Neural Computing and Applications (2022) 34:15441–15455
https://doi.org/10.1007/s00521-022-07248-8

ORIGINAL ARTICLE

A novel approach of data race detection based on CNN-BiLSTM hybrid neural network

Yang Zhang[1] · Jiali Yan[1] · Liu Qiao[1] · Hongbin Gao[1]

Received: 22 July 2021 / Accepted: 29 March 2022 / Published online: 22 April 2022

Abstract
Existing data race detection approaches based on deep learning are suffering from the problems of unique feature extraction and low accuracy. To this end, this paper proposes a novel approach called *SmartRace* based on a CNN-BiLSTM hybrid neural network to detect data race. To build the dataset, *SmartRace* selects 25 real-world applications from five benchmark suites and then extracts multi-level features by a static analysis tool *Wala*. A tool ConRacer is employed to help label the samples. *SmartRace* leverages the Kmeans-SMOTE algorithm to make the positive samples and negative counterparts well distributed. The samples are vectorized and fed into a deep learning model. Finally, a CNN-BiLSTM hybrid neural network is built and trained to detect data race. The experimental results show that the accuracy of *SmartRace* is up to 4.94% higher than that of the existing detection approach. Furthermore, we compare *SmartRace* with the existing dynamic and static detection tools, demonstrating the effectiveness of *SmartRace*.

Keywords Data race · Concurrent program · Deep learning · Feature extraction · CNN-BiLSTM model

1 Introduction

Multithreaded programming becomes increasingly popular and is constantly leveraged by more and more programmers with the prevalence of multicore processors. Although it brings many advantages (e.g., faster execution time and higher coefficient), concurrent programs are suffering from several defects. A data race is one of the most serious defects in the concurrent program. A data race occurs when more than two threads access the same memory location without being ordered by a synchronization operation and at least one access operation is a write. It usually exists in a specific scenario and remains a potential threat, which can lead to significant damage to real-world applications. Therefore, there is an urgent need to detect data race as early as possible.

✉ Yang Zhang
zhangyang@hebust.edu.cn

✉ Liu Qiao
13663126977@163.com

[1] School of Information Science and Engineering, Hebei University of Science and Technology, Shijiazhuang 050018, Hebei, China

Most existing works employ dynamic analysis, static analysis, and hybrid analysis approaches [1–3] to detect data race. The dynamic analysis approach tends to infer data races by analyzing the access events and synchronization patterns in a program execution trace. The advantage of dynamic analysis is the low false-positive rate, whereas the disadvantage is the high false-negative rate and the high overhead. The existing dynamic detection tools include Said [4], RVPredict [5], and SlimFast [6], etc.

Unlike the dynamic analysis approach, data race detection based on static analysis is performed by analyzing read and write access to variables with the help of various static program analysis techniques (e.g., happens-before analysis, alias analysis, escape analysis, etc.) at source code or an intermediate representation. It owns the advantages of lower overhead, more comprehensive detection, and lower false negatives, while it suffers from the disadvantage of false negatives. The existing static detection tools based on static analysis include RELAY [7], Elmas [8], and SRD [9]. Other researchers try to combine both dynamic and static analysis to improve the overall efficiency of data race detection, such as RaceTracker [10] and HistLock + [11].

With the development of machine learning and the prevalence of deep learning in recent years, some

researchers have employed these promising techniques for concurrency defect detection. However, barely a few types of research have been conducted on data race detection. For example, Yu et al. [12] predicted concurrency defects by proposing static and dynamic features to judge whether a method has defects or not. Li et al. [13] proposed a framework called Defect Prediction to leverage Convolutional Neural Network (CNN) for feature generation. Ali et al. [14] proposed a data race detection method based on CNN. However, they barely extracted file-level features to build a data race training dataset, demonstrating that the accuracy of the detection is around 85%.

Although many promising techniques are proposed, several problems still exist and remain unsolved. Firstly, samples in the dataset are not collected from large-scale real-world applications and may not be representative. Secondly, the features extracted from these applications are relatively homogeneous. Without multiple features, we cannot train a CNN with high accuracy. Furthermore, without considering the generation conditions of a data race, we cannot extract enough features to detect data race. Thirdly, existing approaches mainly rely on CNN which needs further optimization on the models. Finally, the accuracy of existing approaches is not satisfying and is possible to conduct further improvements.

To solve these problems, this paper proposes a new method of data race detection based on CNN-BiLSTM hybrid neural network called *SmartRace*. Firstly, the approach extracts multi-level features (instruction-level, method-level, and file-levels) from several real-world applications via the software analysis tool WALA [15] to build the training set, followed by the ConRacer [16] tool for judging the real data race before labeling the samples. Secondly, the labeled samples are vectorized by the embedding layer of Keras, and the Kmeans-SMOTE [17] algorithm is used to distribute the samples balanced. Finally, a deep neural network of CNN-BiLSTM [18] is constructed and trained to detect data race. In the experimentation, 25 benchmarks selected from Dacapo [19], IBM Contest [20], JBench [21], JGF [22] and PJBench [23] are used to detect data race. The experimental results show that the accuracy of *SmartRace* is ranging from 90.91 to 99.69%, which is improved by 9.91–13.69% compared with the existing deep learning-based detection method. In addition, we compare *SmartRace* with the CNN-BiLSTM hybrid neural network used by *SmartRace* compared with the deep neural networks such as RNN, GRU, CNN, LSTM, and BiLSTM, which have higher accuracy. *SmartRace* is compared with the existing data race detection tools Said [4], RVPredict [5], and SRD [9], and the results show that *SmartRace* can detect more data races.

The main contributions are summarized as follows:

- We construct a dataset of data race by extracting multiple-level features from 25 real-world applications.
- We propose a novel method of data race detection based on CNN-BiLSTM hybrid neural network which uses the convolutional kernel of CNN to get relevant features and extracts bidirectional timing features by BiLSTM.
- We compare *SmartRace* with deep neural networks (such as RNN, GRU, CNN, LSTM, and BiLSTM) via cross-validation and statistical test.
- We compare *SmartRace* with the existing data race detection tools (such as Said, RVPredict, and SRD), demonstrating the effectiveness of *SmartRace*.

The rest of the paper is organized as follows. Section 2 examines the related literature on data race detection. An overview of *SmartRace* and the design of each component are presented in Sect. 3. Section 4 conducts experimentation to validate the effectiveness of *SmartRace*. Finally, the conclusion is drawn in Sect. 5.

2 Related work

Data race detection is one of the hot topics in the field of concurrent defects. Many methods such as program analysis, machine learning, and deep learning are used.

The detection method based on program analysis is classified into dynamic detection, static detection, and hybrid detection methods. Dynamic detection obtains the runtime state by program analyses such as program instruments while running the source program. Said et al. [4] proposed a symbolic analysis method based on an SMT solver that can analyze thread scheduling effectively and locate data race precisely. RVPredict [5] adds abstract control flow information to the execution model with causal analysis and detects data races by using satisfiability modulo theories solvers. SlimFast [6] is a sound race detector that exploits the novel invariance of dynamic race detection to reduce data redundancy, memory usage, and runtime overhead. Li et al. [24] proposed a dynamic contention hybrid detection algorithm *AsampleLock* based on lock mode and FastTrack algorithm using cross-thread sampling technique and using mapping to record read and write operations on shared variables to reduce false-positive and false-negative.

The static race detectors try to detect data races via analyzing source code without executing the code. RELAY [7] is a static data race detection tool based on flow-sensitive and inter-process analysis. Elmas [8] is a detection method proposed based on model detection theory, which analyzes the lock operation paths in a program and filters the results by the happen-before relationship. SRD [9]

employs program slicing techniques to determine statically happens-before relationships between access events and combines them with static analysis techniques (e.g., alias analysis) to detect data races. Gao et al. [25] proposed a static detection technique GUARD to detect possible data race sub-paths based on value flow analysis and to use MHP analysis to screen out infeasible data race paths.

The hybrid detection approach first finds all possible data races by static analysis and then leverages dynamic analysis to identify the real data race. RaceTracker [10] adopts both dynamic and static analysis methods. It first employs the static analysis to generate potential races and then performs some instrumentation operations at the location of the potential race to identify data races. HistLock + [11] is built based on thread epochs and lock release events to infer whether two operations on the same memory location have any lock subset relationship without performing expensive lock set comparisons.

Some researchers leverage machine learning and deep learning models to detect concurrency defects, including data race. Yu et al. [12] proposed features for concurrency defect prediction. They leveraged static and dynamic features to predict whether a method has defects or not. The prediction task was conducted by using Machine Learning models, such as Bayesian Network, Decision Tree, Logistic Regression and Random Forest. Li et al. [13] proposed a framework called Defect Prediction based on CNN for effective feature generation. Ali et al. [14] proposed DeepRace, a deep learning-based data race detection tool. The specific data race type is generated by mutation analysis [26], followed by constructing the dataset by walking through the AST of each source file. Finally, the vectorized values are fed into CNN for training. Their detection accuracy is around 85%, which shows that the accuracy still needs to be further improved. Sun et al. [27] proposed a machine learning-based data race detection method to detect instruction-level data race. However, they only extracted only a few features and did not extract features on the semantic aspects. Furthermore, the size of the selected programs is too small. A significant improvement has been achieved for the hybrid model. Ali et al. proposed a dynamic deep hybrid spatio-temporal model DHSTNet [28, 29] to predict inflow and outflow throughout a city or region. DHSTNet handles dynamically spatial and temporal correlations through the CNN and LSTM models based on spatial and temporal correlation.

3 Deep-learning-based data race detection

This section introduces the detection framework of *SmartRace* and then describes each component of the framework in detail.

3.1 Detection framework

The framework of *SmartRace* is shown in Fig. 1. Firstly, to construct the training dataset, we select 25 concurrent applications with data race from Dacapo [19], IBM Contest [20], JBench [21], JGF [22], and PJBench [23]. We extract the features and construct training and test samples by the static analysis tool WALA [15], and label the samples using ConRacer [16]. Secondly, to make the textual feature processed by deep learning models, the embedding layer of Keras [30] is used to vectorize the text. Considering that the number of positive samples is relatively small and further results in the imbalanced distribution of positive and negative samples, we use the data enhancement algorithm Kmeans-SMOTE [17] to increase the number of positive samples. Finally, we build a deep neural network model based on CNN and BiLSTM. With the training dataset, we obtain a well-trained classifier for data race detection.

3.2 Pseudo-code of SmartRace

We describe the process of SmartRace in the form of Pseudo-code in Fig. 2. Benchmarks are taken as the input. We create C_{set} to collect the feature information for detecting data race (Line 1). These features of the code are extracted by walking through the class and the methods, and are obtained the access operations by static analysis tool WALA. Next, the feature f is extracted from the access operations and is added to the C_{set} (Lines 2–8). The data race analysis tool *ConRacer* is leveraged to generate our Samples (Lines 9–10). We use the embedding layer in Keras to transform those text features into vectors (Line 11). The Kmeans-Smote oversampling method is used for data enhancement (Line 12). The Dataset is divided into training and testing sets (Lines 13–14). It is utilized for the initialization of the neural network model parameters (Lines 15–17). We applied TensorFlow to build the network. A CNN model is composed of four convolutional layers, and each convolutional layer is followed by a max pooling layer to remove redundant information (Lines 18–25). A BiLSTM layer is added as a temporal feature extraction model (Lines 26–27). The concatenation function is used to concatenate the features extracted by CNN and BiLSTM (Line 28). Fully connected layers are considered as classifiers for data race classification prediction (Lines 29–31). Finally, the training and testing sets are placed into the CNN-BiLSTM hybrid neural network model and adopted k-fold cross-validation ($k = 6$) for training and testing (Lines 32–37).

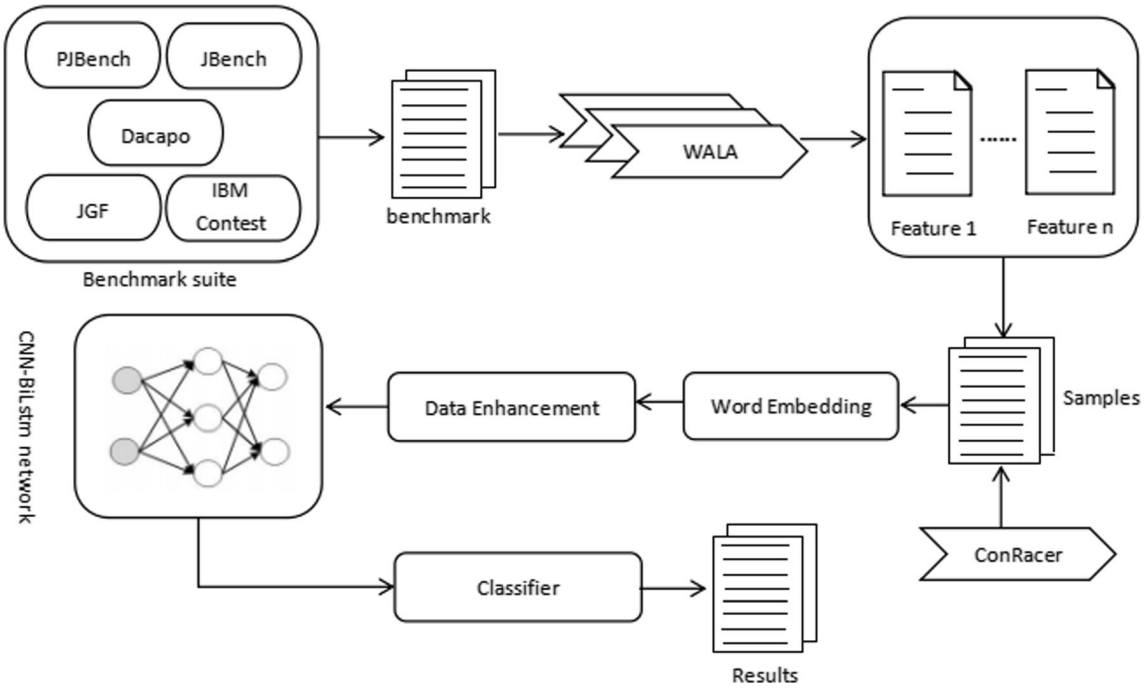

Fig. 1 Framework of *SmartRace*

Input: Benchmarks with data race
Output: Data race

Step1:Samples generation

1. $C_{set} \leftarrow \varphi$
2. for each class c in Benchmarks
3. for each method in c
4. Acquire access operations by Wala
5. Extract features f by Wala
6. Insert f into C_{set}
7. end for
8. end for
9. Samples←ConRacer(C_{set})
10. return Samples

Step2:Data Pre-Processing

11. Dataset←Vectorized(Samples)
12. Kmeans-Smote(Dataset)
13. (X_{train}, X_{test}, Y_{train}, Y_{test})←Dataset
14. return Dataset

Step3:CNN-BiLSTM model training and testing

//Initialize the parameters
15. n_classes←2, epochs←50, loss←categorical_crossentropy
16. optimizer←adam, padding←same, units←128, rate←0.5
17. kernel_initializer←glorot, axis←1, shuffle←true

//CNN layer
18. Add Convolution2D(filter,kernel_size,padding, kernel_initializer)
19. Add MaxPooling2D(pooling_size, strides, padding)
20. Add Convolution2D(filter,kernel_size, padding,kernel_initializer)
21. Add MaxPooling2D(pooling_size, strides, padding)
22. Add Convolution2D(filter,kernel_size, padding)
23. Add MaxPooling2D(pooling_size, strides, padding)
24. Add Convolution2D(filter, kernel_size, padding)
25. Add GlobalMaxPooling2D( )

//BiLSTM layer
26. Add Bidirectional(LSTM(units, input_shape, activation))
27. Add Dense(shape, activation)

//Feature fusion
28. Concatenate((CNN, BiLSTM), axis)

//Prediction
29. Add Dense(shape)
30. Add Dropout(rate)
31. Add Dense(n_classes, activation)

//Training and Testing
32. model.compile(loss, optimizer)
33. for each i in epochs
34. model .fit(X_{train}, Y_{train}, epochs, shuffle)
35. end for
36. Results←models.evaluate(X_{test}, Y_{test}, batch_size)
37. return Results

Fig. 2 Pseudo-code of SmartRace

3.3 Selecting real-world applications

Since no public dataset is available for data race detection, we first construct a dataset to train a deep neural network for data race detection.

Twenty-five benchmarks with data race are selected from benchmark suites Dacapo, IBM Contest, JBench, JGF, and PJBench. We extract 19,949 samples from 16 of 25 benchmarks as the training dataset of *SmartRace*, and 2053 data samples from 9 benchmarks as the test dataset. The configuration of these benchmarks is described in detail in Sect. 4.2.

3.4 Features extraction

During constructing the dataset, we consider the occurrence conditions of data race: (1) Threads access the same memory position simultaneously. (2) At least one operation is a write. (3) The operations are protected by a lock with the same monitor object.

Based on these conditions, we construct the dataset by selecting the features including the instructions (such as the hash value of the instruction, whether it is a write operation, whether it is included by the synchronization block or synchronization method) and the location where data race occurs (such as the package name, class name, method name, and variable name). Where instruction-related information is used to indicate the conditions under which the data race is generated, and location-related information is used to indicate the location where the data race occurs.

To obtain these features from benchmarks, *SmartRace* leverages WALA [15] to extract features. Firstly, *SmartRace* uses the method makeNCFABuilder() of WALA to build a control flow graph (CG). Secondly, it traverses CG to collect all access operations under CGNode. Then, it obtains the instructions in each access operation and determines whether the instructions are written operations and whether they are protected by a synchronized block or synchronized method. Also, it extracts the hash value as the unique identification of variable access. Finally, the location of each data race is indicated by obtaining the package name, class name, method name, and variable name (including both static and instance variables).

Table 1 presents the numerical features in the Account benchmark of the IBM Contest benchmark suite [20], where "1" in the "Read/Write" column represents a write operation and "0" represents a read operation, "1" in the "Label" column represents that there is data race while "0" represents no data race. The other columns have the value "1" for yes and the value "0" for no. Each sample is composed of two access operations and includes four instruction-level features such as read/write access, hash value, protected by synchronization method or block.

The textual features of some samples in the Account benchmark are illustrated in Table 2 where "1" in the "Label" column represents the occurrence of data race and "0" represents no data race. Each sample consists of two access operations each of which contains four textual features such as package name, class name, method name, and variable name. Among these features, package name and class name are considered as file-level features, while method name and variable name belong to method-level features.

We use a context-sensitive analysis tool ConRacer [16] to judge whether data race occurs, and further label the sample. ConRacer can find data race effectively by considering the context of function calls. As a result, few false positives and false negatives are reported. Therefore, we choose ConRacer as the auxiliary tool of *SmartRace*. To remove false positives and false negatives in ConRacer and enhance the accuracy of the dataset, we manually verified the labeled samples.

3.5 Textual feature vectorization

For the deep learning model, the textual feature is hardly processed since it can only take numerical features as input. To this end, we must transform the textual features into numerical vectors.

For textual feature vectorization, *SmartRace* uses the embedding layer of Keras [30] which is a supervised approach to learn and update weights based on labeled information. Its definition can be formulated as:

$$f : N \rightarrow R_{\mathrm{n}} \tag{1}$$

where N represents the integer encoding of the word, R_{n} represents the n-dimensional vector, and f represents a mapping from words to n-dimensional vectors.

Figure 3 demonstrates the process of textual feature vectorization by taking textual feature information in the Account benchmark as an example. The dimension of vector n is 8. Firstly, all the words in the word list are extracted for word frequency statistics and integer encoding. The conversion from words to numeric vectors is case-insensitive. For example, the word "Account" and "account" are coded as 53. The coding of "go", "out", "num", "Service", and "Bank_Total" are 60, 239, 189, 581, and 554, respectively. Each word in the sample is represented by an integer code in this case. Secondly, the numerical encoding is transformed into a word vector matrix M5000 $\times$ 8 by embedding layer processing, where the row size is the number of words and the column size is the dimensionality of the word vector. Finally, the mean

Table 1 Numerical feature

No	Access operation 1				Access operation 2				Label
	Read/ Write	Hash value	Protected by synchronized method	Protected by synchronized block	Read/ Write	Hash value	Protected by synchronized method	Protected by synchronized block	
1	0	1,055,461,584	0	0	0	952,975,668	0	0	0
2	0	1,055,461,584	0	0	0	1,656,402,084	0	0	0
3	1	1,578,914,784	0	0	1	1,578,914,784	0	0	1

Table 2 Text feature

Serial number	Access operation 1				Access operation 2				Label
	Package Name	Class Name	Method Name	Variable Name	Package Name	Class Name	Method Name	Variable Name	
1	Account	Account	go	out	account	Account	go	num	0
2	Account	Account	go	out	account	Account	go	out	0
3	Account	Account	Service	Bank_Total	account	Account	Service	Bank_Total	1

value is calculated so that each word is represented by only one value.

3.6 Balanced distribution of sample

We adopt the oversampling method to achieve a balanced sample distribution, which can both ensure the integrity of feature information and contribute to the expansion of training samples to improve the model accuracy. Although 25 concurrent applications with data race are selected during sample extraction, the number of samples with label "1" is far less than the number of the sample with label "0" in the generated dataset, which leads to an imbalance between distribution positive and negative samples. Using this dataset for training can seriously decrease the accuracy of the deep learning model. To avoid imbalanced distribution, the undersampling and oversampling methods are usually used. The undersampling method makes data balance by reducing the number of training instances from the majority of samples. The oversampling method increases the number of training instances by analyzing a few samples, which helps in increasing the number of positive samples for a data race.

The training dataset is expanded from the original 19,949 up to 39,304 by the Kmeans-SMOTE algorithm. After expansion, the distribution of positive and negative samples is balanced.

3.7 CNN-BiLSTM neural network

We implement a neural network CNN-BiLSTM in Keras. By combining both CNN and Bi-directional Long Short-Term Memory (BiLSTM), we enhance the accuracy of detection for data race detection. The semantic features of the text are extracted by the convolutional layer in CNN and refined by the pooling layer to retain the representative information in the data race. However, the presence of convolutional kernels limits the long-term dependency problem of CNN in processing time-series signal data. BiLSTM can not only retain longer effective memory information but also better capture bidirectional semantic dependencies.

The architecture of *SmartRace* deep learning model is presented in Fig. 4. To train the network, feature information for each pair of access operations is fed into the CNN-BiLSTM neural network. The feature input of *SmartRace* is defined as:

$$\text{Input} = <d_1, d_2, d_3, d_4, m_1, m_2, f_1, f_2> \quad (2)$$

where $d_i (1 \leq i \leq 4)$ represents the ith instruction-level feature, m_1 and m_2 represent method-level features, f_1 and f_2 represent file-level features.

The CNN contains four convolutional layers and four max-pooling layers. The BiLSTM consists of a BiLSTM layer and a dense layer. The function *concatenate* is used to merge both the output of the convolutional feature in CNN and the temporal features extracted by BiLSTM into the dense layer, and to prevent overfitting by the function

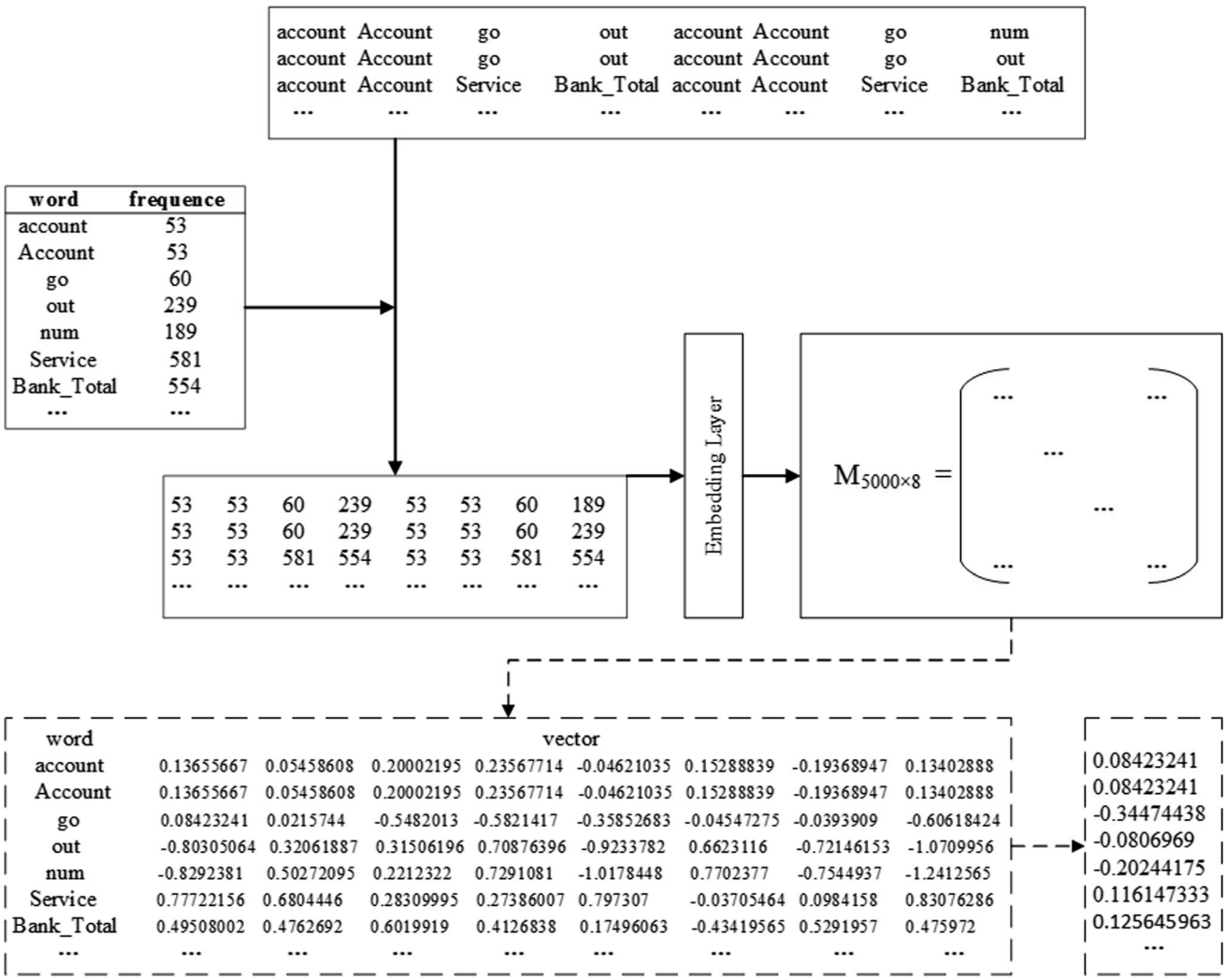

Fig. 3 Text feature vectorization

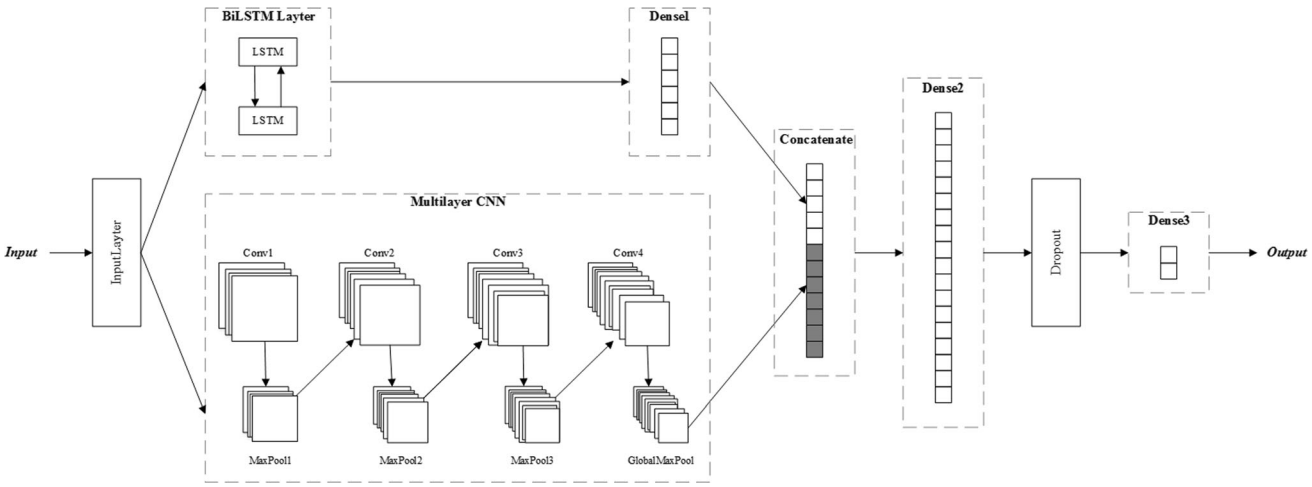

Fig. 4 CNN-BiLSTM neural network model

dropout. All layers use *Relu* as the activation function except for the final layer which uses the activation function *Sigmoid*. The BiLSTM layer uses Tanh as the activation function. We will discuss the details of parameter settings in Sect. 4.4.

4 Evaluation

This section first introduces the experimental setup, then evaluates the effectiveness of *SmartRace*. Finally, *SmartRace* is compared with traditional data race detection tools based on dynamic and static analysis.

4.1 Experimental setup

The evaluation is conducted on a Dell Z820 workstation with a 3.2 GHz Intel Xeon CPU and 8 GB main memory. The operating system is 64-bit Windows 7. Python 3.7 and Tensorflow 1.9 are used as the runtime environment for CNN-BiLSTM hybrid neural network. Eclipse 4.5.1 and JDK 1.8.0_31 are used as a running platform for WALA.

4.2 Dataset

We extract samples from 16 benchmarks from the Dacapo [19], IBM Contest [20], JBench [21], JGF [22], and PJBench [23] benchmark suites, as training datasets for *SmartRace*. Table 3 lists the configurations of these benchmarks. The number of samples is expanded from 19,949 to 39,304. Benchmarks Lusearch and Sunflow are selected from the Dacapo benchmark suite. We extract a maximum of the number of training samples. The number of samples is expanded from 5683 to 11,336 for the Lusearch benchmark, and from 7891 to 15,720 for the Sunflow benchmark. For benchmark *SimpleExample*, only 13 samples are extracted initially, which is the minimum number of training samples among all benchmarks. The number of training samples is expanded to 22 by the Kmeans-SMOTE algorithm. Other benchmarks own the number of training samples ranging from 36 to 4618 after expansion.

We validate the performance of *SmartRace* on 9 benchmarks such as Account, AirlineTickets, and Boundedbuffer. The LOC and the number of samples are listed in Table 4. Benchmarks *Boundedbuffer* extracts a total of 961 test samples, which owns a maximum of testing samples among the nine benchmarks. Benchmark *Critical* extracts only 11 test samples, which has the smallest number of testing samples. The number of samples in the other benchmarks ranges from 25 to 477.

4.3 Metrics

We evaluate *SmartRace* against existing detection approaches by calculating accuracy, precision, recall, and F_1. These metrics can be computed as follows.

Table 3 Training dataset

Benchmarks	Functional description	LOCs	Original dataset	Expanded dataset
Animator	Interpreter command set algorithm	1397	161	308
Elevator	Elevator scheduling algorithm	1155	960	1850
JBench_Bench	Java concurrent vulnerability detection program	37	83	162
Lufact	LU factor decomposition algorithm	806	326	326
Lusearch	Text Search Tools	49,785	5683	11,336
Rax	Multithreading algorithm	52	23	36
RayTracer	Ray tracing program	985	789	1560
Readerswriters	Reader–Writer algorithm	285	193	382
ReplicatedCaseStudies	Multithreading algorithm	890	424	844
Series	Fourier coefficient analysis algorithm	476	796	1588
SimpleExample	Multithreading case algorithm	23	13	22
SOR	Successive over-relaxation method	499	95	184
Sunflow	Ray tracing rendering image program	25,118	7891	15,720
TestRace	Simple multithreading algorithm	217	128	238
Tsp	Traveling salesman problem solving algorithm	450	2315	4618
Weblech	Java web download tool	1464	69	130
Total		83,639	19,949	39,304

Table 4 Testing dataset

Benchmarks	Functional description	LOCs	Number of test samples
Account	Account counting system	87	143
AirlineTickets	Airline ticket sales system	83	139
Bubblesort	Bubble sort algorithm	274	177
BoundedBuffer	Producer–Consumer algorithm	334	961
Bufwriter	Multi-threaded buffer pool simulation program	199	88
Critical	Two-threaded environment simulation program	63	11
Mergesort	Summarization algorithm	298	25
Montecarlo	Financial simulation program	1501	477
PingPong	Table tennis game simulation program	124	32
Total		2963	2053

$$\text{Accuracy} = \frac{\text{TN} + \text{TP}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{3}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{4}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{5}$$

$$F_1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{6}$$

where TP, TN, FP, and FN represent the numbers of true positives, true negatives, false positives, and false negatives, respectively.

4.4 The configuration of CNN-BiLSTM

Table 5 presents the configuration of CNN-BiLSTM architecture. These parameters are selected by conducting experimentation multiple times.

Table 5 Configuration of CNN-BiLSTM

Network layer	Parameters
Conv1	Filters = 32, activation = relu
MaxPool1	Shape = 1 × 32
Conv2	Filters = 64, activation = relu
MaxPool2	Shape = 1 × 64
Conv3	Filters = 128, activation = relu
MaxPool3	Shape = 1 × 128
Conv4	Filters = 256, activation = relu
GlobalMaxPool	Shape = 1 × 256
BiLstm	Units = 128, activation = tanh
Dense1	Shape = 1 × 200
Concatenate	Shape = 456
Dense2	Shape = 1 × 1024
Dropout	Rate = 0.5
Dense3	Shape = 1 × 2, activation = softmax

Selecting a better oversampling method plays a pivotal role in both dataset enhancements and accuracy improvement. To this end, we conduct experimentation to compare the performance of several oversampling. Table 6 summarizes the results of these oversampling methods including RandomOverSample, Smote, Svm-Smote, Kmeans-Smote [17, 31]. Kmeans-Smote obtains the highest accuracy and F_1 compared to the other three methods. Therefore, we use the Kmeans-SMOTE algorithm for data enhancement, which contains clustering, filtering, and oversampling steps. In the *clustering* step, the input space is clustered into k groups using a k-means cluster. The *filtering* step is charged for sparsely distributing minority samples. In the *oversampling* step, SMOTE algorithm is applied to each cluster for oversampling. It retains those clusters with a high proportion of minority samples and assigns more samples to clusters with selected clusters to achieve a target ratio of minority to majority instances.

4.5 Results of SmartRace

Table 7 presents the detection results of SmartRace. The *BoundedBuffer* benchmark obtains the highest accuracy (99.69%) and the highest recall (99.69%) among all benchmarks. Considering that samples in the *Montecarlo* benchmark are all negative, the precision can reach 100.00% and the F_1 can reach 99.79%, which is the highest among all benchmarks. The *Critical* benchmark has the worst performance because of insufficient training and testing samples. The performance of a deep model is highly dependable on the size of the dataset. Generally, the more training samples are available to the model, the better the performance could be. In general, the accuracy of the datasets ranges from 90.91 to 99.69%, which indicates *SmartRace* is effective in finding data races.

Table 6 Oversampling method

Oversampling method	Accuracy (%)	Precision (%)	Recall (%)	F_1 (%)
RandomOverSampler	93.13	98.37	91.02	93.94
Smote	94.63	98.37	92.11	94.59
Svm-smote	94.74	98.47	93.50	95.42
Kmeans-smote	97.26	98.09	97.23	97.89

Table 7 Detection results of *SmartRace*

Benchmarks	Accuracy (%)	Precision (%)	Recall (%)	F_1 (%)
Accoount	99.30	99.44	99.30	99.34
AirlineTickets	98.56	98.49	98.56	98.02
Bubblesort	96.63	97.48	96.63	96.92
BoundedBuffer	99.69	99.99	99.69	98.74
Bufwriter	96.63	98.31	96.63	97.16
Critical	90.91	91.92	90.91	90.27
Mergesort	92.00	95.20	92.00	92.81
Montecarlo	99.58	100.00	99.58	99.79
PingPong	93.75	95.00	93.75	93.96

4.6 Comparison with other deep neural network methods

DeepRace [14] is an existing data race detection tool based on deep learning. Since neither its dataset nor its model is open, we cannot compare it directly with SmartRace. Hence, we reimplement its model according to the design of DeepRace [14]. Since DeepRace uses a deep learning model CNN, we use CNN to represent the DeepRace method in this section.

To verify the performance of the CNN-BiLSTM hybrid neural network, we compare it with RNN [32], GRU [33], CNN [14], LSTM [34], and BiLSTM [35] under the same configuration. For a fair comparison, we follow the same process and tools to parse source code and encode tokens, as well as handle data imbalance. When constructing the network layers, all employed the Adam [36] optimizer with a learning rate of 0.001.

To make our experimental results reliable, we use a sixfold cross-validation method to evaluate our results. We mix the training dataset with the test dataset and then divide the whole dataset into 6 groups. 5 of 6 groups are taken as the training set and the remainder is considered as the test dataset. We repeat this process until each group is used as the test dataset, and finally, compute the mean value of six times. The experimental results are presented in Table 8. It can be seen from Table 8 that CNN-BiLSTM obtains the best performance, which demonstrates the effectiveness of SmartRace. For data race detection, CNN-BiLSTM significantly outperforms other models (RNN, GRU, CNN, LSTM, and BiLSTM) when considering the results of precision, recall, and F_1. To be more specific, CNN-BiLSTM improves F_1 by 3.84% (= 97.89–94.05%) compared to CNN. It also improves accuracy by 4.94% (= 97.26–92.32%) compared to CNN. Both LSTM and BiLSTM have lower accuracy and F_1 than CNN-BiLSTM. Specifically, it has 14.06% (= 97.26–83.20%) improvement for accuracy and 9.62% (= 97.89–88.27%) improvement for F_1 when comparing to LSTM. The same situation happens for comparison with BiLSTM with 9.28% (= 97.26–87.98%) and 6.21% (= 97.89–91.68%) improvements for accuracy and F_1, respectively.

To evaluate the performance of multiple algorithms across a set of data sets more intuitively, the STAC tool [37] is leveraged to test the significant statistical results between CNN-BiLSTM and other models. We use a non-parametric Friedman Aligned Ranks test [38] with significance level of 0.05 to examine the significant difference of accuracy as we consider that the data set is less than twice the number of models. From the first part, it can be seen that the H0 is rejected which means that there is a significant difference among these methods. Therefore, it is necessary to conduct pairwise comparisons to confirm the differences between the methods. We use the post-hoc Holm test [39] to choose CNN-BiLSTM as the control method to compare with other algorithms. From the Ranking part of Table 9, it indicates the average algorithm ranking of all algorithms on the same dataset. The lower the average rank of an algorithm is, the better the classification performance of the algorithm is. Results observed from Table 9 shows that the CNN-BiLSTM obtained the Rank value of 3.50000, which achieves the top rank when compared to other methods like LSTM, BiLSTM, GRU, RNN, and CNN. The next algorithm in the Ranking is the

Table 8 Comparison of different deep neural networks

	Accuracy (%)	Precision (%)	Recall (%)	F_1 (%)
RNN	89.73	91.11	89.48	89.76
GRU	89.45	90.32	89.77	89.37
CNN	92.32	96.35	92.32	94.05
LSTM	83.20	96.05	82.74	88.27
BiLSTM	87.98	97.61	87.86	91.68
CNN-BiLSTM	97.26	98.09	97.23	97.89

Table 9 STAC Analysis

Statistic		*p* value	Result
Friedman Aligned Ranks test (significance level of 0.05)			
25.84163		0.00010	H0 is rejected
Ranking			
Algorithm			Rank
CNN-BiLSTM			3.50000
CNN			10.75000
RNN			17.83333
GRU			20.66667
BiLSTM			24.83333
LSTM			33.41667
Comparison	Statistic	Adjusted *p* value	Result
CNN-BiLSTM versus LSTM	4.91827	0.00000	H0 is rejected
CNN-BiLSTM versus BiLSTM	3.50718	0.00181	H0 is rejected
CNN-BiLSTM versus GRU	2.82218	0.01431	H0 is rejected
CNN-BiLSTM versus RNN	2.35639	0.03691	H0 is rejected
CNN-BiLSTM versus CNN	1.19189	0.23330	H0 is accepted

CNN approach, followed by the RNN, GRU, BiLSTM, LSTM approaches with a big difference in the Ranking. In addition, it can be seen that the H0 is accepted between CNN-BiLSTM and CNN from the last part of Table 9, which shows that there is no statistically significant difference for the accuracy. Based on these comparisons, we find that the two models CNN and CNN-BiLSTM produce similar accuracy. However, the performance of CNN-BiLSTM is observed to be the best among the considered models in terms of statistical significance.

4.7 Compared with dynamic detection tools

We compare *SmartRace* with existing dynamic detection tools (Said [4] and RVPredict [5]) in Table 10, where "#R-races" represents the actual number of the known data races, "#TP" represents the number of detected data races, '#FN' represents the number of false negatives, and '#FP' represents the number of false positives.

As seen in Table 10, *SmartRace* reports 52 real data races in total, while Said reports 42 real data races and RVPredict reports 48 real data races. For the benchmarks Airlinetickets, BoundedBuffer, Bufwriter, and Critical, the real number of races detected by *SmartRace* is higher than Said. For the benchmark PingPong, *SmartRace* detected more than 4 real races than Said and RVPredict. By checking the real races of these benchmarks, we find that the numbers of data races detected by *SmartRace* are the same as the number of real races except for the Bubblesort benchmark.

For false negatives, the total number of false negatives for the Said and RVPredict tools are 11 and 5, respectively. Compared to both tools, the total number of false negatives for *SmartRace* is 1. In comparison, *SmartRace* detects fewer false negatives of data race. We find that this false negative occurs only in the *Bubblesort* benchmark. We insight this benchmark and related dataset. The possible reason is the low word relevance when converting a text

Table 10 Comparison with dynamic detection tools

Benchmarks	#R-races	Said			RVPredict			*SmartRace*		
		#TP	#FN	#FP	#TP	#FN	#FP	#TP	#FN	#FP
Account	4	4	0	1	4	0	1	4	0	1
AirlineTickets	7	6	1	0	7	0	2	7	0	2
Bubblesort	9	8	1	0	8	1	0	8	1	4
BoundedBuffer	12	10	2	0	12	0	1	12	0	3
Bufwriter	2	0	2	0	2	0	0	2	0	3
Critical	8	7	1	0	8	0	0	8	0	1
Mergesort	3	3	0	4	3	0	6	3	0	2
Montecarlo	0	0	0	0	0	0	0	0	0	2
PingPong	8	4	4	0	4	4	0	8	0	2
Total	53	42	11	5	48	5	10	52	1	20

feature with a label of 1 into a numerical vector, which results in a false negative.

Although *SmartRace* detects more real data races and has fewer false negatives, the other two detection tools outperform *SmartRace* when considering the number of false positives. SmartRace reports 20 false positives. Most false positives are found in Bubblesort, BoundedBuffer, and Bufwriter benchmarks. The possible reason is that many samples are labeled with 0 while few samples are labeled with 1, which results in an unbalanced sample distribution. When the Kmeans-SMOTE algorithm is used to enhance the balanced distribution, it inevitably results in a few false positives, which shows that *SmartRace* requires more improvement in detecting false positives.

4.8 Compared with static data race detection tools

To evaluate the effectiveness of *SmartRace*, we evaluate it on 9 benchmarks and compare it with the existing static tool SRD [9] in Table 11. The total number of real data races detected by *SmartRace* is the closest to the number of real races. A total of 52 data races is detected by *SmartRace*, while only 39 real data races are detected by SRD. SRD also reports 14 false negatives in total on the Bubblesort, Critical, and PingPong. Specifically, it reports 2 false negatives for the Critical benchmark and 4 false positives for the PingPong benchmark.

For these two benchmarks, SmartRace does not report any false negatives. For the Bubblesort benchmark, SRD reports 8 false negatives.

We should note that one false negatives is reported by *SmartRace*. The possible reason for false negatives has been mentioned in Sect. 4.7.

As far as false positives are concerned, *SmartRace* has one less than SRD in general. Compared to SRD, *SmartRace* effectively decreases the number of false positives for these benchmarks. However, the false positives of SRD are mainly manifested in Mergesort, while *SmartRace* is evenly distributed in every program. The possible reason is that the final test result of the neural network cannot reach 100% without overfitting. Overall, *SmartRace* outperforms SRD, in terms of the total number of actual races detected, as well as false negatives and false positives.

Table 11 Compared with static data race detection tools

Benchmarks	#R-races	SRD			*SmartRace*		
		#TP	#FN	#FP	#TP	#FN	#FP
Account	4	4	0	1	4	0	1
AirlineTickets	7	7	0	3	7	0	2
Bubblesort	9	1	8	0	8	1	4
BoundedBuffer	12	12	0	4	12	0	3
Bufwriter	2	2	0	0	2	0	3
Critical	8	6	2	0	8	0	1
Mergesort	3	3	0	12	3	0	2
Montecarlo	0	0	0	1	0	0	2
PingPong	8	4	4	0	8	0	2
Total	53	39	14	21	52	1	20

Table 12 Time overheads

Component	Time (S)
Feature extraction	86.17
Data pre-processing	3.40
Embedding layer	30.37
Data set expansion	0.91
Data set loading	0.02
Model training	6096.73
Total	6217.60

4.9 Time overhead

We evaluate the time performance of *SmartRace* in detecting data race for 9 benchmarks. Table 12 presents the time overhead of *SmartRace* in detecting data race, with a total time of 6217.6 s. The time-consuming step is the training time of the deep neural network model with a total of 6096.73 s which is more than 98% of the overall detection time. The main reason is that both the time and space complexity of the model have a huge impact on the training time. *SmartRace* leverages four convolutional layers and one BiLSTM layer to obtain different features and merge them. The number of iterations also contributes to the training time.

5 Threats to validity

This section discusses several factors that threaten the validity of our experimental results.

The main threat to the validity of our results is that we barely generate the dataset from 25 real-world applications that may not be representative of all programs. However, we have chosen programs from several different benchmark suites: Dacapo, IBM Contest, JBench, JGF, and PJBench. Although it may be that other classes of applications could exhibit very different attributes, it seems very likely that our results should still obtain good accuracy based on these applications.

The second threat to the validity of our results is that we use the tool ConRacer to mark the dataset. Although ConRacer uses a context-sensitive program analysis approach to effectively report data race, there are still false positives and false negatives. To mitigate this threat, we employ a manual approach to reported data races by checking the location of codes where data races occur to remove false positives and false negatives as much as possible to maximize the accuracy of the dataset.

The third threat to the validity of our results is that when converting text features into numerical vectors, conversion similarity affects the accuracy of the final result. In this paper, we adopt the embedding layer of Keras for text vectorization. By training the embedding layer of Keras and tuning its parameters, the conversion accuracy is up to 99.2. Although 100% accuracy is not achieved, the numerical vectors can better describe the textual features of the data race, effectively reducing the impact of text vectorization on the final results.

The final threat to the validity of our results is that various data enhancement algorithms have variable effects on the final results of training. Kmeans-SMOTE employs both the Kmeans algorithm [40] and SMOTE oversampling to reduce the impact of data enhancement algorithms on model training. The uniqueness of the method lies not only in the breadth and simplicity of SMOTE and k-means but also in its novel and effective synthetic sample distribution method. The sample distribution is based on the clustering density, which produces more samples in sparse minority areas than in dense areas to eliminate imbalances. Moreover, Kmeans-SMOTE is applicable since it clusters regardless of the class label to enable the detection of areas safe for oversampling. Overfitting is prevented by generating new samples instead of replicating them.

6 Conclusion

This paper proposes a novel approach based on CNN-BiLSTM hybrid neural network called *SmartRace*. Firstly, the approach extracts multi-level features from several real-world applications via a static analysis tool WALA to build the training set. To judge the real data race, we employ the existing data race detection tool ConRracer and mark the samples. Secondly, the samples are vectorized by the Embedding layer of Keras. We also leverage the Kmeans-SMOTE algorithm to make the samples distribute balancing. Finally, the CNN-BiLSTM network is constructed and trained to detect data race. In the experimentation, 9 benchmarks are selected to verify the effectiveness of the method. The experimental results show that the accuracy of *SmartRace* is 97.26%, which is improved 4.94% compared with the existing deep learning-based detection method called DeepRace. In addition, we have analyzed the experimental of each model using statistical tests including Friedman Aligned Ranks test and the post-hoc Holm test. The study also finds that the two models CNN and CNN-BiLSTM produced more or less similar forecast accuracy. However, the performance of CNN-BiLSTM is observed to be best among the considered models. In addition, we compare *SmartRace* with the existing data race detection tools Said, RVPredict and SRD and validate the effectiveness of *SmartRace*.

The future work includes that we will improve our approach to eliminate false positives as many as possible by enriching the training dataset with the positive samples. Although the accuracy of SmartRace is higher than existing deep-learning-based approaches, we will continue to optimize our model to further improve the accuracy.

Acknowledgements The authors gratefully acknowledge the helpful comments and suggestions from the reviewers. This work was supported in part by the Natural Science Foundation of Hebei Province under Grant F2022208009.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Gu Y, Mellor-Crummey J (2018) Dynamic data race detection for OpenMP programs. In: International conference for high performance computing, networking, storage and analysis, pp 767–778. IEEE
2. Blackshear S, Gorogiannis N, O'Hearn PW, Sergey I (2018) RacerD: compositional static race detection. Proc ACM Program Lang 2:1–28
3. Sen A, Kalaci O (2018) Hybrid data race detection for multicore software. Comput Inform 37(1):186–212
4. Said M, Wang C, Yang Z, Sakallah K (2011) Generating data race witnesses by an SMT-based analysis. In: NASA formal methods symposium. Springer, Berlin, Heidelberg, pp 313–327
5. Huang J, Meredith PON, Rosu G (2014) Maximal sound predictive race detection with control flow abstraction. In: Proceedings of the 35th ACM SIGPLAN conference on programming language design and implementation, pp 337–348
6. Peng Y, DeLozier C, Eizenberg A, Mansky W, Devietti J (2018) Slimfast: reducing metadata redundancy in sound and complete dynamic data race detection. In: 2018 IEEE international parallel and distributed processing symposium (IPDPS), pp 835–844. IEEE
7. Voung JW, Jhala R, Lerner S (2007) RELAY: static race detection on millions of lines of code. In: Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering, pp 205–214
8. Taft ST, Schanda F, Moy Y (2016) High-integrity multitasking in spark: static detection of data races and locking cycles. In: 2016 IEEE 17th international symposium on high assurance systems engineering (HASE), pp 238–239. IEEE
9. Zhang Y, Liang Y, Zhang D, Sun S (2019) Data race detection approach in concurrent programs. J Comput Appl 39(1):61
10. Yang, Z, Yu Z, Su X, Ma P (2016) RaceTracker: effective and efficient detection of data races. In: 2016 17th IEEE/ACIS international conference on software engineering, artificial intelligence, networking and parallel/distributed computing (SNPD), pp 293–300. IEEE
11. Yang J, Jiang B, Chan WK (2018) Histlock+: precise memory access maintenance without lockset comparison for complete hybrid data race detection. IEEE Trans Reliab 67(3):786–801
12. Yu T, Wen W, Han X, Hayes JH (2018) Conpredictor: concurrency defect prediction in real-world applications. IEEE Trans Softw Eng 45(6):558–575
13. Li, J, He P, Zhu J, Lyu MR (2017) Software defect prediction via convolutional neural network. In: 2017 IEEE international conference on software quality, reliability and security (QRS), pp 318–328. IEEE.
14. Tehrani A, Khaleel M, Akbari R, Jannesari A (2019) DeepRace: finding data race bugs via deep learning. arXiv preprint arXiv:1907.07110
15. IBM. T.J Watson libraries for analysis (WALA). http://wala.sourceforge.net
16. Zhang Y, Liu H, Qiao L (2021) Context-sensitive data race detection for concurrent programs. IEEE Access 9:20861–20867
17. Douzas G, Bacao F, Last F (2018) Improving imbalanced learning through a heuristic oversampling method based on k-means and SMOTE. Inf Sci 465:1–20
18. Rhanoui M, Mikram M, Yousfi S, Barzali S (2019) A CNN-BiLSTM model for document-level sentiment analysis. Mach Learn Knowl Extr 1(3):832–847
19. Blackburn SM, Garner R, Hoffmann C, Khang AM, McKinley KS, Bentzur R, Diwan A, Feinberg D, Frampton D, Guyer SZ, Hirzel M, Wiedermann B (2006) The DaCapo benchmarks: java benchmarking development and analysis. In: Proceedings of the 21st annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications, pp 169–190
20. Farchi E, Nir Y, Ur S (2003) Concurrent bug patterns and how to test them. In: Proceedings international parallel and distributed processing symposium. IEEE
21. Gao J, Yang X, Jiang Y, Liu H, Ying W, Zhang X (2018) Jbench: a dataset of data races for concurrency testing. In: Proceedings of the 15th international conference on mining software repositories, pp 6–9
22. Smith LA, Bull JM, Obdrizalek J (2001) A parallel Java Grande benchmark suite. In: SC'01: proceedings of the 2001 ACM/IEEE conference on supercomputing, pp 6–6. IEEE
23. Yu M, Bae DH (2016) SimpleLock+: fast and accurate hybrid data race detection. Comput J 59(6):793–809
24. Li MK, Zheng QS, Wang L (2020) A dynamic hybrid data race detection algorithm based on sampling technique. Comput Sci 47(10):315–321
25. Gao F, Wang Y, Zhou J et al (2021) High precision large-scale program data race detection method. J Softw 32(7):2039–2055. https://doi.org/10.13328/j.cnki.jos.006260
26. Kusano M., Wang C (2013) CCmutator: a mutation generator for concurrency constructs in multithreaded C/C++ applications. In: 28th IEEE/ACM international conference on automated software engineering (ASE), pp 722–725. IEEE
27. Jiaze S, Jiawei Y, Zijiang Y (2020) Multithreaded program data race random forest instruction level detection model. J Tsinghua Univ (NATURAL SCIENCE EDITION) 60(10):804–813
28. Ali A, Zhu Y, Chen Q et al (2019) Leveraging spatio-temporal patterns for predicting citywide traffic crowd flows using deep hybrid neural networks. In: 2019 IEEE 25th international conference on parallel and distributed systems (ICPADS). IEEE
29. Ali A, Zhu Y, Zakarya M (2021) A data aggregation based approach to exploit dynamic spatio-temporal correlations for citywide crowd flows prediction in fog computing. Multim Tools Appl 80(20):31401–31433. https://doi.org/10.1007/s11042-020-10486-4
30. The Python Deep Learning library (Keras). https://keras.io/
31. Chawla NV, Bowyer KW, Hall LO, Kegelmeyer WP (2002) SMOTE: synthetic minority over-sampling technique. J Artif Intell Res 16:321–357
32. Zaremba W, Sutskever I, Vinyals O (2014) Recurrent neural network regularization. Eprint Arxiv
33. Cho K, Merrienboer BV, Gulcehre C et al (2014) Learning phrase representations using RNN encoder-decoder for statistical machine translation. Comput Sci
34. Makarenkov V, Guy I, Hazon N, Meisels T, Shapira B, Rokach L (2019) Implicit dimension identification in user-generated text with LSTM networks. Inf Process Manag 56(5):1880–1893
35. Liu G, Guo J (2019) Bidirectional LSTM with attention mechanism and convolutional layer for text classification. Neurocomputing 337:325–338
36. Kingma DP, Ba J (2014) Adam: a method for stochastic optimization. arXiv preprint arXiv:1412.6980.
37. Rodríguez-Fdez I, Canosa A, Mucientes M et al (2015) STAC: a web platform for the comparison of algorithms using statistical tests. In: 2015 IEEE international conference on fuzzy systems (FUZZ-IEEE), pp 1–8. IEEE
38. Hodges JL, Lehmann EL (2012) Rank methods for combination of independent experiments in analysis of variance. In: Rojo J

(ed) Selected works of E. L. Lehmann. Springer, Boston, MA, pp 403–418
39. Holm S (1979) A simple sequentially rejective multiple test procedure. Scand J Stat 6(2):65–70
40. Hamerly G, Elkan C (2004) Learning the k in k-means. Adv Neural Inf Process Syst 16:281–288