SEDAC: A CVAE-Based Data Augmentation Method for Security Bug Report Identification

Yansheng Liao

School of Computer Science and Engineering
Macau University of Science and Technology
Macau, China
liuaansang@gmail.com

Tao Zhang
School of Computer Science and Engineering
Macau University of Science and Technology
Macau, China

tazhang@must.edu.mo

Abstract—Bug tracking systems store many bug reports, some of which are related to security. Identifying those security bug reports (SBRs) may help us predict some security-related bugs and solve security issues promptly so that the project can avoid threats and attacks. However, in the real world, the ratio of security bug reports is severely low; thus, directly training a prediction model with raw data may result in inaccurate results. Faced with the massive challenge of data imbalance, many researchers in the past have attempted to use text filtering or clustering methods to minimize the proportion of non-security bug reports (NSBRs) or apply oversampling methods to synthesize SBRs to make the dataset as balanced as possible. Nevertheless, there are still two challenges to those methods: 1) They ignore longdistance contextual information. 2) They fail to generate an utterly balanced dataset. To tackle these two challenges, we propose SEDAC, a new SBR identification method that generates similar bug report vectors to solve data imbalance problems and accurately detect security bug reports. Unlike previous studies, it first converts bug reports into individual bug report vectors with distilBERT, which are based on word2vec. Then, it trains a generative model through conditional variational autoencoder (CVAE) to generate similar vectors with security labels, which makes the number of SBRs equal to NSBRs'. Finally, balanced data are used to train a security bug report classifier. To evaluate the effectiveness of our framework, we conduct it on 45,940 bug reports from Chromium and four Apache projects. The experimental results show that SEDAC outperforms all the baselines in g-measure with improvements of around 14.24%-50.10%.

Index Terms—security bug report, data imbalance, distilBERT, conditional variational auto-encoder

I. INTRODUCTION

The bug tracking system can effectively manage the entire cycle of submission, repair, and closure in the software development process, including reporting bugs, querying bug records, generating reports, etc. These bugs have different classifications and levels. Some bugs can be described as security-related bug reports, which are marked as security bug reports (SBRs), while others are marked as non-security bug reports (NSBRs). Effective use of these bug information can provide developers with security bug predictions and prevent projects from being maliciously attacked in a timely manner.

There is a traditional workflow of SBR identification, which is mainly divided into three stages:

The first stage is text representation for bug reports because the text information in bug reports is difficult to use directly in identification involving deep learning methods.

The second stage is to solve the data imbalance problem between SBRs and NSBRs. In the real world, the ratio of SBRs to NSBRs is seriously out of balance, such as bug report data from Apache Wicket¹, Ambari², Camel³, Derby⁴. This severe data imbalance must be addressed if the information from security bug reports is to be used to provide developers with accurate predictions. Mainstream works [1]–[4] continue to revolve around the data imbalance problem to do research.

The third stage is to classify the data set obtained after solving the data imbalance problem. The mainstream method is to use basic machine learning algorithms directly because it is the most straightforward, direct, effective, and efficient solution to the binary classification problem. There are also some researchers who are trying to add deep learning methods, such as CNN, Bi-LSTM, or attention mechanisms, in the classification stage.

Several methods [1]–[4] were proposed for SBR identification, and all of them have made practical improvements, but there are still two challenges:

Challenge 1. Ignoring long-distance contextual information: Previous studies [1]–[4] mainly used tf-idf and word2vec [5]–[7] to represent bug reports. Both of these algorithms have their advantages and disadvantages. The tf-idf algorithm can avoid operations in high-dimensional spaces and obtain representations simply and quickly, but it can easily lead to vector sparsity problems. Although word2vec is a very versatile word embedding method, it makes minimal use of the contextual relationship of words.

Challenge 2. Failed to generate an utterly balanced dataset: Current approaches [1]–[4] to SBR identification primarily fall into two categories from the perspectives of filtering out NSBRs and synthesizing SBRs. The methods to filter NSBRs mainly focus on NSBRs that are the most similar to SBRs, using security-related keywords or text similarity to rank NSBRs and filter them out. The latest research uses

¹https://issues.apache.org/jira/browse/WICKET

²https://issues.apache.org/jira/browse/AMBARI

³https://issues.apache.org/jira/browse/CAMEL

⁴https://issues.apache.org/jira/browse/DERBY

k-means to cluster NSBRs and group them with SBRs. The synthesis method randomly synthesizes SBRs between every two nearest neighbors of them, using the Synthetic Minority Oversampling Technique (SMOTE) [8]. Neither filtering nor synthesis methods generate a balanced data set.

To overcome the challenges, we propose SEDAC, a SEcurity bug report identification framework composed of DistilBERT And Conditional variation autoencoder. To tackle Challenge 1, SEDAC first converts bug reports into individual bug report vectors using distilber [9], a distilled version of BERT [10], which uses Transformer [11] to obtain richer text information and capture longer-distance contextual relationships. Each vector corresponds to a one-hot encoding label. To tackle Challenge 2, we introduce conditional variation auto-encoder (CVAE) [12]. CVAE is a generative model that consists of an encoder and a decoder. We use the original vectors and corresponding labels to train and obtain the well-trained CVAE model. Then, the decoder from CVAE is separated and fed with random vectors from Gaussian distribution and specified condition vectors to generate new bug report vectors. After the balance data is obtained, it can be sent to train an SBR classifier using Logistic Regression (LR), which is proven to perform better than other machine learning algorithms (See Section V-C2).

To evaluate the performance of SEDAC, we conduct experiments on five open-source projects: four from Apache projects, each containing 1,000 bug reports, and one from Chromium projects, containing 41,940 bug reports. K-fold cross-validation method [13], [14] is adopted to avoid the impact of random seed on classifier training. The experimental results on benchmark datasets show that SEDAC outperforms the state-of-the-art method on average by 14.24%-50.10% in *g-measure* and 6.99%-45.04% in *pd*, respectively, with maximum *pf* of no more than 1%.

Overall, there are three contributions of our paper:

- **Significance.** We provide a new perspective to solve the data imbalance problem in security defect report identification, synthesizing a completely balanced data set, which benefits the identification work.
- Approach. We introduce SEDAC, a novel SBR detection framework that can automatically extract diverse bug reports and generate security bug reports to improve the performance of SBR identification.
- Experiments. We conduct experiments on five real-world projects, and empirical results prove that SEDAC is more efficient for identifying security bug reports. We have made our work publicly accessible at https://github.com/YanshengLiao/SEDAC.

The remainder of the paper is organized as follows. Section II introduces the related work of SBR identification. Section III presents the background knowledge essential for understanding our approach. Section IV illustrates our framework in detail. Section V evaluates the effectiveness of our framework. Section VI discusses the reasons for improvement and the limitations of our method. Finally, Section 7 concludes the paper.

II. RELATED WORK

Bug tracking systems collect bug reports from various sources, such as development teams, testing teams, and end users. Reporters usually have to determine whether the submitted bug reports involve security issues and mark these security-related bug reports as security bug reports (SBRs). However, in the bug reporting process, bug reporters often mislabel SBRs as NSBRs partly due to a lack of security domain knowledge. These mislabeled security bug reports can easily lead to delayed identification and remediation of vulnerabilities. Gegick et al. [15] first raised this problem and developed a text mining method to identify SBRs manually incorrectly labeled as NSBRs.

However, after solving the problem of mislabeling, it is still difficult for engineers to use past bug report information to accurately predict future bugs because, in reality, the proportion of SBRs in bug reports is deficient. As shown in Table II, among the 45,940 bug reports from Chromium and four projects in Apache, the proportion of SBRs does not exceed 10%.

To solve the data imbalance problem, Peters et al. [1] first proposed FARSEC, a framework that filters out NSBRs through security-related words and uses the idea of blending to build a prediction model. It first filters out the top 100 security-related keywords from SBRs through tf-idf and then uses them to score and rank NSBRs. NSBRs with high scores are filtered out because it means they are very similar to SBRs. Afterward, Goseva et al. [16] proposed a supervised approach and an unsupervised approach to identify security bug reports. They created feature vectors for bug reports using bf, tf, and tf-idf. Then, they combined the vectors with supervised and unsupervised classification methods. Shu et al. [2] improved FARSEC by combining SMOTE with the filters of it. Instead of filtering out NSBRs, SMOTE calculates the k nearest neighbors for each minority class sample, that is, SBRs, and oversamples to make more of them. Additionally, they optimize the hyperparameters of classification learners. In subsequent work, researchers continue to improve methods for detecting SBRs. Jiang et al. [3] proposed LTRWES, a method for filtering NSBRs based on text similarity using $BM25F_{ext}$ [17]. The method of LTRWES is similar to FARSEC, which changes the focus of filtering from security-related words to text similarity. Ma et al. [4] proposed CASMS, a method for filtering NSBRs based on k-means. CASMS divides NSBRs into groups through k-means, combining different groups with SBRs as training data. In addition, it adds a mixed model CNN-BLSTM [18], [19] and Attention mechanism [20] to enhance the text information of bug reports.

As shown in Table I, there is a summary of related works mentioned above and the proposed framework SEDAC. The supervised approach and the unsupervised approach proposed by Goseva et al. [16] are not included because their experiments were conducted on three datasets from NASA instead of the five projects used by the other four frameworks. This table summarizes that: 1) Previous studies revolve around *tf*-

TABLE I SUMMARY OF RELATED WORKS AND SEDAC.

Framework	Stage 1 Text Representation	Stage 2 Measures for Data Imbalance (Filter/Sythesize)	Stage 3 Classification
FARSEC	$tf ext{-}idf$	Security-related keywords (F)	Machine learning
SMOTUNED	$tf ext{-}idf$	SMOTE (S)	Parameter optimization for machine learning
LTRWES	$tf ext{-}idf$	Text similarity using $BM25F_{ext}$ (F)	Machine learning
CASMS	$word2vec\ \&\ tf\mbox{-}idf$	K-Means (F)	CNN-BLSTM & Attention
SEDAC	distilBERT	CVAE model (S)	Machine learning



Fig. 1. An example of a bug report from the Chromium project.

idf in the stage of text representation. 2) The primary measure to solve the data imbalance problem is to filter NSBRs using various methods. 3) Machine learning algorithms are mainly used for classification, and some researchers attempt to make innovations.

The SEDAC that we propose differs from the approaches mentioned above; distilBERT is adopted in the stage of text representation, and the CVAE model is used to synthesize SBRs.

III. BACKGROUND

A. Bug Reports

A bug report is a record written by software developers after fixing the bugs during software maintenance [21], which is generally stored in a bug-tracking system. An effective bug report should include title, environment (OS, software version, etc.), steps to reproduce a bug, expected results, actual results, etc. A good bug report can cover all essential information about the bug, help with detailed bug analysis, help developers find the correct direction and method for program debugging, and help avoid the same bugs in future versions. In the research field of software engineering, there are many research directions related to bug reports, such as bug localization [22], [23], duplicate bug report detection [24], [25], bug report

summarization [26], [27], etc. Figure 1 shows an example of a bug report from the Chromium project. The framed summary and description are the text information used in our method. Researchers with specific knowledge in software engineering can determine the type of a bug through the summary and description, such as usability, security, functionality, logic, etc. Our approach studies the identification of security bug reports.

B. Bidirectional Encoder Representations from Transformers and Its Variants

Bidirectional Encoder Representation from Transformers (BERT) [10] is a language representation model designed to pre-train deep bidirectional representations from the unlabeled text by joint conditioning on both left and right context in all layers. It consists of 12 layers of transformer blocks with a hidden size 768. Unlike traditional left-to-right or right-toleft language models, BERT can do both simultaneously. It is trained with two unsupervised tasks: Masked LM (MLM) and Next Sentence Prediction (NSP). For the MLM task, BERT randomly masks some percentage of the input tokens and then predicts those masked tokens. NSP task is to establish a predictive relationship between the previous sentence and the following one. As a pre-trained language model, it can be connected to various downstream tasks after BERT, such as text classification, question answering, etc., and then finetuned.

After BERT came out, researchers tried to make different improvements to BERT, resulting in various variants of BERT. There are some common variants, such as RoBERTa [28], ALBERT [29], distilBERT [9], distilRoBERTa, ELECTRA [30], etc. RoBERTa [28] uses a more extensive byte-level BPE dictionary at the text encoding stage, which has solved the outof-vocabulary (OOV) problem [31] and proposes a dynamic masking strategy. ALBERT [29] is a lite BERT model that proposes two methods to decrease the parameters to alleviate the memory consumption problem. DistilBERT [9] uses knowledge distillation to accelerate the pre-trained model's reasoning and reduce the model's size based on ensuring the effect of the model. DistilRoBERTa is also a distilled version of RoBERTa, following the same training procedure as distilBERT [9]. ELECTRA [30] introduces a generator and discriminator similar to the confrontation network (GAN)

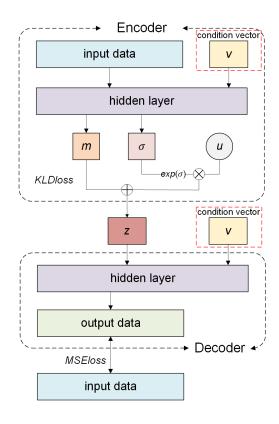


Fig. 2. The difference between VAE and CVAE.

to update model parameters more efficiently. Many studies [32], [33] have proved that all the variants mentioned have outstanding performance in text representation.

In our approach, distilBERT is applied to convert bug reports into vector representations. It performs slightly better in SEDAC than other BERT variants.

C. Conditional Variational Auto-Encoder

The core of SEDAC is to increase the number of SBRs through the data augmentation method to make the data set completely balanced. With the development of deep generative models, we consider conditional variational autoencoders (CVAE) more suitable for our framework.

Before the emergence of CVAE, there were two major generative models, namely Generative Adversarial Nets (GAN) [34] and variational autoencoder (VAE) [35]. GAN, consisting of discriminator and generator nets, is a popular neural net. It is based on game theory and aims to find the Nash Equilibrium [36] in the discriminator and generator networks. Many combinations with GAN, like DCGAN [37]), Sequence-GAN [38], LSTM-GAN [39], etc., have shown outstanding results, which also means that the discriminator and generator play a very influential role in the adversarial generative network. The generating and training modes of VAE are very different from GAN. VAE is based on Bayesian inference [35], [40], modeling the potential distribution of data and then sampling new data from the potential distribution. Latent variables can be likened to a summary description of the input data, which

summarizes what kind of data we want. To a certain extent, VAE is an expression model that specifies a standard and template for the data that must be generated through latent variables so that the model does not unthinkingly generate data. As shown in Figure 2 (contents in the red dotted box are excluded), the VAE model consists of an encoder and a decoder. The encoder samples the input data through the hidden layer and obtains the Gaussian distribution of the sample data that obeys the corresponding mean and variance.

However, whether it is GAN or VAE, their generation process is random and uncontrollable. Thus, conditional variational automatic encoding (CVAE) came into being. As illustrated in Figure 2, CVAE has one more condition vector than VAE, which refers to the label of the input data and is a form of one-hot encoding. The condition vector and the input data are combined to train the encoder and decoder. The training goal of the model is that the decoded data should be similar to the input data. Due to the existence of a condition vector, the decoder can be separated from the entire CVAE model and input random vectors and given labels to generate the data we want.

IV. APPROACH

A. Overview

The framework of SEDAC is shown in Figure 3, divided into four stages: text representation, model training, data synthesis, and SBR identification. In the first stage, SEDAC uses distilBERT to convert the text in bug reports into bug report vectors. In the second stage, SEDAC obtains the decoder for data synthesis by training the CVAE model. In the third stage, SEDAC feeds random vectors and specific labels as input data into the obtained decoder for data synthesis. The synthesized data in this stage is 1:1 balanced. In the fourth stage, the fully balanced dataset is used as input data to train the SBR identification model.

B. Text Representation

Bug reports contain rich information, such as summary, description, resolution, priority, status, etc. The textual information in the summary and description best summarizes whether a bug report is security-related. Pre-trained distilBERT can be used to encode the text. Therefore, we merge the summary and description contents and perform word embedding on them through distilBERT. The max embedding length of distilBERT is 512; thus, we uniformly adopt truncation measures for a text that exceeds 512 embedding lengths. As mentioned before (See Section III-B), distilBERT contains 768 hidden units, which means that each obtained embedding word is a tensor with the size of 1×768 . Hence, the embedding representation of a sentence containing N words is a matrix of size $N \times 768$. However, whether the matrix sizes are different due to different sentence lengths or the matrix is too heavy, it is not conducive to subsequent model training. To simplify the embedding representation, we perform row averaging on each sentence representation so each bug report can be represented as a vector with a size of 1×768 . For M bug reports, they are

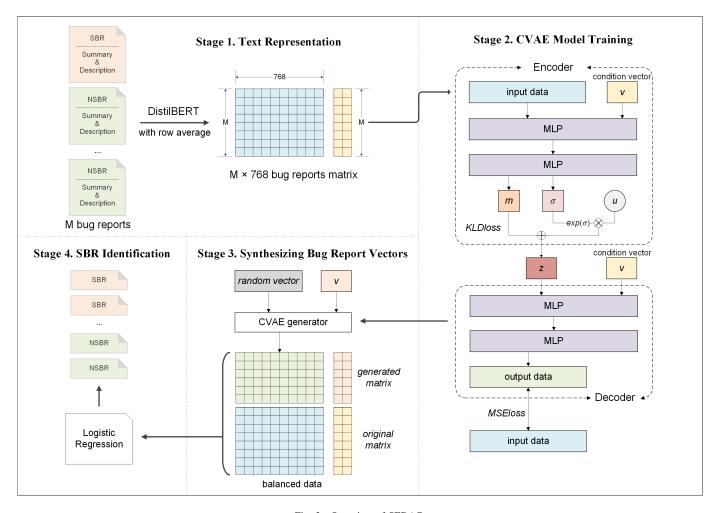


Fig. 3. Overview of SEDAC.

converted into a matrix with a size of $M \times 768$. Specifically, the labels of SBRs (i.e., the value of 1) and NSBRs (i.e., the value of 0) result in one-hot vectors [0, 1] and [1, 0], respectively.

In short, each bug report is turned into an independent text vector after processing, which is used as input data for model training, with the label as a condition vector in one-hot encoding form.

C. CVAE Model Training

As shown in Figure 3, the second part is training the CVAE model using an encoder-decoder structure. In the encoder, the bug report vectors and the corresponding one-hot condition vectors serve as input to our model. Followed by two layers of MLPs, they form a fully connected network, matching each input data with a Gaussian distribution that obeys the corresponding mean m and standard deviation σ . The u is a noise vector randomly sampled from the Gaussian distribution. The encoder generates the latent variables z by Equation 1 [41].

$$z = m + e^{\sigma} \times u \tag{1}$$

To prevent the value of e^{σ} from being infinitely close to 0, the KLDloss function is set in the sampling process. It can be calculated by Equation 2 [41].

$$KLDloss = -\sum \frac{1 + \sigma - m^2 - e^t}{2} \tag{2}$$

In the decoder, the network of two-layer MLPs is learned to generate the bug report vectors from the sampled distribution. The mean square error (MSE) ensures the generated quality of the reconstructed bug report vectors. It can be calculated by Equation 3 [41]. X(i) refers to the initial input data at the very beginning of CVAE model training. $X_{reconstruct}(i)$ refers to the output data in the decoder.

$$MSEloss = \frac{\sum_{j=1}^{768} (X(i) - X_{reconstruct}(i))^2}{768}$$
 (3)

The training process aims to ensure that the reconstructed bug report vectors are as consistent as possible with the input data by minimizing KLDloss and MSEloss.

D. Synthesizing Bug Report Vectors

The decoder plays a vital role in the data synthesis stage. It is separated from the well-trained CVAE model as a generator and synthesizes bug report vectors of specified labels.

Specifically, we can generate synthesized bug report vectors by feeding the decoder with a random vector from the Gaussian distribution and a specified condition vector. The condition vector should be [0,1] because the data we need are SBRs. Note that at this stage, the amount of generated data is calculated and is the difference between NSBRs and SBRs. It also means that the generated vectors combined with the original vectors become a fully balanced data set with SBRs: NSBRs = 1:1, which is beneficial to subsequent classification tasks.

E. SBR Identification

The generated data output from the synthesis stage is combined with the original data as the input data for SBR identification. Our identification model of choice is logistic regression, which performs better than other machine learning algorithms (See Section V-C2).

V. EXPERIMENT

Our experiment was conducted on an AMD EPYC 7742 64-core Processor with 96G RAM.

A. Datasets

We evaluate SEDAC on five real-world projects with historical bug reports that were labeled SBRs and NSBRs: four from Ohira et al. [42], and a subset of bug reports from the Chromium project, compiled by Peters et al. [1]. Four open-source Apache projects are applied in different domains and are tracked by the bug tracking system JIRA⁵. Ohira et al. [42] supply them in Microsoft Excel format, organized into comma-separated value (CSV) files by Peters et al. The Chromium dataset comes from the 2011 mining challenge of the Mining Software Repositories conference⁶ [1]. Security bugs are labeled as Bug-Security while other types of bugs are treated as non-security bug reports. We use the same datasets as baselines for a fair comparison. Table II shows each project's domain and other details. We have not updated the dataset to the latest date to provide a fair comparison with the state-of-the-art work.

B. Evaluation Metrics

To evaluate the performance of the classification models, we use the measures shown in Table III. The confusion matrix is used, where *TP*, *FP*, *FN* and *TN* are true positive, true negative, false positive, and false negative respectively:

- TP: The number of SBRs that are correctly predicted.
- *FP*: The number of NSBRs that are mistakenly predicted as SBRs.
- FN: The number of SBRs that are mistakenly predicted as NSBRs.
- *TN*: The number of NSBRs that are correctly predicted. We mainly focus on probability of detection *pd*, probability of false alarm *pf*, and *g-measure* [43]:
 - pd: the fractions of SBRS that are correctly predicted.

- pf: the ratio of the NSBRs that are incorrectly predicted as SBRs.
- g-measure: the harmonic mean of pd and (100-pf). The 100-pf is known as a concept named specificity, which is defined as the fractions of not predicting NSBRs as SBRs. It is proved that the g-measure is more suitable for unbalanced data [1], [43].

C. Research Questions and Results

- 1) RQ1.How does SEDAC perform in SBR identification compared with the state-of-the-art methods?:
- a) Baselines: In order to evaluate the effect of SEDAC, the following frameworks are selected as baselines for comparison with SEDAC:
 - **FARSEC:** FARSEC [1] uses *tf-idf* to identify security-related keywords and filter out NSBRs based on them. It finally uses the idea of blending to rank bug reports.
 - SMOTUNED: SMOTUNED [2] is an improvement of FARSEC; it combines SMOTE with FARSEC and optimizes the hyper-parameters of machine learning algorithms.
 - LTRWES: LTRWES [3] is also an improvement of FAR-SEC. It calculates text similarity based on BM25F_{ext} to filter out NSBRs that are most similar to SBRs, replacing the method in FARSEC that uses security-related keywords to filter NSBRs.
 - CASMS: CASMS [4] clusters NSBRs into different classes through k-means, and then combines different NSBR groups and SBRs into a smaller but relatively balanced data set for SBR identification. During the process, it also combines the CNN-BLSTM model and attention mechanism.

In the process of reproducing FARSEC, we could not reproduce it smoothly due to the problems with the *Clojure* language version. Similarly, when we reproduced LTRWES, we could not find some packages used in their code, thus we could not reproduce it, either. For SMOTUNED and CASMS, the code is not given in the article. Therefore, we uniformly select the results with the best g-measure performance among their results to compare with the results of our method. Since the data sets used by our method are consistent with those used by baselines, such comparison is reliable.

b) Results: As can be seen from Table IV, for the Chromium project, the best baseline result comes from LTR-WES, whose g-measure is 86.67% and pd is 93.04%. The results of SEDAC are that g-measure is 99.77% and pd is 99.54%, and the corresponding improvements are 15.11% and 6.99% respectively. For Wicket, the highest g-measure comes from SMOTUNED, which is 75.85%, while SEDAC's g-measure is 99.49%, with an improvement of 31.17%. The highest pd comes from LTRWES, 83.33%, while the pd of SEDAC is 98.99%, with an improvement of 18.79%. For Ambari, the highest g-measure and pd are both from LTRWES, which are 86.16% and 85.71% respectively, while the results of SEDAC are 98.43% and 97.12%, with improvements of 14.24% and 13.31% respectively. For Camel, all baseline

⁵https://www.atlassian.com/software/jira/

⁶http://2011.msrconf.org/msr-challenge.html

TABLE II DETAILS OF COLLECTED PROJECTS.

Project	Domain	Time Period	BRs	SBRs	SBRs(%)
Chromium	Web browser called Chrome.	08/30/2008-06/11/2010	41,940	192	0.5
Wicket	Component-based web application framework for Java programming.	10/20/2006-11/09/2014	1,000	10	1.0
Ambari	Hadoop management web UI backed by its RESTful APIs.	09/26/2011-08/08/2014	1,000	29	3.0
Camel	A rule-based routing and mediation engine.	07/08/2007-09/18/2013	1,000	32	3.0
Derby	A relational database management system.	09/28/2004-09/17/2014	1,000	88	9.0

TABLE III EVALUATION METRICS: CONFUSION MATRIX, DEFINITIONS OF pd, pf and g-measure.

		A	ctual
		SBRs	NSBRs
Predict	SBRs	TP	FN
Tredict	NSBRs	FP	TN
	pd		$\frac{\Gamma P}{+FN}$
	pf	\overline{FP}	$\frac{FP}{+TN}$
	g-measure	$2 \times pd \times$	$\frac{(100-pf)}{100-pf)}$

results perform poorly, and only the results of LTRWES are slightly better than others, which are 65.47% in *g-measure* and 66.67% in *pd* respectively. SEDAC's performance in Camel is still very good, with 98.27% in *g-measure* and 96.70% in *pd*, accompanied by improvements of 50.10% and 45.04% respectively. For Derby, the highest *g-measure* and highest *pd* are from LTRWES and CASMS, which are 75.92% and 70.00% respectively. The results of SEDAC are 95.80% and 92.6%, with improvements of 26.19% and 32.36%. For all projects, the *pf* results of SEDAC are as low as 0%, as high as 0.77%, or even no more than 1%. On average, SEDAC's results improve over the best baselines by 27.68% in *g-measure* and 21.92% in *pd*. In summary, SEDAC outperforms all baselines.

- 2) RQ2.How effective is the combination of distil-BERT(uncased) and Logistic Regression compared to other language models and machine learning algorithms?:
- a) Text representation: In the stage of text representation stage, distilBERT (uncased) is used to capture the contextual information of bug reports and convert them into vectors. It is a distilled and case-insensitive version of BERT. We select some other common BERT variants to compare with distilBERT (uncased) as follows:
 - **BERT** (**cased**): BERT [10] is an unsupervised pre-trained language representation model that stands for Bidirectional Encoder Representations from Transformers. It totalizes 110M parameters and it is case-sensitive.
 - BERT (uncased): An uncased version of BERT. It is case-insensitive.
 - **RoBERTa:** RoBERTa [28] has the same architecture as BERT but uses a byte-level BPE as a tokenizer (same as GPT-2) and uses a different pretraining scheme, including dynamic masking, training with larger batches, etc. It is case-sensitive.

- **DistilRoBERTa:** DistilRoBERTa is a distilled version of RoBERTa, which follows the same training procedure as distilBERT [9]. The model has six layers, totaling 82M parameters. On average, it is twice as fast as RoBERTabase. It is case-sensitive.
- ALBERT: ALBERT [29] presents parameter-reduction techniques to lower memory consumption and increase the training speed of BERT. There are two versions of the ALBERT base model, and version 2 is selected because it has better results in nearly all downstream tasks due to different dropout rates, additional training data, and more extended training. It is case-insensitive.
- **DistilBERT (cased):** DistilBERT [9] is a distilled version of BERT, with 40% less parameters and 60% faster than BERT, and preserves over 95% performance of BERT. It is case-sensitive.
- ELECTRA: ELECTRA [30] adjusts the pretraining framework by training two transformer models: the generator and the discriminator. It is case-insensitive.

All models are used in their base version due to lower costs with fewer parameters.

- b) Machine learning algorithms: In the stage of SBR identification, Logistic Regression is used to classify the balanced data. It is a traditional statistics technique that models the probability of a discrete outcome. Salem et al. [44] have proved their usefulness for the prediction of software failures. Some common and classic machine learning algorithms are selected to compare with it as follows:
 - Support vector machine (SVM) is a powerful supervised method for classification and regression tasks. It worked well when used by Xing et al. [45] for software quality prediction, and Ye et al. [46] for attack detection.
 - Naive Bayes (NB) is a classic supervised algorithm for classification tasks. Catal et al. [47] applied it to software fault prediction, and Ji et al. [48] made use of it for software defect detection and it showed great performance. Considering that our data is continuous, we used Gaussian NB instead of other NB methods.
 - Random Forest (RF) is a combination of multiple decision trees. Li et al. [49] adopted it on malicious software classification, and it outperformed several state-of-the-art classifiers.
 - K-Nearest Neighbors (KNN) is a non-parametric learning algorithm that uses proximity to group data and makes predictions or classifications. Liu et al. [50] proposed a

TABLE IV COMPARISON OF FARSEC, SMOTUNED, LTRWES, CASMS, AND SEDAC IN g-measure(%), pd(%), AND pf(%).

Project	Metric	FARSEC	SMOTUNED	LTRWES	CASMS	SEDAC	Improvement
	g-measure	65.40	80.53	86.67	74.00	99.77	↑15.11%
Chromium	pd	49.60	74.80	93.04	73.05	99.54	↑6.99 %
	pf	3.80	12.80	18.88	24.41	0.00	-
	g-measure	65.00	75.85	70.93	64.60	99.49	↑31.17%
Wicket	pd	66.70	66.70	83.33	73.33	98.99	↑18.79%
	pf	36.60	12.10	38.26	40.85	0.00	-
	g-measure	71.90	71.64	86.16	83.12	98.43	↑14.24%
Ambari	pd	57.10	57.10	85.71	82.85	97.12	↑13.31%
	pf	3.00	3.90	13.39	16.10	0.21	-
	g-measure	53.80	64.43	65.47	62.99	98.27	↑50.10%
Camel	pd	50.00	55.60	66.67	64.45	96.70	†45.04%
	pf	41.80	23.40	35.68	37.05	0.10	-
	g-measure	61.70	69.60	75.92	74.72	95.80	↑26.19%
Derby	pd	47.60	57.10	69.05	70.00	92.65	†32.36%
	pf	12.40	10.90	15.69	19.44	0.77	-
	g-measure	63.56	72.41	77.03	71.89	98.35	†27.68%
Average	pd	54.20	62.26	79.56	72.74	97.00	↑21.92%
C	pf	19.52	12.62	24.38	27.57	0.22	-

classification algorithm for noisy data elimination based on it, and it performed well.

 Multilayer Perceptron (MLP) is a feed forward artificial neural network. Nassif et al. [51] developed an MLP neural network model to predict software effort, showing its effectiveness.

In the experiment, we use the default parameters of the model. The main reasons are: 1) With the use of the default parameters, the model results have performed well enough (*g-measure* exceeds 95%). 2) The adjusted parameters for the model that trains on the specified data set are only applicable to that data set. Thus, the parameter adjustment is of little significance.

When dividing the training set and the test set in the early stage of training, different random seeds define different divisions of data sets, resulting in large fluctuations in the training results. In order to reduce the impact of random seeds, we adopt the k-fold cross-validation method. The values of k are generally 3, 5, and 10. Considering that for Chromium, 3 is too small, and for the other four projects, 10 may be too large, thus we selected 5 as our k value. Specifically, a data set is divided into 5 parts, and 5 training times are performed. Each data set is used as a test set in turn, and the remaining 4 parts are used as training sets. The average of the five results obtained is the final result of the model.

c) Results: As shown in Table V, we select 8 language models and 6 machine learning algorithms for experiments, resulting in a total of $8 \times 6 = 48$ sets of results. Each set of results also includes 5 data sets and three indicators. Our experimental results are too large. To express the experimental results more clearly, we take the average of each set of experimental data in data set units to obtain each set's representative results and measure which combination is more suitable for SEDAC. Observing the representative results, the g-measure

of almost all combinations is above 98%, except for the combination of ALBERT and MLP. Among all combinations, the best five are BERT (cased) and logistic regression, BERT (uncased) and logistic regression, distilBERT (cased) and logistic regression, distilBERT (uncased) and logistic regression, and BERT (uncased) and KNN, respectively, with *g-measure* exceeding 98.3%. Among these five combinations, the top three combinations with the highest *pd* are BERT (cased) and logistic regression, and distilBERT (uncased) and logistic regression, with *pd* exceeding 97%. Among them, the combination of distilBERT (uncased) and logistic regression slightly outperforms other two ones with a lower *pf*.

- 3) RQ3.How effective is the CVAE model in the data synthesizing step?:
- a) Approach: The CVAE model is the core of SEDAC. To evaluate the effectiveness of the CVAE model in SEDAC, we removed the CVAE module and classified the original imbalanced data set.
- b) Results: The results are shown in Table VI. For Chromium, g-measure is only 1.03%, and pd is only 0.53%. For Wicket, Ambari, and Camel, both g-measure and pd become 0%. For Derby, although g-measure is still 40.32%, compared with the original result of 95.80%, it still drops by nearly 60%, and pd drops by about 70%. On average, the degradation performance of SEDAC without the CVAE module is reflected in the decline rate of 91.59% in g-measure and the decline rate of 94.51% in pd. In summary, the performance of CVAE in the data synthesis step is very effective.

VI. DISCUSSION

A. Reasons for Improvement

1) BERT Variant: DistilBERT (uncased), as a variant of BERT, well inherits the function of capturing contextual

TABLE V COMPARISON OF DIFFERENT COMBINATIONS IN *g-measure*(%), pd(%), AND pf(%).

Processor Proc	Model	Daningt	5	SVM			NB			LR			RF		ŀ	KNN		1	MLP	
BERT Ambari 94,8 95,9 0.00 99,49 98,99 0.00 99,49 98,99 0.00 99,49 98,99 0.00 99,44 98,99 0.00 99,44 98,99 0.00 99,44 98,99 0.00 99,45 98,94 0.00 98,45 97,12 0.00 98,23 97,20 0.10 98,45 97,12 0.00 98,45 97,12 0.00 98,45 97,12 0.00 98,45 97,12 0.00 98,45 97,12 0.00 98,45 97,12 0.00 98,45 97,12 0.00 98,45 97,12 0.00 98,45 97,10 0.00 98,45 98,40 0.00 99,47 98,45 0.00 99,47 98,45 0.00 99,47	Model	Project	g-measure	pd	pf															
BERT Cansel Marka Mark		Chromium	99.77	99.54	0.00	99.77	99.54	0.00	99.77	99.54	0.00	99.77	99.54	0.00	99.77	99.54	0.00	99.74	99.63	0.15
Cased Cased Cased Part		Wicket	99.49	98.99	0.00	99.49	98.99	0.00	99.49	98.99	0.00	99.49	98.99	0.00	99.49	98.99	0.00	99.44	98.99	0.10
Performance	BERT	Ambari	98.48	97.01	0.00	98.48	97.01	0.00	98.49	97.22	0.21	98.43	97.01	0.10	98.54	97.12	0.00	98.23	97.22	0.72
Network Netw	(cased)	Camel	98.32	96.70	0.00	98.32	96.70	0.00	98.27	96.70	0.10	98.32	96.70	0.00	98.32	96.70	0.00	97.92	96.90	1.03
Chromium 99,77 99,54 0,00 99,77 99,54 0,00 99,77 99,54 0,00 99,77 99,54 0,00 99,78 99,63 0,07		Derby	94.90	90.35	0.00	94.90	90.35	0.00	95.59	92.54	1.10	94.92	90.46	0.11	95.22	91.56	0.77	95.35	94.41	3.62
BERT Ambari 99.49 98.99 0.00 99.44 98.89 0.00 99.44 98.89 0.00 99.49 98.99 0.00 99.49 98.99 0.01 09.94 98.99 0.01 09.44 98.99 0.01 09.44 09.05 0.04 09.45 0.04 0		Average	98.19	96.52	0.00	98.19	96.52	0.00			0.28		96.54	0.04	98.27	96.78	0.15		97.43	
BERT Camel Octame Octa		Chromium	99.77	99.54	0.00	99.77	99.54	0.00	99.77	99.55	0.01	99.77	99.54	0.00	99.77	99.54	0.00	99.78	99.63	0.07
Climbay Clim		Wicket																		
Purby 94,90 90.35 0.00 94,90 90.35 0.00 94,90 90.55 0.00 93.5 95.55 95.55 95.55 95.55 95.55 94.52 3.00 94.50 9	BERT	Ambari	98.48	97.01	0.00	98.48	97.01	0.00		97.22	0.31	98.48	97.01	0.00	98.43	97.01		97.98	97.01	1.03
Part	(uncased)																			
Chromium 99,77 99,54 0,00 99,77 99,54 0,00 99,77 99,54 0,00 99,77 99,54 0,00 99,77 99,54 0,00 99,77 99,54 0,00 99,77 99,54 0,00 99,77 99,54 0,00 99,77 99,54 0,00 99,49 98,99 0,00 99,49 98,99 0,00 99,49 98,99 0,00 99,49 98,99 0,00 99,49 98,99 0,00 99,49 98,99 0,00 99,49 98,99 0,00 99,49 98,99 0,00 99,49 98,99 0,00 99,49 98,99 0,00 98,48 97,01 0,00 98,48 97,01 0,00 98,48 97,01 0,00 98,48 97,01 0,00 98,48 97,01 0,00 98,48 97,01 0,00 98,48 97,01 0,00 98,48 97,01 0,00 98,49 0,00 98,49 0,00 99,49 98,99 0,00 99		Derby	94.90	90.35	0.00	94.90	90.35	0.00	95.65	92.65			90.57	0.11	95.49	91.89			94.52	3.40
ROBERTa Ambari A																				
ROBERTA ROBERT																				
Camel 98.32 96.70 0.00 99.32 96.70 0.00 99.34 98.99 0.00 99.34 9																				
Camel 98.32 96.70 0.00 98.22 96.7	RoBERTa																			
Average 98.19 96.52 0.00 98.19 96.52 0.00 98.19 96.52 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.78 99.67 0.10 0.00 99.49 98.99 0.00 99.77 99.54 0.00 99.																				
Chromium 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.78 99.70 0.10																				
Michael Mich																				
Ambari 98.48 97.01 0.00 98.48 97.01 0.00 98.48 97.01 0.00 98.48 97.01 0.00 98.59 97.22 0.00 97.98 97.12 1.13																				
Camel 98.32 96.70 0.00 98.32 96.70 0.00 98.32 96.70 0.00 98.32 96.70 0.00 98.32 96.70 0.00 99.34 96.80 0.00 97.70																				
Derby 04.90 90.35 0.00 94.90 90.35 0.00 95.15 90.90 0.11 94.90 90.35 0.00 95.08 91.23 0.66 95.80 94.74 3.07	distilRoBERTa																			
Average 98.19 96.52 0.00 98.19 96.52 0.00 98.24 96.63 0.02 98.19 96.52 0.00 98.25 96.74 0.13 98.19 97.46 1.05																				
Chromium 99,77 99,54 0.00 99,77 99,54 0.00 99,77 99,54 0.00 99,77 99,54 0.00 99,77 99,54 0.00 99,77 99,62 0.09																				
Albert Wicket 99.49 98.99 0.00 99.49 98.99 0.00 99.44 98.99 0.10 99.49 98.99 0.00 99.49 98.99 0.00 99.44 98.99 0.10																				
Albert Rambari 98.48 97.01 0.00 98.48 97.01 0.00 98.48 97.01 0.00 98.38 97.12 0.31 98.48 97.01 0.00 98.48 97.01 0.00 98.08 97.12 0.93 Camel 98.32 96.70 0.00 98.32 96.70 0.00 97.81 96.70 1.03 98.32 96.70 0.00 98.32 96.70 0.00 97.51 96.70 1.60 Perby 94.09 0.35 0.00 94.90 90.35 0.00 94.90 90.35 0.00 94.98 92.76 2.63 94.90 90.35 0.00 94.99 91.23 0.88 94.25 93.09 4.49 Average 98.19 96.52 0.00 98.19 96.52 0.00 98.08 97.03 0.82 98.19 96.52 0.00 98.21 96.69 0.18 97.11 1.45 Chromium 99.77 99.54 0.00 99.78 97.01 0.82 0.00 98.32 96.70 0.00 98.32 96.70 0.00 98.32 96.70 0.00 98.32 96.70 0.00 98.32 96.70 0.00 98.32 96.70 0.00 98.32 96.70 0.00 98.32 96.70 0.00 99.78 99.54 0.00 99.77 9																				
ALBERT Camel 98.32 96.70 0.00 98.32 96.70 0.00 97.81 96.70 1.03 98.32 96.70 0.00 98.32 96.70 0.00 97.51 96.70 1.65																				
Derby 94,90 90.35 0.00 94,90 90.35 0.00 94,90 90.35 0.00 94,98 92.76 2.63 94,90 90.35 0.00 94,99 91.23 0.88 94.25 93,09 4.49	ALBERT																			
Average 98.19 96.52 0.00 98.19 96.52 0.00 98.09 9.00 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.79 98.99 0.00 99.49 89.99 0.00 99.49 89.99 0.00 99.49 89.99 0.00 99.49 89.99 0.00 99.49 89.99 0.00 99.39 89.89 0.00 89.89 0.00 89.89 0.00 99.49 89.99 0.00 99.49 89.99 0.00 99.39 89.99 0.00 99.39 89.99 0.00 80.89 0.00 80																				
Chromium 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.79 98.99 0.00 99.49 98.99 0.00 99.49 98.99 0.00 99.49 98.99 0.00 99.49 98.99 0.00 99.49 98.99 0.00 99.49 98.99 0.00 99.49 98.99 0.00 99.49 98.99 0.00 99.49 98.99 0.00 99.49 98.99 0.00 99.49 98.99 0.00 99.49 98.99 0.00 99.49 98.99 0.00 99.49 98.08 97.01 0.82																				
Wicket 99.49 98.99 0.00 99.49 98.99 0.00 99.49 98.99 0.00 99.49 98.99 0.00 99.49 98.99 0.00 99.49 98.99 0.00 99.49 98.99 0.00 99.49 98.99 0.00 99.49 98.99 0.00 99.39 98.99 0.20																				
MisilBERT Ambari 98.48 97.01 0.00 98.48 97.01 0.00 98.43 97.01 0.10 98.43 97.01 0.10 98.48 97.01 0.00 98.08 97.01 0.02 98.02 96.70 0.00 98.02 96.70 0.00 98.02 96.70 0.00 98.02 96.70 0.00 98.02 96.70 0.00 98.02 96.70 0.00 98.02 96.70 0.00 98.02 96.70 0.00 98.02 96.70 0.00 98.02 96.70 0.00 98.02 96.70 0.00 98.02 96.70 0.00 98.02 96.70 0.00 98.02 96.70 0.00 98.02 96.70 0.00 98.02 96.70 0.00 98.02 96.70 0.00 98.02 96.70 0.00 97.87 97.01 1.24																				
Cased Camel 98.32 96.70 0.00 98.32 96.70 0.00 98.22 96.70 0.21 98.32 96.70 0.00 98.32 96.70 0.00 97.87 97.01 1.24	dictilREDT																			
Derby 94.90 90.35 0.00 94.90 90.35 0.00 95.86 95.54 95.86																				
Average 98.19 96.52 0.00 98.19 96.52 0.00 98.19 96.52 0.00 98.19 96.52 0.00 98.19 96.52 0.00 98.35 96.98 0.19 98.17 96.52 0.04 98.24 96.72 0.13 98.09 97.30 1.09 Chromium 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.69 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.49 98.99 0.00 99.49 98.99 0.00 99.49 98.99 0.00 99.49 98.99 0.00 99.49 98.99 0.00 99.49 98.99 0.00 99.49 98.99 0.00 99.49 98.99 0.00 99.49 98.99 0.00 99.49 98.99 0.00 99.49 98.99 0.00 99.49 98.99 0.00 98.43 97.01 0.10 98.43 97.01 0.1	(casea)																			
Chromium 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.60 0.16 Micket 99.49 98.99 0.00 99.44 98.89 0.00 99.44 98.89 0.00 99.49 98.99 0.00 98.02 96.70 0.00 98.02 96.70 0.00 98.02 96.70 0.00 98.02 98.03 97.42 1.34 98.90 0.00 99.77 99.54 0.00 99.77																				
Wicket 99.49 98.99 0.00 99.44 98.89 0.00 99.44 98.89 0.00 99.49 98.99 0.00 99.49 98.99 0.00 99.49 98.99 0.00 99.44 98.99 0.10 distilBERT Ambari 98.48 97.01 0.00 98.48 97.01 0.00 98.43 97.12 0.21 98.43 97.01 0.10 98.43 97.01 0.10 98.43 97.01 0.10 98.43 ELECTRA Mabari 98.48 97.01 0.00 98.10 99.24 98.99 0.00 99.49 98.99 0.00 99.49 98.99 0.00 99.44 98.99 0.10 GistilBERT Ambari 98.48 97.01 0.00 98.32 96.70 0.00 98.02 96.70 0.62 GistilBERT Ambari 98.48 97.01 0.00 98.10 96.50 0.00 98.35 0.00 98.85 97.00 0.22 98.16 96.52 0.06 98.22 95.11 91.56 0.99 95.04 94.63 4.50 GistilBERT Average 98.19 96.52 0.00 98.18 96.50 0.00 98.35 97.00 0.22 98.16 96.52 0.06 98.22 95.11 91.56 0.99 95.04 94.63 4.50 GistilBERT Average 98.19 96.52 0.00 98.18 96.50 0.00 98.35 97.00 0.22 98.16 96.52 0.06 98.22 96.76 0.22 98.03 97.42 1.34 GistilBERT Average 98.19 99.49 98.99 0.00 99.49 98.99 0.00 99.49 98.99 0.00 99.49 98.99 0.00 99.49 98.99 0.00 99.49 GistilBERT Average 98.19 98.99 0.00 99.49																				
distilBERT Ambari 98.48 97.01 0.00 98.48 97.01 0.00 98.43 97.12 0.21 98.43 97.01 0.10 98.43 97.01 0.10 97.88 97.12 1.34																				
Camel 98.32 96.70 0.00 98.32 96.70 0.00 98.32 96.70 0.00 98.27 96.70 0.10 98.32 96.70 0.00 98.32 96.70 0.00 98.02 96.70 0.62 Derby 94.90 90.35 0.00 94.90 90.35 0.00 95.80 92.65 0.77 94.81 90.35 0.22 95.11 91.56 0.99 95.04 94.63 4.50 Average 98.19 96.52 0.00 98.18 96.50 0.00 98.35 97.00 0.22 98.16 96.52 0.06 98.22 96.76 0.22 98.03 97.42 1.34 Chromium 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.77 Wicket 99.49 98.99 0.00 99.24 98.48 0.00 99.49 98.99 0.00 99.49 98.99 0.00 99.49 98.99 0.00 99.34 98.99 0.30 Ambari 98.48 97.01 0.00 98.11 96.29 0.00 98.48 97.01 0.00 98.48 97.01 0.00 98.48 97.01 0.00 98.32 96.70 0.00 98.32 96.70 0.00 98.32 96.70 0.00 98.32 96.70 0.00 98.32 96.70 0.00 98.32 96.70 0.00 98.34 98.99 0.01 98.32 96.70 0.00 98.32 96.70 0.00 98.32 96.70 0.00 98.32 96.70 0.00 98.32 96.70 0.00 98.34 98.91 0.01	distilBERT																			
Derby 94.90 90.35 0.00 94.90 90.35 0.00 95.80 95.80 95.80 92.65 0.77 94.81 90.35 0.22 95.11 91.56 0.99 95.04 94.63 4.50										96.70										
Chromium 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.66 0.11 Wicket 99.49 98.99 0.00 99.24 98.48 0.00 99.49 98.99 0.00 99.49 98.99 0.00 99.49 98.99 0.00 99.49 98.99 0.00 99.49 98.99 0.00 99.34 98.99 0.30 Ambari 98.48 97.01 0.00 98.11 96.29 0.00 98.48 97.01 0.00 98.17 96.80 0.41 98.90				90.35	0.00	94.90	90.35			92.65			90.35	0.22		91.56	0.99		94.63	
Chromium 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.54 0.00 99.77 99.66 0.11 Wicket 99.49 98.99 0.00 99.24 98.48 0.00 99.49 98.99 0.00 99.49 98.99 0.00 99.49 98.99 0.00 99.49 98.99 0.00 99.49 98.99 0.00 99.34 98.99 0.30 Ambari 98.48 97.01 0.00 98.11 96.29 0.00 98.48 97.01 0.00 98.17 96.80 0.41 98.90						98.18		0.00	98.35	97.00	0.22			0.06	98.22		0.22	98.03	97.42	
ELECTRA Wicket Psychology 99.49 Psychology 99.24 Psychology 98.48 Psychology 0.00 Psychology 98.99 Psychology 0.00 Psychology 98.99 Psychology 0.00 Psychology 98.99 Psychology 0.00 Psychology 99.49 Psychology 98.99 Psychology 0.00 Psychology 98.91 Psychology 99.34 Psychology 98.99 Psychology 0.00 Psychology 98.91 Psychology 98.99 Psychology 0.00 Psychology 98.91 Psychology 98.91 Psychology 98.91 Psychology 99.03 Psychology 0.01 Psychology 98.91 Psychology 98.91 Psychology 99.03 Psychology 98.91 Psychology 99.34 Psychology 98.99 Psychology 99.34 Psychology 99.35 Psychology <th< td=""><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></th<>																				
ELECTRA Ambari 98.48 97.01 0.00 98.11 96.29 0.00 98.48 97.01 0.00 98.48 97.01 0.00 98.48 97.01 0.00 98.48 97.01 0.00 98.48 97.01 0.00 98.48 97.01 0.00 98.48 97.01 0.00 98.32 96.70 0.00 98.32 96										98.99					99.49					
ELECTRA Camel 98.32 96.70 0.00 98.10 96.28 0.00 98.32 96.70 0.00 98.32 96.70 0.00 98.32 96.70 0.00 98.32 96.70 0.00 98.32 96.70 0.00 98.32 96.70 0.00 98.17 96.80 0.41 Derby 94.90 90.35 0.00 94.90 90.35 0.00 94.90 90.35 0.00 94.85 90.35 0.11 94.63 90.57 0.88 94.54 91.67 2.30	EL ECTE /																			
Derby 94.90 90.35 0.00 94.90 90.35 0.00 94.90 90.35 0.00 94.85 90.35 0.11 94.63 90.57 0.88 94.54 91.67 2.30	ELECTRA																			
		Derby		90.35	0.00	94.90		0.00		90.35	0.00			0.11	94.63	90.57		94.54	91.67	2.30
Average 98.19 96.52 0.00 98.02 96.19 0.00 98.19 96.52 0.00 98.18 96.52 0.02 98.14 96.56 0.18 98.03 96.83 0.69			98.19	96.52	0.00	98.02	96.19	0.00	98.19	96.52	0.00	98.18	96.52	0.02	98.14	96.56	0.18	98.03	96.83	0.69

TABLE VI COMPARISON OF SEDAC WITH AND WITHOUT CVAE MODULE IN g-measure(%), pd(%), AND pf(%).

Model	Project	Logistic Regression					
Model	Troject	g-measure	pd	pf			
	Chromium	99.77	99.54	0.00			
	Wicket	99.49	98.99	0.00			
distilBERT	Ambari	98.43	97.12	0.21			
(uncased)	Camel	98.27	96.70	0.10			
	Derby	95.80	92.65	0.77			
	Average	98.35	97.00	0.22			
	Chromium	1.03	0.53	0.00			
distilBERT	Wicket	0.00	0.00	0.00			
distrib Eiti	Ambari	0.00	0.00	0.00			
(uncased) w/o CVAE	Camel	0.00	0.00	0.10			
W/O CVAE	Derby	40.32	26.08	0.99			
	Average	8.27(\\dip91.59%)	5.32(\\94.51\%)	0.22			

information of the BERT model and provides an adequate representation for SEDAC. Although other BERT variants have also demonstrated their excellent embedding capabilities, distilBERT slightly surpasses other variants' average results based on their performance when used with different machine learning algorithms.

2) The CVAE Model: The CVAE model provides an adequate data augmentation method for SEDAC. This data

augmentation method is for input data with labels. It generates rare SBRs for imbalanced data sets, thereby constructing a balanced data set and improving the performance of SBR identification. Compared with other augmentation methods that do not use labels, the characteristics of the data generated by the CVAE model are more similar to the original data. Moreover, its encoder-decoder architecture is well suited for processing sequential data, such as converted bug report vectors in our approach.

3) Balanced Data: Completely balanced data is the fundamental reason for the excellent performance of various indicators in the SBR identification stage. It is also SEDAC's core competitiveness that outperforms other baselines. Previous research methods, whether filtering NSBRs or synthesizing SBRs, did not achieve a proper balance between them. Since the proportion of SBRs in the project is extremely low, filtering NSBRs is a temporary solution rather than the root cause. It leads to information loss, and the filtered data set still suffers from data imbalance. Although our method injects new data into the original data set, our classification effect is excellent because the generated security bug report vectors are very similar but different from the original ones, regardless of which machine learning algorithm is used.

B. Quality of Synthesized Data

Although SEDAC performs well, since the amount of generated data is close to the original one, we need to consider whether there is a large amount of the same data, resulting in a perfect classification effect. Therefore, we perform duplication-checking on the synthesized data and obtain the results in Table VII. Original refers to the number of bug reports in the original datasets, and Original(%) is the proportion of duplicated bug report vectors in it. Synthesized is the number of synthesized bug report vectors and Synthesized(%) is the proportion of duplicated ones in it. We can see that in the larger project Chromium, there are 121 duplicate data, accounting for only 0.29% in both the original and the generated data. In Wicket and Ambari, there is one piece of duplicate data each, accounting for about 0.1% of the original and generated data. In Camel and Derby, there is no duplicate data. Since the proportion of these repeated data is very low, we can ignore their impact on the SBR identification stage.

C. Threats to Validity

- 1) Validity of Raw Data: To make a fair comparison with other baselines, we have not updated these five datasets from Chromium and Apache projects. They were all collected and organized several years ago, and newer data sets may positively impact our framework. In addition, the number of SBRs in the four Apache projects is only 1,000. Although they represent the performance of our framework on small data sets, rich data may be more conducive to model training. In addition, the labels of these organized data sets are all manually labeled, which may need to be corrected, thus affecting the effectiveness of our framework.
- 2) Generalizability: Our training and testing process are only implemented on Chromium, Wicket, Ambari, Camel, and Derby projects. However, we encounter more projects in different fields in the real world. On the one hand, whether our approach is also practical for other projects is still being determined. On the other hand, the number of real-world data with labels that can be used for model training may be minimal. Therefore, there may be threats to the generalization ability of our model. More projects should be involved in our experiments.

VII. CONCLUSION

This paper proposes SEDAC, a CVAE-based data augmentation method for identifying security bug reports. SEDAC is a novel method that can solve the data imbalance problem in SBR identification by generating SBRs. Our key idea is to (1) use the distilBERT model to convert summary and description in bug reports into bug report vectors, (2) use the bug report vectors and corresponding labels as input data and condition vectors to train the CVAE model, (3) separate the decoder in the CVAE model and use it as a generator to generate SBR vectors, and (4) combine the generated SBR vectors with the original vectors to form a balanced data set for training SBR classifier. We evaluated SEDAC on the chromium project, and four Apache projects, and the experimental results proved that

SEDAC outperforms the baselines. In the future, we intend to apply SEDAC to more real-world projects.

REFERENCES

- [1] F. Peters, T. T. Tun, Y. Yu, and B. Nuseibeh, "Text Filtering and Ranking for Security Bug Report Prediction," *IEEE Transactions on Software Engineering*, vol. 45, no. 6, pp. 615–631, 2019.
- [2] R. Shu, T. Xia, L. Williams, and T. Menzies, "Better Security Bug Report Classification via Hyperparameter Optimization," 2019.
- [3] Y. Jiang, P. Lu, X. Su, and T. Wang, "LTRWES: A new framework for security bug report detection," *Information and Software Technology*, vol. 124, p. 106314, 2020.
- [4] X. Ma, J. Keung, Z. Yang, X. Yu, Y. Li, and H. Zhang, "CASMS: Combining clustering with attention semantic model for identifying security bug reports," *Information and Software Technology*, vol. 147, p. 106906, 2022.
- [5] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *Advances in neural information processing systems*, vol. 26, 2013.
- [6] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," arXiv preprint arXiv:1301.3781, 2013.
- [7] X. Rong, "Word2vec parameter learning explained," arXiv preprint arXiv:1411.2738, 2014.
- [8] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.
- [9] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "DistilBERT, a distilled version of BERT: Smaller, faster, cheaper and lighter," arXiv preprint arXiv:1910.01108, 2019.
- [10] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," arXiv preprint arXiv:1810.04805, 2018.
- [11] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.
- [12] K. Sohn, H. Lee, and X. Yan, "Learning structured output representation using deep conditional generative models," Advances in neural information processing systems, vol. 28, 2015.
- [13] J. D. Rodriguez, A. Perez, and J. A. Lozano, "Sensitivity analysis of k-fold cross validation in prediction error estimation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 32, no. 3, pp. 569–575, 2009.
- [14] T.-T. Wong and P.-Y. Yeh, "Reliable accuracy estimates from k-fold cross validation," *IEEE Transactions on Knowledge and Data Engineering*, vol. 32, no. 8, pp. 1586–1594, 2019.
- [15] M. Gegick, P. Rotella, and T. Xie, "Identifying security bug reports via text mining: An industrial case study," in 2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010). Cape Town: IEEE, 2010, pp. 11–20.
- [16] K. Goseva-Popstojanova and J. Tyo, "Identification of Security Related Bug Reports via Text Mining Using Supervised and Unsupervised Classification," in 2018 IEEE International Conference on Software Quality, Reliability and Security (QRS). Lisbon: IEEE, 2018, pp. 344–355.
- [17] C. Sun, D. Lo, S.-C. Khoo, and J. Jiang, "Towards more accurate retrieval of duplicate bug reports," in 2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011). IEEE, 2011, pp. 253–262.
- [18] Y. Wu and W. Li, "Automatic audio chord recognition with MIDI-trained deep feature and BLSTM-CRF sequence decoding model," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 27, no. 2, pp. 355–366, 2018.
- [19] B. Liu and S. Li, "ProtDet-CCH: Protein remote homology detection by combining long short-term memory and ranking methods," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 16, no. 4, pp. 1203–1210, 2018.
- [20] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," arXiv preprint arXiv:1409.0473, 2014.

DUPLICATED NUMBER OF VECTORS GENERATED BY SEDAC. ORIGINAL(%) IS THE PROPORTION OF DUPLICATED BUG REPORT VECTORS IN THE ORIGINAL DATA. SYNTHESIZED(%) IS THE PROPORTION OF DUPLICATED BUG REPORT VECTORS IN THE GENERATED DATA.

	Duplicated	Original	Original(%)	Synthesized	Synthesized(%)
Chromium	121	41940	0.29	41556	0.29
Wicket	1	1000	0.10	980	0.10
Ambari	1	1000	0.10	942	0.11
Camel	0	1000	0.00	936	0.00
Derby	0	1000	0.00	824	0.00

- [21] T. Zhang, H. Jiang, X. Luo, and A. T. Chan, "A literature review of research in bug resolution: Tasks, challenges and future directions," *The Computer Journal*, vol. 59, no. 5, pp. 741–773, 2016.
- [22] R. K. Saha, M. Lease, S. Khurshid, and D. E. Perry, "Improving bug localization using structured information retrieval," in 2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, 2013, pp. 345–355.
- [23] J. Zhou, H. Zhang, and D. Lo, "Where should the bugs be fixed? more accurate information retrieval-based bug localization based on bug reports," in 2012 34th International Conference on Software Engineering (ICSE). IEEE, 2012, pp. 14–24.
- [24] 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 *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, 2012, pp. 70–79.
- [25] A. Budhiraja, K. Dutta, R. Reddy, and M. Shrivastava, "DWEN: Deep word embedding network for duplicate bug report detection in software repositories," in *Proceedings of the 40th International Conference on Software Engineering: Companion Proceedings*, 2018, pp. 193–194.
- [26] S. Rastkar, G. C. Murphy, and G. Murray, "Automatic summarization of bug reports," *IEEE Transactions on Software Engineering*, vol. 40, no. 4, pp. 366–380, 2014.
- [27] S. Mani, R. Catherine, V. S. Sinha, and A. Dubey, "Ausum: Approach for unsupervised bug report summarization," in *Proceedings of the ACM* SIGSOFT 20th International Symposium on the Foundations of Software Engineering, 2012, pp. 1–11.
- [28] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," arXiv preprint arXiv:1907.11692, 2019.
- [29] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "Albert: A lite bert for self-supervised learning of language representations," arXiv preprint arXiv:1909.11942, 2019.
- [30] K. Clark, M.-T. Luong, Q. V. Le, and C. D. Manning, "Electra: Pretraining text encoders as discriminators rather than generators," arXiv preprint arXiv:2003.10555, 2020.
- [31] R. Sennrich, B. Haddow, and A. Birch, "Neural machine translation of rare words with subword units," arXiv preprint arXiv:1508.07909, 2015.
- [32] S. Zhang, H. Yu, and G. Zhu, "An emotional classification method of Chinese short comment text based on ELECTRA," *Connection Science*, vol. 34, no. 1, pp. 254–273, 2022.
- [33] W. Liao, B. Zeng, X. Yin, and P. Wei, "An improved aspect-category sentiment analysis model for text sentiment analysis based on RoBERTa," Applied Intelligence, vol. 51, pp. 3522–3533, 2021.
- [34] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," Advances in neural information processing systems, vol. 27, 2014.
- [35] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," arXiv preprint arXiv:1312.6114, 2013.
- [36] D. M. Kreps, "Nash equilibrium," in *Game Theory*. Springer, 1989, pp. 167–177.
- [37] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," arXiv preprint arXiv:1511.06434, 2015.
- [38] L. Yu, W. Zhang, J. Wang, and Y. Yu, "Seqgan: Sequence generative adversarial nets with policy gradient," in *Proceedings of the AAAI* Conference on Artificial Intelligence, vol. 31, 2017.

- [39] G. Zhu, H. Zhao, H. Liu, and H. Sun, "A novel LSTM-GAN algorithm for time series anomaly detection," in 2019 Prognostics and System Health Management Conference (PHM-Qingdao). IEEE, 2019, pp. 1–6.
- [40] T. L. Griffiths, J. B. Tenenbaum, and C. Kemp, "Bayesian inference," The Oxford handbook of thinking and reasoning, pp. 22–35, 2012.
- [41] C. Doersch, "Tutorial on variational autoencoders," arXiv preprint arXiv:1606.05908, 2016.
- [42] M. Ohira, Y. Kashiwa, Y. Yamatani, H. Yoshiyuki, Y. Maeda, N. Limsettho, K. Fujino, H. Hata, A. Ihara, and K. Matsumoto, "A dataset of high impact bugs: Manually-classified issue reports," in 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories. IEEE, 2015, pp. 518–521.
- [43] Y. Jiang, B. Cukic, and Y. Ma, "Techniques for evaluating fault prediction models," *Empirical Software Engineering*, vol. 13, pp. 561–595, 2008.
- [44] A. M. Salem, K. Rekab, and J. A. Whittaker, "Prediction of software failures through logistic regression," *Information and Software Technol*ogy, vol. 46, no. 12, pp. 781–789, 2004.
- [45] Fei Xing, Ping Guo, and M. Lyu, "A Novel Method for Early Software Quality Prediction Based on Support Vector Machine," in 16th IEEE International Symposium on Software Reliability Engineering (ISSRE'05). Chicago, IL, USA: IEEE, 2005, pp. 213–222.
- [46] J. Ye, X. Cheng, J. Zhu, L. Feng, and L. Song, "A DDoS Attack Detection Method Based on SVM in Software Defined Network," Security and Communication Networks, vol. 2018, pp. 1–8, 2018.
- [47] C. Catal, U. Sevim, and B. Diri, "Practical development of an Eclipse-based software fault prediction tool using Naive Bayes algorithm," *Expert Systems with Applications*, vol. 38, no. 3, pp. 2347–2353, 2011.
- [48] H. Ji, S. Huang, Y. Wu, Z. Hui, and C. Zheng, "A new weighted naive Bayes method based on information diffusion for software defect prediction," *Software Quality Journal*, vol. 27, no. 3, pp. 923–968, 2019.
- [49] F.-Q. Li, S.-L. Wang, A. W.-C. Liew, W. Ding, and G.-S. Liu, "Large-Scale Malicious Software Classification With Fuzzified Features and Boosted Fuzzy Random Forest," *IEEE Transactions on Fuzzy Systems*, vol. 29, no. 11, pp. 3205–3218, 2021.
- [50] H. Liu and S. Zhang, "Noisy data elimination using mutual k-nearest neighbor for classification mining," *Journal of Systems and Software*, vol. 85, no. 5, pp. 1067–1074, 2012.
- [51] A. B. Nassif, D. Ho, and L. F. Capretz, "Towards an early software estimation using log-linear regression and a multilayer perceptron model," *Journal of Systems and Software*, vol. 86, no. 1, pp. 144–160, 2013.