, "A discriminative model approach for accurate duplicate bug report retrieval," in *Proc. 32nd ACM/IEEE Int. Conf. Softw. Eng.*, vol. 1, May 2010, pp. 45–54.
[26] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun, "An approach to detecting duplicate bug reports using natural language and execution information," in *Proc. 30th Int. Conf. Softw. Eng.*, 2008, pp. 461–470.
[27] N. Jalbert and W. Weimer, "Automated duplicate detection for bug tracking systems," in *Proc. IEEE Int. Conf. Dependable Syst. Netw. With FTCS DCC (DSN)*, Jun. 2008, pp. 52–61.
[28] K. Aggarwal, T. Rutgers, F. Timbers, A. Hindle, R. Greiner, and E. Stroulia, "Detecting duplicate bug reports with software engineering domain knowledge," in *Proc. IEEE 22nd Int. Conf. Softw. Anal., Evol., Reeng. (SANER)*, Mar. 2015, pp. 211–220.
[29] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in Neural Information Processing Systems*, vol. 26, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds. Red Hook, NY, USA: Curran Associates, 2013, pp. 3111–3119.
[30] J. Pennington, R. Socher, and C. Manning, "Glove: Global vectors for word representation," in *Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP)*, 2014, pp. 1532–1543.
[31] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep Learning*, vol. 1. Cambridge, MA, USA: MIT Press, 2016.
[32] P. Terdchanakul, H. Hata, P. Phannachitta, and K. Matsumoto, "Bug or not? Bug report classification using N-Gram IDF," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol. (ICSME)*, Sep. 2017, pp. 534–538.
[33] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 5998–6008.
[34] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," 2014, *arXiv:1409.0473*. [Online]. Available: http://arxiv.org/abs/1409.0473
[35] M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," 2015, *arXiv:1508.04025*. [Online]. Available: http://arxiv.org/abs/1508.04025
[36] J. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho, and Y. Bengio, "Attention-based models for speech recognition," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 577–585.
[37] N. K. Tran and C. Niedereée, "Multihop attention networks for question answer matching," in *Proc. 41st Int. ACM SIGIR Conf. Res. Develop. Inf. Retr.*, Jun. 2018, pp. 325–334.
[38] W. Yin, H. Schütze, B. Xiang, and B. Zhou, "ABCNN: Attention-based convolutional neural network for modeling sentence pairs," *Trans. Assoc. Comput. Linguistics*, vol. 4, pp. 259–272, Dec. 2016.
[39] S. Jain and B. C. Wallace, "Attention is not explanation," 2019, *arXiv:1902.10186*. [Online]. Available: http://arxiv.org/abs/1902.10186
[40] W. Lan and W. Xu, "Neural network models for paraphrase identification, semantic textual similarity, natural language inference, and question answering," 2018, *arXiv:1806.04330*. [Online]. Available: http://arxiv.org/abs/1806.04330
[41] T. Young, D. Hazarika, S. Poria, and E. Cambria, "Recent trends in deep learning based natural language processing," *IEEE Comput. Intell. Mag.*, vol. 13, no. 3, pp. 55–75, Aug. 2018.
[42] Jane Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah, "Signature verification using a 'siamese' time delay neural network," in *Proc. Adv. Neural Inf. Process. Syst.*, 1994, pp. 737–744.
[43] T. Ranasinghe, C. Orasan, and R. Mitkov, "Semantic textual similarity with siamese neural networks," in *Proc. Int. Conf. Recent Adv. Natural Lang. Process. (RANLP)*, 2019, pp. 1004–1011.
[44] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence embeddings using siamese BERT-networks," 2019, *arXiv:1908.10084*. [Online]. Available: http://arxiv.org/abs/1908.10084
[45] A. Imani, A. Vakili, A. Montazer, and A. Shakery, "Deep neural networks for query expansion using word embeddings," in *Proc. Eur. Conf. Inf. Retr.* Springer, 2019, pp. 203–210.
[46] S. Chopra, R. Hadsell, and Y. LeCun, "Learning a similarity metric discriminatively, with application to face verification," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, vol. 1, Jun. 2005, pp. 539–546.
[47] Z. Wang, H. Mi, and A. Ittycheriah, "Semi-supervised clustering for short text via deep representation learning," 2016, *arXiv:1602.06797*. [Online]. Available: http://arxiv.org/abs/1602.06797
[48] M. A. Nielsen, *Neural Networks and Deep Learning*, vol. 25. San Francisco, CA, USA: Determination Press, 2015.
[49] G. Chechik, V. Sharma, U. Shalit, and S. Bengio, "Large scale online learning of image similarity through ranking," *J. Mach. Learn. Res.*, vol. 11, pp. 1109–1135, Mar. 2010.
[50] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," *CoRR*, vol. abs/1503.03832, pp. 1–10, Mar. 2015.
[51] F. Chollet. (2015). *Keras*. [Online]. Available: https://keras.io
[52] A. Lazar, S. Ritchey, and B. Sharif, "Generating duplicate bug datasets," in *Proc. 11th Work. Conf. Mining Softw. Repositories*, 2014, pp. 392–395.
[53] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. 3rd Int. Conf. Learn. Represent.*, San Diego, CA, USA, 2015, pp. 1–15. [Online]. Available: https://arxiv.org/abs/1412.6980
[54] J. Vig, "A multiscale visualization of attention in the transformer model," 2019, *arXiv:1906.05714*. [Online]. Available: http://arxiv.org/abs/1906.05714
[55] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "ALBERT: A lite BERT for self-supervised learning of language representations," 2019, *arXiv:1909.11942*. [Online]. Available: http://arxiv.org/abs/1909.11942
[56] M. Sadat, A. B. Bener, and A. V. Miranskyy, "Rediscovery datasets: Connecting duplicate reports of apache, eclipse, and KDE," presented at the 14th Int. Conf. Mining Softw. Repositories. Zenodo, Mar. 2017.

THIAGO MARQUES ROCHA received the M.Sc. degree in computer science from Federal Amazonas University, Manaus, in 2020.

He has been actively involved in projects on deep machine learning, natural language processing, and software engineering with Federal Amazonas University. He is currently certified as a Machine Learning Engineer by Udacity in 2018, further graduated in Systems Analysis and Development at Amazonas State University, in 2016. His experience is on participate in major national projects at companies, such as Samsung that encourage research on Artificial Intelligence and Mobile App Development. As a developer, he gave the Inova Apps contest in 2015 and the contest of Civic Applications TCE/AM in 2019. He is also working with the Sidia located in Manaus, developing global solutions for bug report technical group recommendation, and automatic wizard navigation on android devices within a group focused on research.

ANDRÉ LUIZ DA COSTA CARVALHO received the Ph.D. degree from the Federal University of Amazonas, Brazil, in 2012.

He is currently the Coordinator of the Software Engineering B.Sc. degree from the Federal University of Amazonas, where he is also an Adjunct Professor. During his career, he has coordinated a number of research projects with different large companies, such as Samsung, Motorola and Nokia, mostly focused in applied Machine Learning. During the first years of his career, his research focus was on improving information retrieval by applying machine learning methods. In more recent years, his research focus has shifted to the application of deep learning techniques on natural language processing tasks.