NegCPARBP: Enhancing Privacy Protection for Cross-Project Aging-Related Bug Prediction Based on Negative Database

Dongdong Zhao, Zhihui Liu, Fengji Zhang, Lei Liu, Jacky Wai Keung, *Senior Member, IEEE*, and Xiao Yu

***Abstract*—The emergence of Aging-Related Bugs (ARBs) poses a significant challenge to software systems, resulting in performance degradation and increased error rates in resource-intensive systems. Consequently, numerous ARB prediction methods have been developed to mitigate these issues. However, in scenarios where training data is limited, the effectiveness of ARB prediction is often suboptimal. To address this problem, Cross-Project Aging-Related Bug Prediction (CPARBP) is proposed, which utilizes data from other projects (i.e., source projects) to train a model aimed at predicting potential ARBs in a target project. However, the use of source-project data raises privacy concerns and discourages companies from sharing their data. Therefore, we propose a method called Cross-Project Aging-Related Bug Prediction based on Negative Database (NegCPARBP) for privacy protection. NegCPARBP first converts the feature vector of a software file into a binary string. Second, the corresponding Negative DataBase (*NDB*) is generated based on this binary string, containing data that is significantly more expressive from the original feature vector. Furthermore, to ensure more accurate prediction of ARB-prone and ARB-free files based on privacy-protected data (i.e., maintain the data utility), we propose a novel negative database generation algorithm that captures more information about important features, using information gain as a measure. Finally, NegCPARBP extracts a new feature vector from the *NDB* to represent the original feature vector, facilitating data sharing and ARB prediction objectives. Experimental results on Linux, MySQL, and NetBSD datasets demonstrate that NegCPARBP achieves a high defense against attacks (privacy protection performance reaching 0.97) and better data utility compared to existing privacy protection methods.**

***Index Terms*—Aging-related bugs prediction, privacy protection, negative database.**

Received 18 March 2024; revised 22 January 2025; accepted 24 February 2025. Date of publication 6 March 2025; date of current version 27 June 2025. This work was supported in part by the National Natural Science Foundation of China under Grant 61806151, in part by the National Key Research and Development Program under Grant 2022YFB3104001 and Grant 2022YFC3321102, and in part by the Natural Science Foundation of Chongqing City under Grant CSTC2021JCYJ-MSXMX0002 and Grant CSTC2021JCYJ-MSXMX1115. *(Corresponding author: Xiao Yu.)*

Dongdong Zhao is with the School of Computer Science and Artificial Intelligence, Wuhan University of Technology, Wuhan 430062, China, and also with the Chongqing Research Institute, Wuhan University of Technology, Chongqing 401135, China (e-mail: zdd@whut.edu.cn).

Zhihui Liu is with the School of Computer Science and Artificial Intelligence, Wuhan University of Technology, Wuhan 430062, China (e-mail: zhihuil@whut.edu.cn).

Fengji Zhang and Jacky Wai Keung are with the Department of Computer Science, City University of Hong Kong, Hong Kong (e-mail: fengji.zhang@my.cityu.edu.hk; jacky.keung@cityu.edu.hk).

Lei Liu is with the School of Electronic Science and Engineering, Xi'an Jiaotong University, Xi'an 710049, China (e-mail: lei.liu@stu.xjtu.edu.cn).

Xiao Yu is with the State Key Laboratory of Blockchain and Data Security, Zhejiang University, Hangzhou 310027, China (e-mail: xiao.yu@zju.edu.cn).

Digital Object Identifier 10.1109/TETC.2025.3546549

I. Introduction

PROLONGED operation of software systems can lead to performance degradation and increased error rates, ultimately resulting in system failures [1], [2]. This phenomenon, known as software aging, has been observed in several systems and fields such as operating systems, telecommunications systems, web servers, database systems, and embedded systems [3]. Software aging can cause serious damage, including economic losses, damage to company credibility, compromised security, increased maintenance costs, and potential risks to human lives in extreme cases [4], [5]. Aging-Related Bug (ARB) is one of the key factors that cause software aging, which can lead to issues such as resource leaks, memory fragmentation, and performance degradation, thereby exacerbating the degree of software aging [6], [7]. Therefore, detecting and predicting ARBs automatically is crucial, as it can help developers identify potential issues early and mitigate the effects of software aging [1], [8].

Recent studies have explored the feasibility of using static source code features to build machine learning models to predict ARBs [9], [10]. However, these methods mainly focus on within-project ARB prediction, which requires a large amount of training data to build models in order to perform well [1]. However, in practice, collecting the training data for ARB prediction is challenging [6]. First, unlike many other types of software bugs, ARBs often lead to the accumulation of errors, which may eventually result in system failures, requiring prolonged execution times to observe [7], [10]. Second, ARBs account for a small proportion of all analyzed bugs, necessitating the analysis of a large number of bug reports to select ARBs, which increases the difficulty of collecting a sufficient amount of training data [7], [11]. Third, for projects in the initial development stage or for

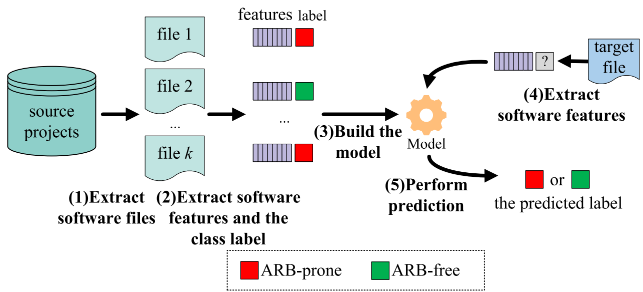

Figure 1. The process of CPARBP.

smaller companies, there may be insufficient historical data or the cost of collecting training data may be high [6].

To overcome these challenges, several researchers [1], [6] have proposed Cross-Project Aging-Related Bugs Prediction (CPARBP), which involves training a model by using data from source projects, and use it to predict ARBs in target projects.

Figure 1 shows the process of CPARBP. In the first phase, software files are extracted from one or more other projects. Moving on to the second phase, the software features and corresponding class labels (i.e., ARB-prone or ARB-free) of these files are extracted. These two steps contribute to the construction of a software ARB dataset. The third phase involves building a CPARBP model using the constructed dataset. Finally, in the fourth and fifth phases, after extracting the same software features from the target software file, the CPARBP model trained in the third phase is employed to predict the class label of the target software file.

A. Motivation

CPARBP effectively resolves the issue of insufficient training data and demonstrates impressive defect prediction performance [12], [13], [14]. However, the utilization of source-project data also presents an increased risk of sensitive information leakage, as highlighted in prior Cross-Project Defect Prediction (CPDP) research [15], [16], [17], [18]. These privacy concerns often discourage companies from sharing their data, especially when it involves sensitive features [15]. For example, features such as lines of code, which measure the size and complexity of a software program, can potentially reveal business-sensitive information such as project development efforts [19]. Additionally, features like McCabe's complexity that measure code complexity can inadvertently expose sensitive proprietary logic or project intricacies [16]. Hence, protecting the privacy of data owners is crucial for enabling data sharing.

Currently, there is no relevant work on privacy protection in the CPARBP field. However, several privacy protection methods have been proposed for traditional CPDP, including the perturbation-based MORPH method proposed by Peters et al. [15], the LACE method based on generalization and perturbation that outperformed MORPH in maintaining defect prediction performance [16], LACE2 designed for security concerns in multi-party CPDP scenarios [17], and the SRDO method by Li et al. [18], which enhanced the LACE method using sparse matrices and double perturbation. However, although their methods exhibited relatively good privacy protection capabilities (with the best privacy protection rate being 0.95) in the experiments, they may not necessarily meet the requirements of some scenarios with higher privacy protection demands.

B. Our Work and Contributions

To enhance privacy protection, we propose a privacy protection method named NegCPARBP (Cross-Project Aging-Related Bug Prediction based on Negactive Database).

NegCPARBP is inspired by the negative selection mechanism in the field of artificial immune systems, which has the ability to distinguish between self and non-self antigens. The NegCPARBP method involves three main steps: data preprocessing, Negative DataBase (*NDB*) generation, and *NDB* data extraction. First, we convert the feature vector of a software file into a binary string. Second, we generate the corresponding *NDB* for the binary string, where the *NDB* contains data that significantly differs from the original feature vector. Furthermore, to maintain data utility (i.e., the CPARBP models established based on privacy-protected data can accurately predict ARB-prone and ARB-free files), we propose a novel *NDB* generation algorithm that can capture more information about important features for CPARBP (measured by information gain). Finally, we extract a new feature vector from the *NDB* to represent the original feature vector for data sharing and ARB prediction purposes. In our experiments, conducted on the Linux, MySQL, and NetBSD datasets, we demonstrate that the probability of successfully defending against attacks surpasses 0.97, which is better than the performance of MORPH [15], LACE [16], and SRDO [18] by 8.1%–10.9%, 8.4%–10.7%, and 7.2%–10.5%. Moreover, compared to MORPH, LACE, SRDO, and the latest *NDB* generation algorithm *QK*-hidden, CPARBP models trained on data processed by NegCPARBP achieve better *PD* (3.6%–1161.3% higher than other methods), *G-measure* (0.5%–618.3%), and *Balance* (0.7%–135.2%).

Our contributions are summarized as follows:

- We propose a novel privacy-preserving method called NegCPARBP, marking the first attempt to introduce *NDB* for generating privacy-preserving data intended for sharing and ARB prediction.
- We propose a novel *NDB* generation algorithm named *IK*-hidden, which can capture more information about important features for CPARBP based on information gain.
- We conduct experiments on three datasets to compare the privacy-preserving capability and data utility of our method with three existing privacy-preserving methods in the CPDP domain and the latest *NDB* generation method. Experimental results demonstrate that our method achieves better privacy protection and data utility.
- We have made the code of our work publicly available, including the *NDB* generation and *NDB* data extraction program[1].

[1]https://github.com/AtLeastIAmHere/IK-hidden.git

C. Organization

The rest of this paper is organized as follows. Section II introduces the background of CPARBP, related work for privacy protection in CPDP, and the background of *NDB*. In Section III, we describe the proposed method. In Sections IV and V, we present the experimental setup and our experimental results. Section VI discusses the impact of feature selection, highlights the key findings of our study, and potential threats to the validity of our study. Section VII concludes this work.

II. Related Work and Background

This section introduces the existing CPARBP methods and the privacy-preserving methods in CPDP. Meanwhile, we describe *NDB* with its relevant applications in the security field in recent years.

A. Cross-Project Aging-Related Bug Prediction

In recent years, numerous CPARBP methods have been proposed. Qin et al. [12] introduced the TLAP (Transfer Learning based Aging-related bug Prediction) method, which pioneered the fusion of transfer component analysis with random oversampling to deal with the severe class imbalance issue in cross-project scenarios. Subsequently, Qin et al. [6] additionally investigated the ARBs prediction with the Apache HTTPD server project using the TLAP method. The experimental results showed that the number of ARBs in susceptible files and the similarity of their distribution can affect ARB prediction performance. Wan et al. [1] proposed the SRLA (Supervised Representation Learning Approach) method, leveraging deep autoencoder techniques to enhance label-enriched representations and mitigate class imbalance through random oversampling. Xu et al. [13] proposed the JDA-ISDA (Joint Distribution Adaptation and Improved Subclass Discriminant Analysis) method, which utilized JDA to jointly reduce marginal and conditional distribution differences, and then applied ISDA to alleviate severe class imbalance issues. Kaur et al. [14] achieved CPARBP in cloud computing applications by automatically extracting and predicting ARBs. The results showed that the Naive Bayes classifier can exhibit great performance when handling imbalanced data.

B. Privacy Protection for Cross-Project Defect Prediction

Generally, data owners express concerns about the privacy and security of their data. Currently, there is no research about privacy protection for ARB data. However, research has been conducted on privacy protection in traditional CPDP domain.

Peters et al. [15] proposed the MORPH method, which perturbed data using other data with the closest euclidean distance but encountered challenges in maintaining the data utility for defect prediction in some datasets. MORPH performed data perturbation based on Formula (1),

$$\boldsymbol{x}' = \boldsymbol{x} \pm (\boldsymbol{x} - \boldsymbol{z}) \times r, \tag{1}$$

where $\boldsymbol{x}$ represented the feature vector of the software file *M*, $\boldsymbol{z}$ was the feature vector of that file closest to *M* which has a different label, *r* was a randomly generated number within the range of [0.15, 0.35], and $\boldsymbol{x}'$ denoted the new feature vector of *M* after processing, which replaced **x** for sharing purposes.

Peters et al. [16] proposed the LACE method, which combined the newly proposed CLIFF data pruner and the MORPH method. LACE starts by employing the CLIFF algorithm to remove a certain percentage of data from the original dataset. Subsequently, it applies the MORPH method to perturb the remaining data. CLIFF utilizes the approach introduced by Jalali et al. [20] to calculate weights for each file, and removes the subset of data with lower weights.

To enhance the privacy protection capabilities in multi-party scenarios, Peters et al. [17] introduced the LACE2 method, which integrated LeaF technology [21]. LeaF, based on the leader-follower algorithm for data clustering, enabled a multi-party environment where data owners can progressively incorporate "interesting" data into a shared private cache, leveraging the existing content within the cache.

Additionally, Li et al. [18] proposed the SRDO method as an improvement over LACE [16]. SRDO is similar to LACE but improves the MORPH method. SRDO uses the CLIFF method to remove some data with lower weights. Then the data perturbation is performed using Formula (2),

$$\boldsymbol{x}' = \boldsymbol{x} + (\boldsymbol{x} - \boldsymbol{z}_{same}) \times r_1 - sign(r_1)(\boldsymbol{x} - \boldsymbol{z}_{diff}) \times |r_2|, \tag{2}$$

where $\boldsymbol{x}$ and $\boldsymbol{x}'$ have the same meaning as in Formula (1). The values of r_1 and r_2 are randomly chosen from the range of [−0.35, −0.15] or [0.15, 0.35]. $\boldsymbol{z}_{same}$ represents the feature vector of that file closest to the software file *M* which has the same class label, while $\boldsymbol{z}_{diff}$ represents the feature vector of that file closest to the software file *M* which has a different class label.

Additionally, SRDO applied the sparse representation technique [22], where a software file's feature vector with *T* dimensions is encoded as a sparse linear combination of dictionary atoms. This technique is robust against noisy data and requires solving the following optimization problem to achieve a sparse representation:

$$\min_{\mu \geq 0} ||\boldsymbol{x} - \boldsymbol{B}\boldsymbol{\eta}||_2^2 + \mu||\boldsymbol{\eta}||_1, \tag{3}$$

where $\boldsymbol{B} \in \mathbb{R}^{T \times n}$ denoted the dictionary atoms, typically constructed using the feature vectors of the *n* training software files. The parameter μ controlled the sparsity of the solution. The coefficient vector $\boldsymbol{\eta}$ was sparse, containing only a few non-zero values, where the feature vector, which is the most similar to $\boldsymbol{x}$, corresponds to the largest non-zero value.

Despite these advanced privacy protection methods for CPDP, the level of privacy protection achieved may still fall short of expectations.

C. Negative Database

The *NDB* is an information representation approach inspired by the negative selection mechanism in the artificial immune system. In the artificial immune system, the negative selection mechanism is used to exclude data that is highly similar to known patterns in order to highlight relatively rare or anomalous data. Similarly, an *NDB* is constructed by excluding data from the

TABLE I
AN EXAMPLE OF THE *NDB* WITH THE CORRESPONDING *DB*

positive DB	$U - DB$	*NDB*
001	000	*00
010	011	1**
	100	*11
	101	
	110	
	111	

universal set that is similar to known patterns, thus emphasizing data with distinctive features or significantly differing from the known patterns. The *NDB* is capable of storing the complement of the original data and can perform operations similar to those of the original database. Esponda et al. [23], [24] demonstrated that attacking the *NDB* is equivalent to solving the boolean satisfiability problem, which is known to be an NP-hard problem. Therefore, utilizing the *NDB* can effectively prevent attackers from directly accessing sensitive information, ensuring robust data security.

To illustrate an example of the *NDB*, we consider a positive database DB consisting of the two strings "001" and "010". Table I presents the corresponding *NDB*. In the case of binary strings with a length of L=3, the size of the universal set U={000, 001, 010, 011, 100, 101, 110, 111} is 2^L (=8). By excluding the original data "001" and "010" from this universal set, we derive the $U - DB$={000, 011, 100, 101, 110, 111}. Since the size of the *NDB* is usually large, compression becomes necessary in practical applications. Compression is achieved using the symbol '*', allowing strings like "000" and "100" to be compressed and represented as "*00".

NDB is capable of representing the original data in an equivalent manner and provides robust privacy protection. It has demonstrated impressive performance across various fields, such as data mining [25], secure multi-party computing [26], and deep learning model [27].

Currently, the widely used generation algorithms for *NDB* primarily consist of *q*-hidden [28], *p*-hidden [29], *K*-hidden [30], and *QK*-hidden [25]. *K*-hidden utilized the parameters *K* and $[p_1, p_2, \ldots, p_K]$ to control the hardness of the resulting *NDB*. The *QK*-hidden improved the *K*-hidden algorithm by introducing a set of parameters $[q_1, q_2, \ldots, q_L]$, which controlled the probabilities of choosing bits when generating a specific type of record. The parameters $[q_1, q_2, \ldots, q_L]$ enabled *QK*-hidden to capture more information about important bits for classification and clustering.

III. OUR APPROACH

A. Overview

A software with T features can be represented as $M = (\boldsymbol{x}, y)$, where $\boldsymbol{x} = (x_1, x_2, \ldots, x_T)$ represents the feature vector of the software file *M* and *y* is the class label (i.e., 1 represents ARB-prone or 0 represents ARB-free). The primary objective of our methods is to transform the feature vector $\boldsymbol{x}$ of each file in the ARB dataset into a new feature vector $\boldsymbol{x}'$. Since the new feature vector $\boldsymbol{x}'$ is challenging to reverse to the original feature vector $\boldsymbol{x}$, we can employ it to replace $\boldsymbol{x}$ for data sharing and ARB prediction.

Algorithm 1: *IK*-Hidden.

Input: an *m*-bits string *s*; the number of specified bits in record *K*; the number of features *T*; the length of the binary representation of each feature *L*; a constant *r*; the probability parameters $\boldsymbol{p} = [p_1, p_2, \ldots, p_K]$, $\boldsymbol{q} = [q_1, q_2, \ldots, q_L]$, and $\boldsymbol{f} = [f_1, f_2, \ldots, f_T]$.
Ouput: NDB_s.
1: $NDB_s \leftarrow \emptyset$;
2: $N \leftarrow m \times r$;
3: $\boldsymbol{P} = [P_0, P_1, \ldots, P_K]: \ P_0 \leftarrow 0, \ P_i \leftarrow p_1 + \cdots + p_i$;
4: $\boldsymbol{Q} = [Q_0, Q_1, \ldots, Q_L]$: $Q_0 \leftarrow 0, \ Q_i \leftarrow q_1 + \cdots + q_i$;
5: $\boldsymbol{F} = [F_0, F_1, \ldots, F_T]: \ F_0 \leftarrow 0, \ F_i \leftarrow f_1 + \cdots + f_i$;
6: **while**$(|NDB_s| < N)$
7: Initialize a record τ with m '*';
8: $rndp \leftarrow random([0, 1))$;
9: Find $type$: $P_{type-1} \leq rndp < P_{type}$;
10: **for** idx from $1 \rightarrow type$:
11: $rndf \leftarrow random([0, 1))$;
12: Find *i*: $F_{i-1} \leq rndf < F_i$;
13: Select the *i*th feature in τ;
14: $rndq \leftarrow random([0, 1))$;
15: Find *j*: $Q_{j-1} \leq rndq < Q_j$;
16: Select the *j*th bit of the *i*th feature in τ;
17: **if** this bit has been selected: goto to line 11;
18: **else**: Make the selected bit different from s;
19: **end for**
20: Randomly select other $K - type$ bit(s) of τ to be same with s;
21: $NDB_s \leftarrow NDB_s \cup \tau$;
22: **end while**
23: **return** NDB_s.

The NegCPARBP method consists of three main steps: data preprocessing, *NDB* generation, and *NDB* data extraction.

1) In the data preprocessing phase, NegCPARBP employs max-min normalization to scale all feature values of a software file into the [0, 1] range. Subsequently, it converts the decimal digits of feature values into binary strings for feature representation. These binary strings from all features are concatenated to create a new string s.
2) In the second step, the *NDB* of the string s is generated using our proposed *IK*-hidden algorithm.
3) Finally, we extract a new feature vector $\boldsymbol{x}'$ from the *NDB* to represent the software file *M* for data sharing and ARB prediction.

B. IK-Hidden Algorithm

In most cases of ARB prediction, different features usually have different impacts on the prediction performance and some important software features contribute more to prediction performance [10]. However, the latest *NDB* generation algorithm, *QK*-hidden, treats all features equally, including some important features [25]. Therefore, the data utility of ARB datasets may be compromised.

To solve the above problem, we make improvements to the *QK*-hidden algorithm and propose a new *NDB* generation algorithm, *IK*-hidden. We employ a set of new parameters $[f_1, f_2, \ldots, f_T]$ to control the probability of selecting different features, which can capture more information about important features for CPARBP.

As shown in Algorithm 1, in *IK*-hidden, the input is the hidden string *s*, the number of specified bits in record *K*, the number of features *T*, the length of the binary representation of each feature *L*, the parameter *r* which controls the size of NDB_s, and the probability parameters $[p_1, p_2, \ldots, p_K]$, $[q_1, q_2, \ldots, q_L]$, and $[f_1, f_2, \ldots, f_T]$. The output is the negative database NDB_s of the hidden string *s*.

The following is the main flow of the *IK*-hidden algorithm:

1) Initialize NDB_s as the empty set (Line 1), and initialize the other parameters (Lines 2-5).
2) Randomly choose which *type* (the number of specified bits in the record that are opposite to the hidden string s) of the record to generate based on probability parameters $[p_1, p_2, \ldots, p_K]$ (Lines 8-9).
3) Generate a record, where *type* bit(s) is/are randomly selected based on probability parameters $[f_1, f_2, \ldots, f_T]$ and $[q_1, q_2, \ldots, q_L]$. In this step, a random number $rndf$ is generated (Line 11). Next, a value of *i* such that $f_1 + \cdots + f_{i-1} \leq rndf < f_1 + \cdots + f_i$ is found (Line 12). Then, the *i*th feature is selected (Line 13), and the *j*th bit of the *i*th feature is selected by $[q_1, q_2, \ldots, q_L]$ in the same way (Lines 14-16). If this bit has been selected previously, then proceed to reselect the bit (Line 17). If the bit has not been selected before, then set the selected bit to be the opposite of the jth bit of the ith feature of s (Line 18). This process is iterated until *type* different bit(s) is/are chosen to construct the new record (Lines 10-19). The remaining $K - type$ bit(s) is/are randomly selected with the same probability for each bit (Line 20). Finally, the record is added to NDB_s (Line 21).
4) Repeat 2) and 3) until the size of NDB_s reaches *N*.

For the parameter setting of $[f_1, f_2, \ldots, f_T]$, we employ the classical information gain [31] calculation method to assess the contribution of each feature to ARB prediction (i.e., identify the important software features). Information gain measures the reduction in entropy achieved by partitioning a dataset based on a specific feature, aiding algorithms in selecting the most informative splitting attributes [31]. Information gain is defined as

$$IG(x_i) = H(C) - H(C|x_i), \tag{4}$$

where x_i is the *i*th feature of files, and *C* is the set of class labels {0, 1}. And $H()$ represents the entropy, which can bey defined as

$$H(C) = -\sum_{c\in C} P(c) \times log_2 P(c) \tag{5}$$

and

$$H(C|x_i) = -\sum_{a\in D(x_i)} P(a) \sum_{c\in C} P(c|a) \times log_2 P(c|a), \tag{6}$$

where $P()$ represents the probability of a specific feature value occurring within the dataset, and $D(x_i)$ is the set of values of feature x_i. If a feature consists of floating-point values, we divide the range of values for that feature in the dataset into 10 intervals. When computing information gain, values within each interval are considered the same.

Meanwhile, the average information gain across all features can be calculated by $avgIG = \frac{1}{T}\sum_{i=1}^{T} IG(x_i)$. In the *IK*-hidden algorithm, we establish the probability of selecting each feature to satisfy $f_i = 2f_j$ if $IG(x_j) < avgIG \leq IG(x_i)$, where features with information gain values greater than or equal to $avgIG$ are chosen with double the probability compared to those with information gain values less than $avgIG$. This step selectively generates the bits of records in the *NDB* for different features, allowing it to capture more information about important features for CPARBP.

It is worth noting that the parameters $[f_1, f_2, \ldots, f_T]$ in the *IK*-hidden algorithm only affect the probability of selecting certain bits when generating a record of a specific type, and do not impact the distribution of different types of records.

The distribution of records is determined by the parameters $[p_1, p_2, \ldots, p_K]$ [30]. Specifically, it employs *K* parameters to control the generation probabilities of *K* types of records, where a type *i* record has exactly *i* bits that differ from the hidden string (i.e., the original data). For each time to generate a record, there is a probability p_i that a type *i* record will be generated. The probabilities $[p_1, p_2, \ldots, p_K]$ are ordered such that p_1 corresponds to the probability of generating a record with one different bit, p_2 for two different bits, and so on. To make the generated NDB_s difficult to reverse with respect to the local search strategy, the parameters *K* and $[p_1, p_2, \ldots, p_K]$ need to satisfy the hardness condition in (7) [30].

$$\sum_{i=1}^{K} (K - 2i)p_i > 0 \tag{7}$$

C. Data Extraction

After generating NDB_s of the hidden string *s*, the next step is to perform data extraction on NDB_s. The process for data extraction is shown in Algorithm 2. In this algorithm, the input consists of the *NDB* of a software file, denoted as NDB_s, along with the corresponding class label *y*. Then, we denote the length of records in NDB_s as m (Line 1), and initialize two counting arrays, ***one*** and ***zero***, with all elements set to 0 (Lines 2-3). These two arrays serve the purpose of storing the frequency of '1' and '0' at each position across all records in NDB_s (Lines 4-9). For each record in NDB_s, if the *j*th bit in the record is '1', the value of one_j is incremented by 1 (Line 6). Similarly, if the bit is '0', the value of $zero_j$ is incremented by 1 (Line 7). After processing all records, the final result of the counting arrays ***one*** and ***zero*** is obtained. Then, the counting arrays, ***one*** and ***zero***, are concatenated to create data with 2*m* dimensions.

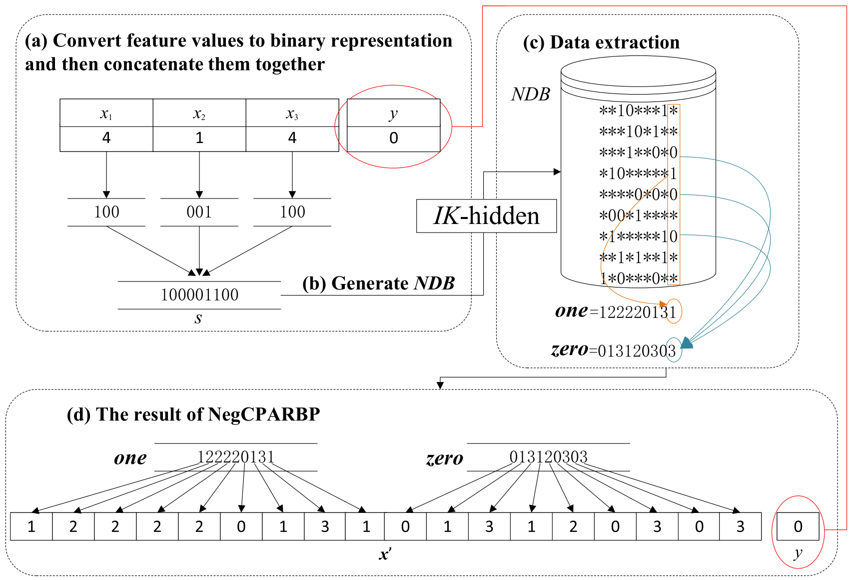

Figure 2. The example of NegCPARBP.

Algorithm 2: Data Extraction.

Input: NDB_s, class label y
Ouput: $\boldsymbol{x}'$
1: $m \leftarrow$ length of *record* in NDB_s;
2: Initialize *one*=[$one_1, \ldots\ one_m$]=[0,..., 0];
3: Initialize *zero*=[$zero_1, \ldots\ zero_m$]=[0,..., 0];
4: **for** each *record* in NDB_s:
5: **for** *j* from 1 $\rightarrow$ *m:*
6: if $record[j]$ == '1': one_j += 1;
7: if $record[j]$ == '0': $zero_j$ += 1;
8: **end for**
9: **end for**
10: $\boldsymbol{x}' = [one_1, \ldots\ one_m,\ zero_1, \ldots\ zero_m,\ y]$;
11: **return** $\boldsymbol{x}'$.

The original class label *y* is also included in this new data (Line 10). The extracted privacy-preserving data, denoted as $\boldsymbol{x}'$, is then used to replace the original data for sharing and ARB prediction.

D. Example

In this subsection, we provide illustrative examples of the NegCPARBP method and the *IK*-hidden algorithm to enhance comprehension. The examples provided here are merely for the convenience of readers to understand the algorithmic process. Hence, we have omitted the steps for max-min normalization and information gain calculation.

(1) NegCPARBP Method

First, we demonstrate the conversion process of the original software file data into privacy-preserving data using the NegCPARBP method, as shown in Figure 2.

- We assume that the feature vector of a software file is $(x_1, x_2, x_3) =$ (4, 1, 4), and the class label *y* is 0. The NegCPARBP method first performs max-min normalization on the feature vectors and converts them into binary strings. In this example, we omit the normalization step and directly convert the data (4, 1, 4) into binary strings as (100, 001, 100). By concatenating these binary strings, we obtain $s = 100001100$, as shown in step (a).
- Next, we generate the NDB_s of the string *s* using the *IK*-hidden algorithm. The detailed procedure of this step will be provided in the *IK*-hidden algorithm example.
- For demonstration purposes, we set the parameter *r* (controlling the size of *NDB*) to 1, yielding the NDB_s depicted in step (c). In the data extraction step, we count the number of each bit (0 or 1) from each record in the NDB_s. In the example depicted in step (c), the last bit of the fourth record is '1', while the bits of other records are not '1', resulting in $one_9 = 1$. The last bits of the third, fifth, and seventh records are '0', leading to $zero_9 = 3$. Similar counting procedures are applied to the other positions.
- Finally, in step (d), the $\boldsymbol{one}$ and $\boldsymbol{zero}$ arrays of length 9 are concatenated, and the original class label of the software file is added. This concatenation produces privacy-preserving data of length $2 \times 9 + 1$. This privacy-preserving data will replace the original data for sharing purposes, and it can effectively preserve the data utility for ARB prediction while masking private values.

(2) *IK*-hidden Algorithm

Figure 3 shows an illustrative example of the *IK*-hidden algorithm.

- In step (a), we start by generating a string consisting of asterisks '*' of equal length to the original string *s*.
- Moving on to step (b), we generate a random decimal value, denoted as *rndp*, within the range of [0, 1). Assuming the parameter $K = 3$ and $\boldsymbol{p}$=[0.35, 0.5, 0.15],

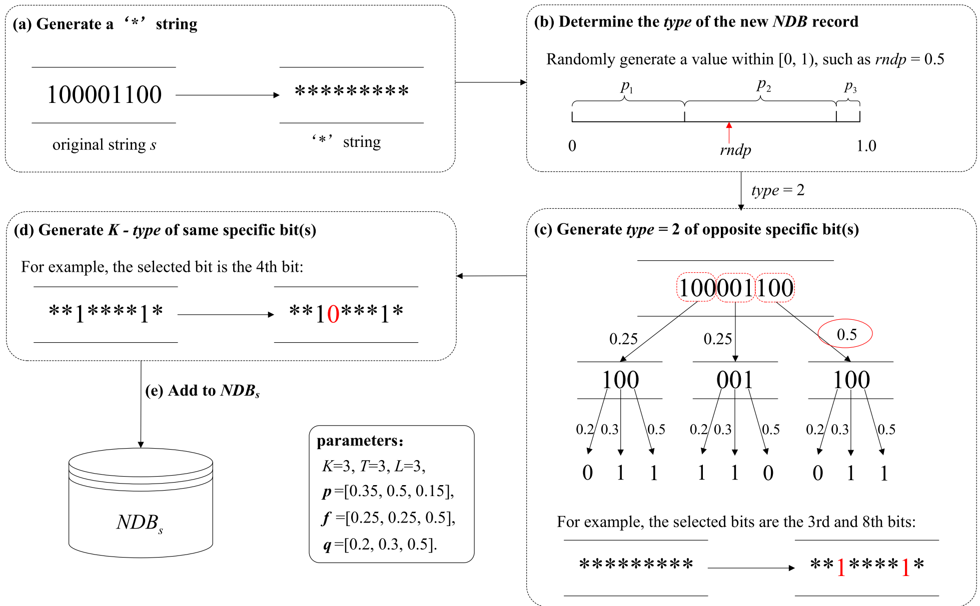

Figure 3. The example of *IK*-hidden.

given that $rndp = 0.5$, we determine that *rndp* falls within the interval $p_1 \leq rndp < p_1 + p_2$, indicating $type = 2$ (Similarly, if the random number *rndp* falls within the range [0, 0.35), i.e., $0 \leq rndp < p_1$, the NegCPARBP method will generate a record with one opposite bit and two same bits. If *rndp* falls within the range [0.85,1), satisfying $p_1 + p_2 \leq rndp < p_1 + p_2 + p_3$, the NegCPARBP method will generate a record with three opposite bits). In the currently generated record, there are *type*=2 determined bits that are opposite to the string s. Since $K = 3$, indicating there are three determined bits in each record, the remaining $K - type$ ($3 - 2 = 1$) determined bit is the same as the string s. Apart from these three bits, the remaining bits remain unchanged.

- In step (c), based on the number of features and the length of the binary representation of each feature, we set *T*=3 and *L*=3. Assuming the information gain values for these three features are [0.056, 0.244, 0.581], with an average value of 0.297. Since only the information gain of x_3 exceeds the average value, the probability parameter f_3 for x_3 is twice that of f_1 and f_2, and they sum up to 1, thus resulting in $\boldsymbol{f}$=[0.25, 0.25, 0.5]. According to the work of [25], different bits in the binary representation of features may capture varying information for classification. Since we set the same length L for the binary representation of each feature, padding with zeros in the higher bits for those shorter than L, we consider the data information to be more concentrated in the lower bits.
Therefore, we set $\boldsymbol{q}$=[0.5, 0.3, 0.2].
According to $\boldsymbol{f}$=[0.25, 0.25, 0.5], we know that during the random selection of features, the probability of choosing the 1st, 2nd, and 3 rd feature is 0.25, 0.25, and 0.5, respectively. Suppose we randomly select the first feature according to these probabilities. Then we need to randomly select a bit in the first feature. $\boldsymbol{q}$=[0.5, 0.3, 0.2] indicates that the probability of choosing the lowest, 2nd, and highest bit of this feature is 0.5, 0.3, and 0.2, respectively. Suppose we randomly select the lowest bit according to these probabilities (i.e., the third bit of the string s). Since the third bit of s is '0', the third bit in the record will be set to '1'. Similarly, we follow this process to generate another bit (e.g. the 8th bit) opposite to s in the record. Since the 8th bit of s is '0', the 8th bit of the generated record will be set to '1'.
- In step (d), we randomly select $K - type$ bits from the remaining uncertain bits with equal probability. Here, the $K - type$ ($3 - 2 = 1$) bit will be set to be the same as s. For instance, we choose the 4th bit of the record to be the same as the corresponding bit in the string s, which is '0'.
- Finally, as shown in step (e), the resulting record is added to NDB_s.
- We repeat steps (a-e) in Figure 3 until the number of records in NDB_s reaches $N = m \times r = 9$, thereby completing the *NDB* generation for the original string s.

E. Privacy Protection Capability Analysis

According to the work of [27], we analyze the probability of an attacker successfully attacking a single feature in the file processed by the NegCPARBP method. We also provide a comprehensive theoretical derivation of this process and outline the calculation procedure.

Since the *NDB* is generated based on a probability distribution, reverse attacks could be conducted using Bayes' theorem. Therefore, we assume that the attacker has access to the encrypted dataset and is familiar with all the steps and parameters of the *IK*-hidden algorithm to estimate the probability of successfully compromising the target feature within the target file.

From the attacker's perspective, his goal is to obtain the original, unprocessed value of a specific feature of a file. However, the attacker can only access the parameters of the NegCPARBP algorithm and the results of NegCPARBP, which are statistical data of NDB_s, as illustrated in step (d) of Figure 2. The attacker cannot access the intermediate data of the NegCPARBP method, such as the specific details of each record in NDB_s.

Therefore, based on the information available to the attacker, we have developed the following attack model.

(1) *Attack Method*

First of all, the attacker can calculate a $P_{diff}[i][j]$ for the *j*th bit of the *i*th feature in *IK*-hidden, which is the probability that the *j*th bit of the *i*th feature is different from the hidden string *s* when generating *NDB* records. According to the process of the *IK*-hidden algorithm, we can find that $P_{diff}[i][j]$ is controlled by the probability parameters f_i, q_j, and $\boldsymbol{p}$=$[p_1, p_2, \ldots, p_K]$, which can be calculated as Formula (8).

$$P_{diff}[i][j] = \frac{N_{diff}[i][j]}{N_{diff}[i][j] + N_{same}[i][j]} \tag{8}$$

where $N_{diff}[i][j]$ denotes the expected number of bits different from the hidden string *s* at the *j*th bit of the *i*th feature in the NDB_s, which can be calculated as Formula (9). Meanwhile, $N_{same}[i][j]$ denotes the expected number of bits same to the hidden string *s* at the *j*th bit of the *i*th feature in NDB_s, which can be calculated as Formula (10).

$$N_{diff}[i][j] = m \times r \times f_i \times q_j \times \sum_{k=1}^{K} p_k \times k \tag{9}$$

$$N_{same}[i][j] = m \times r \times T^{-1} \times L^{-1} \times \sum_{k=1}^{K} p_k \times (K-k) \tag{10}$$

Additionally, $P_{same}[i][j]$ is the probability that the *j*th bit of the *i*th feature is the same as the hidden string *s* when generating *NDB* records. It can be calculated as $P_{same}[i][j] = 1 - P_{diff}[i][j]$. The attacker can calculate the frequency of '0' and '1' in NDB_s to guess the value of $s[i][j]$ by (11).

$$\boldsymbol{s}[i][j] = \begin{cases} 0 & (n_0 > n_1 \ and \ P_{same}[i][j] > P_{diff}[i][j]) \ or \\ & (n_0 < n_1 \ and \ P_{same}[i][j] < P_{diff}[i][j]) \\ 1 & (n_0 < n_1 \ and \ P_{same}[i][j] > P_{diff}[i][j]) \ or \\ & (n_0 > n_1 \ and \ P_{same}[i][j] < P_{diff}[i][j]) \\ rand\{0,1\} & otherwise, \end{cases} \tag{11}$$

where n_0 is the number of records with '0' at the *j*th bit of the *i*th feature and n_1 is the number of records with '1' at the *j*th bit of the *i*th feature in NDB_s.

(2) *Success Rate of the Attack*

In this part, we calculate the success rate of the above attack method.

Assume $P(s[i][j] = 0)$ is the probability that the attacker infers $s[i][j] = 0$ based on the observation that there are n_0 records with '0' at the *j*th bit of the *i*th feature and n_1 records with '1' at the *j*th bit of the *i*th feature in NDB_s, then it can be calculated as:

$$P(s[i][j] = 0) = \frac{1}{1 + \left(\frac{T^{-1} \times L^{-1} \times \sum_{k=1}^{K} p_k \times (K-k)}{f_i \times q_j \times \sum_{k=1}^{K} p_k \times k}\right)^{n_1 - n_0}}. \tag{12}$$

The following is the derivation process of the Formula (12). Assume there are three events:

1) Event A_0: The attacker infers that the *j*th bit of the *i*th feature of the hidden string *s* is '0' (i.e., $s[i][j] = 0$).
2) Event A_1: The attacker infers that the *j*th bit of the *i*th feature of the hidden string s is '1' (i.e., $s[i][j] = 1$).
3) Event *B:* In NDB_s, there are n_0 records with '0' at the *j*th bit of the *i*th feature, and n_1 records with '1' at the *j*th bit of the *i*th feature.

In Bayes' theorem, the probability of the event "based on the observation that there are n_0 records with '0' at the *j*th bit of the *i*th feature, and n_1 records with '1' at the *j*th bit of the *i*th feature in NDB_s, the attacker infers that $s[i][j] = 0$" can be represented as $P(A_0|B)$. The probability of the event "given that the *j*th bit of the *i*th feature of the hidden string *s* is '0', obtaining n_0 records with '0' and n_1 records with '1' at the *j*th bit of the *i*th feature in NDB_s" can be represented as $P(B|A_0)$.

In NDB_s, there are $n_0 + n_1$ records with the jth bit of the ith feature being determined, either '0' or '1' (where NDB_s has a total of $m \times r$ records, and the remaining $m \times r - (n_0 + n_1)$ records have the jth bit of the ith feature marked with the compression symbol '*'). In the situation that event A_0 is observed (i.e. $s[i][j] = 0$), the *j*th bit of the *i*th feature is chosen for constructing records $n_0 + n_1$ times when generating the records of NDB_s, where the probability of generating '0' and '1' is $P_{same}[i][j]$ and $P_{diff}[i][j]$ respectively at each time, and we suppose this process satisfies the binomial distribution. Therefore, the total number of cases generating n_0 '0's and n_1 '1's is $C_{n_0+n_1}^{n_0}$ (binomial coefficient), and the probability $P(B|A_0)$ is $C_{n_0+n_1}^{n_0} \times (P_{same}[i][j])^{n_0} \times (P_{diff}[i][j])^{n_1}$. Similarly, $P(B|A_1)$ can be calculated as:

$$P(B|A_1) = C_{n_0+n_1}^{n_1} (P_{same}[i][j])^{n_1} \times (P_{diff}[i][j])^{n_0}. \tag{13}$$

Consequently, according to Bayes' theorem, we can calculate $P(A_0|B)$ as:

$$P(A_0|B) = \frac{P(A_0)P(B|A_0)}{P(A_0)P(B|A_0) + P(A_1)P(B|A_1)}, \tag{14}$$

where $P(A_0)$ and $P(A_1)$ represent the prior probability of the attacker inferring that the *j*th bit of the *i*th feature of the hidden string *s* is '0' and '1', respectively. In the case that the attacker has no prior knowledge about the hidden string *s*, we assume that from the attacker's perspective, each bit follows a uniform distribution, meaning that each bit has an equal probability of being '0' or '1'. Thus, the prior probabilities should be equal, and we have $P(A_0) = P(A_1) = 1/2$.

Hence, we derive $P(A_0|B)$, representing $P(s[i][j] = 0)$, which simplifies to the form shown in Formula (12).

Similarly, the probability of the *j*th bit of the *i*th feature of the hidden string being '1' can be calculated as:

$$P(s[i][j]=1)=\frac{1}{1+\left(\frac{T^{-1}\times L^{-1}\times\sum_{k=1}^{K}p_k\times(K-k)}{f_i\times q_j\times\sum_{k=1}^{K}p_k\times k}\right)^{n_0-n_1}}. \quad (15)$$

Therefore, if the actual value of the target feature *s*[*i*] is ***b***, the probability that the attacker successfully guesses *s*[*i*] according to Formula (11) can be calculated by Formula (16):

$$P(s[i]=\boldsymbol{b})=\prod_{j=1}^{L}P(s[i][j]=b[j]), \quad (16)$$

where *b*[*j*] represents the *j*th bit of the binary representation of ***b***.

The success probability can be used for evaluating the privacy protection capability of NDB_s.

IV. Experiments Setup

A. Research Questions

To examine the performance of data privacy protection and the ability to maintain data utility (the capability to accurately predict ARB-prone and ARB-free files) of the NegCPARBP method, we organize the experiments based on the following two Research Questions (RQs):

RQ1: Does NegCPARBP improve privacy protection capabilities?

To assess the privacy-preserving capability of NegCPARBP, we employ the methods detailed in Section E to simulate attacks on the data protected by NegCPARBP, computing the probability of our method successfully resisting attacks. Additionally, we compare the privacy protection effectiveness of the proposed *IK*-hidden method with three existing CPDP privacy-preserving methods (MORPH [15], LACE [16], and SRDO [18]) and the latest *NDB* generation method (*QK*-hidden [25]).

RQ2: Can NegCPARBP better maintain data utility compared to existing methods?

We propose the NegCPARBP method, which transforms the original feature vector $\boldsymbol{x}$ of a software file into a new feature vector $\boldsymbol{x}'$ to achieve privacy protection. To investigate whether the data obtained by NegCPARBP can be effectively used to train an ARB prediction model (i.e., maintain data utility), we compare NegCPARBP with MORPH [15], LACE [16], SRDO [18], and *QK*-hidden [25].

B. Dataset

The experiments are conducted on three datasets: Linux[2], MySQL[3], and NetBSD[4], renowned for their large, complex, and long-running software systems. These datasets are widely used for CPARBP in previous studies [1], [6]. Linux is a famous open-source operation system. MySQL is a well-known database with lots of users. NetBSD is a free and exceptionally portable open-source operating system rooted in UNIX. The ARBs in these datasets are all from long-running systems, which can lead to performance degradation and eventual system crashes [1].

[2]Linux: http://www.kernel.org
[3]MySQL: http://www.mysql.com
[4]NetBSD: http://www.netbsd.org

TABLE II
The Details of the Experimental Datasets

Project	Files	ARB-prone files	ARB-prone files%
Linux	3400	20	0.59%
MySQL	470	39	8.30%
NetBSD	1731	21	1.21%

In Table II, "Files" represents the number of files in the dataset, "ARB-prone files" signifies the total number of files that contain ARBs, and "ARB-prone files%" denotes the percentage of files that contain ARBs.

The datasets comprise 82 features, which can be divided into four types: program size, McCabe's complexity, Halstead features, and aging-related features, as shown in Table III. Cotroneo et al. [10] defined the six aging-related features aimed at enhancing ARB prediction performance, which are presented at the end of Table III. Specifically, *AllocOps* and *DeallocOps* refer to counts of memory allocation and deallocation operations, respectively. *DerefSet* and *DerefUse* represent counts of pointer variable dereferences during reading and writing operations. *UniqueDerefSet* and *UniqueDerefUse* are used to measure the unique dereference sets and use sets in the software, respectively. A detailed description of all the features used in our study can be found in [32].

C. Attack Strategies for Evaluating Privacy Protection Baselines

To assess the privacy protection capabilities of MORPH, LACE, and SRDO, the authors in [16], [18] employed the Increased Privacy Ratio (IPR) [16] to calculate the probability of the attacker obtaining the original value of a feature. A higher IPR denotes superior privacy protection, with IPR=1 indicating resistance against all attacks and IPR=0 suggesting vulnerable to any attack. In the process of calculating the IPR, they divide feature values into 10 bins using Equal Frequency Binning (EFB) [33]. Based on EFB, the target feature can be represented as S=[$s_1, s_2, \ldots, s_{10}$]. Given queries $Q=\{q_1,q_2,\ldots,q_{|Q|}\}$, where q_i represents an attack (i.e. query) such as $\{F_{know1}=$ [1-2], $F_{know2}=$ (4-6]$\}$, and the IPR can be calculated as Formula (17):

$$IPR=1-\frac{1}{|Q|}\sum_{i=1}^{|Q|}Breach(S,G_i^*), \quad (17)$$

where G_i^* is a group of subranges of target feature of files from a dataset which matches the attack q_i, and $Breach(S,G_i^*)$ can be calculated as Formula (18):

$$Breach(S,G_i^*)=\begin{cases}1, & \text{if } s_{max}(G_i)=s_{max}(G_i'),\\ 0, & \text{otherwise,}\end{cases} \quad (18)$$

where G_i is the group from the original data, G_i' is the group from the encrypted data, and $s_{max}(G_i^*)$ is the most common target feature value in G_i^*. For example, if $G_i^*=\{$[1-3], [1-3], (4-5]$\}$

TABLE III
THE SUMMARY OF FEATURES IN ARB DATASETS

Program size	AltAvgLineBlank, AltAvgLineCode, AltAvgLineComment, AltCountLineBlank, AltCountLineCode, AltCountLineComment, AvgCyclomatic, AvgCyclomaticModified, AvgCyclomaticStrict, AvgEssential, AvgLine, AvgLineBlank, AvgLineCode, AvgLineComment, CountClassBase, CountClassCoupled, CountClassDerived, CountDeclClass, CountDeclClassMethod, CountDeclClassVariable, CountDeclFunction, CountDeclInstanceMethod, CountDeclInstanceVariable, CountDeclInstanceVariablePrivate, CountDeclInstanceVariableProtected, CountDeclInstanceVariablePublic, CountDeclMethod, CountDeclMethodAll, CountDeclMethodConst, CountDeclMethodFriend, CountDeclMethodPrivate, CountDeclMethodProtected, CountDeclMethodPublic, CountInput, CountLine, CountLineBlank, CountLineCode, CountLineCodeDecl, CountLineCodeExe, CountLineComment, CountLineInactive, CountLinePreprocessor, CountOutput, CountPath, CountSemicolon, CountStmt, CountStmtDecl, CountStmtEmpty, CountStmtExe, RatioCommentToCode, PercentLackOfCohesion
McCabe complexity	Cyclomatic, CyclomaticModified, CyclomaticStrict, MaxCyclomatic, MaxCyclomaticModified, MaxCyclomaticStrict, MaxEssentialKnots, MaxInheritanceTree, MaxNesting, MinEssentialKnots, SumCyclomatic, SumCyclomaticModified, SumCyclomaticStrict, SumEssential, Essential, Knots
Halstead features	n1, n2, N1, N2, Program Length, Program Vocabulary, Program Volume, Difficulty, Effort
Aging-related features	AllocOps, DeallocOps, DerefUse, UniqueDerefUse, DerefSet, UniqueDerefSet

represents the results of the *i*th attack returned from a dataset, then $s_{max}(G_i^*) = \{[1\text{-}3]\}$.

Thus, the process of an attack is as follows:

1) Given a set of features, F (i.e., all the features except for the target feature and class label), and all their possible subranges (created using EFB with 10 bins for each attribute in the dataset), we randomly select k feature(s) from F. For example, if $k = 2$, the selected features and their respective subranges could be F_{know1} with {[1-2], [4-9], (13-14]} and F_{know2} with {(4-6], (7-9]}.
2) Based on the original dataset, we randomly select a file M. From its feature vector **x**, we identify the values corresponding to the selected features F_{know1} and F_{know2}. For example, if the feature vector x for file M contains $F^M_{know1} = 1$ and $F^M_{know2} = 4.5$, then we have $F'_{know1} = [1{-}2]$ and $F'_{\text{know2}} = (4{-}6]$.

According to this procedure, we generate the attack $q = \{F'_{know1} = [1{-}2], F'_{know2} = (4{-}6]\}$. Suppose the attacker uses the partially known feature values to search within the encrypted data, satisfying the condition in the Formula (18) (i.e., the interval for F_{target} searched by the attacker in the encrypted data matches the interval for F_{target} obtained from the original dataset using attack q), suggesting that the attack is successful.

We will use the methods mentioned above to test how well MORPH, LACE, and SRDO protect privacy. At the same time, we will apply the methods in Formulas (11), (12), (15), and (16) to evaluate how well NegCPARBP protects privacy.

D. Evaluation Metrics

We employ four metrics, *PD*, *PF*, *G-measure*, and *Balance*, to assess the ability to accurately predict ARB-prone and ARB-free files, as these metrics are recommended for software engineering tasks involving imbalanced data classification [34], [35], [36].

These metrics can be computed using the confusion matrix as illustrated in Table IV. TP represents the number of predicted ARB-prone files that are genuinely ARB-prone, while FP denotes the number of predicted ARB-prone files that are actually ARB-free. FN records the number of predicted ARB-free files that are genuinely ARB-prone, and TN records the number of predicted ARB-free files that are truly ARB-free.

TABLE IV
THE CONFUSION MATRIX OF ARB PREDICTION

	Actual ARB-prone	Actual ARB-free
Predicted ARB-prone	TP	FP
Predicted ARB-free	FN	TN

(1) *PD* (Probability of Detection) represents the proportion of files correctly predicted as ARB-prone among those that are actually ARB-prone:

$$PD = \frac{TP}{TP + FN}. \tag{19}$$

(2) *PF* (Probability of False alarms) represents the proportion of files incorrectly predicted as ARB-prone among those that are actually ARB-free:

$$PF = \frac{FP}{FP + TN}. \tag{20}$$

(3) *G-measure* is the harmonic mean of the proportions of true positives (*PD*) and true negatives ($1 - PF$) within the total samples:

$$G\text{-}measure = \frac{2 \times PD \times (1 - PF)}{PD + (1 - PF)}. \tag{21}$$

(4) *Balance* is determined by computing the euclidean distance from the actual (*PF, PD*) point to (0, 1), where the point (*PF*=0, *PD*=1) represents the optimal position on the ROC curve, indicating perfect ARBs recognition:

$$Balance = 1 - \frac{\sqrt{(0 - PD)^2 + (1 - PF)^2}}{\sqrt{2}}. \tag{22}$$

E. Statistic Test

The Wilcoxon signed-rank test [37] is a non-parametric sample test, which is used to compare pairs of results and is able to compare the difference against zero. The null hypothesis of the Wilcoxon signed-rank test posits that the methods have no significant difference, with a predefined significance level of 0.05. If the p-value is less than 0.05, the null hypothesis is rejected, indicating that there exists statistical significance between the pairwise methods. Otherwise, the null hypothesis cannot be rejected. Then if the test shows a significant difference, we employ Cliff's δ [38] to examine whether the magnitude of the difference is of practical importance or not. The effect size is considered negligible (0 <|Cliff's δ | <0.147), small (0.147 $\leq$ |Cliff's δ | <0.33), medium (0.33 $\leq$ |Cliff's δ | <0.474), or large (|Cliff's δ | $\geq$ 0.474), respectively. In summary, a method performs significantly better or worse than another method, if the p-value is less than 0.05 and the effect size is not negligible based on Cliff's δ. The difference between the two methods is not of practical importance, if the p-value is not less than 0.05 or the p-value is less than 0.05 and the effect size is negligible (less than 0.147) [39].

F. Experimental Design

(1) The NegCPARBP method normalizes and converts the feature values of all files to the range [0, 1]. We multiply each normalized feature value by 10^8, discard the decimal part to convert it into an integer, and then represent it in binary with a length of 27 bits. Since the datasets have *T*=82 features, the length of string *s* is $m = 82 \times 27 = 2214$. For the parameters in *IK*-hidden, we set *K*=3, *r*=15, and $\boldsymbol{p}$=[p_1, p_2, p_3]=[0.752, 0.226, 0.022] as default, satisfying Formula (7) to make the generated *NDB* difficult to reverse w.r.t the local search strategy. Additionally, according to [25], we set $q_i = 2q_j$ ($1 \leq i \leq L/2$, $L/2 < j \leq L$), while ensuring that $\sum_{i=1}^{L} q_i = 1$.

(2) The experimental setup in this study follows the general steps of cross-project experiments [6], where two datasets are combined as a training set, and the remaining one serves as the test set. To address data distribution disparities between the training and test sets, as well as the class imbalance problem, we apply the classic CPARBP method TLAP [6] with three distinct classifiers, Naive Bayes (NB) [40], Support Vector Machines (SVM) [41], and Random Forest (RF) [42]. TLAP combines Transfer Component Analysis (TCA) with random oversampling to alleviate data distribution differences and class imbalance issues. Since the training and testing data come from different projects with different data distributions, TCA minimizes these differences by transforming the data into a common feature space where they are more similar. This improves the accuracy of predictions, and makes the model to generalize better from the training data to the testing data. To mitigate potential biases during hyperparameter tuning, we repeat the entire process 10 times. These ten-run results are utilized for statistical significance analysis using the Wilcoxon signed-rank test [37] and Cliff's δ [38]. The average of ten results is shown in Tables VI, VII, VIII, and IX.

V. Experimental Results

A. RQ1: Does NegCPARBP Improve Privacy Protection Capabilities?

Methods: Following previous privacy protection studies [16], [18], we design a simulated attack scenario to evaluate the resilience of our encrypted dataset against attacks. In this scenario, we assume that the attacker possesses knowledge of the NegCPARBP algorithm and all its parameters, as well as access to the encrypted dataset. Additionally, we assume that the attacker has some knowledge of the original value (i.e., unprocessed values) of a single feature F_1 (i.e., $F_1 = value_{F_1}$) of a software file M from the original dataset. The attacker's objective is to obtain (attack) the original value of another feature (referred to as target feature F_{target}) of the file M from the encrypted dataset. Therefore, the attacker undertakes the following steps:

1) The attacker needs to obtain the original values of the feature F_1 for all files in the encrypted dataset.
2) The attacker uses the knowledge that the file *M* has $F_1 = value_{F_1}$ to find all files with $F_1 = value_{F_1}$ in the encrypted dataset, resulting in files $\{M_1', M_2', \ldots\}$.
3) The attacker retrieves the values of the target feature F_{target} from the files $\{M_1', M_2', \ldots\}$.

In the above steps, each time the attacker obtains the original value of a feature from the data encrypted by NegCPARBP, they must perform an attack using Formulas (11), (12), (15), and (16). In step 2, if the list of files obtained by the attacker contains only one file (i.e., $\{M_1'\}$), it indicates that the attacker can use the original value of only one feature of the target file M for a successful attack, representing the scenario, where the number of known features required for the attack is *k*=1. If multiple files in the encrypted dataset have the same value of feature F_1, the attacker cannot identify which file matches the target file *M*. At this point, we assume the attacker knows the original value of another feature F_2 in file *M*. Then, the attacker simultaneously uses both F_1 and F_2 to conduct attacks (representing the case of *k*=2). If there are still multiple matching files even using two known features, we consider that the attacker may need to know the values of more features (representing the case of *k*>2) to identify the target file *M*.

The probability of successfully attacking F_1 (i.e., obtaining the original values of the feature F_1) can be represented as P_1, for F_2 it is P_2, and for F_3 it is P_3. Assuming the probability of successfully attacking the target feature F_{target} is P_{target}, the probability of successfully obtaining the original value of the target feature of file *M* using the original value of a single feature F_1 can be calculated as $P_1 \times P_{target}$. Using two known features, F_1 and F_2, the probability of a successful attack can be calculated as $P_1 \times P_2 \times P_{target}$. Similarly, for all three, it can be calculated as $P_1 \times P_2 \times P_3 \times P_{target}$.

Furthermore, we employ the method outlined in Formula (17) to compute the privacy protection capabilities of three CPDP privacy-preserving methods (MORPH, LACE, and SRDO).

In simulated attacks, within CPDP datasets, attackers target the "LineOfCode" feature according to [16], [18]. Correspondingly, in CPARBP datasets, we set the "CountLineCode" feature as the attacker's target feature.

Result: We can draw the following conclusions from Table V:

1) When compared to the three methods (MORPH, LACE, and SRDO) in CPDP, which achieve privacy protection rates between 0.898 and 0.907, *IK*-hidden consistently achieves privacy protection rates exceeding 0.97 across various scenarios. Hence, it can be inferred that *IK*-hidden

TABLE V
THE PRIVACY-PRESERVING CAPABILITIES OF MORPH, LACE, SRDO, *QK*-HIDDEN, AND *IK*-HIDDEN

k	Methods	Linux	MySQL	NetBSD	Average
1	MORPH	0.883	0.906	0.912	0.900
	LACE	0.914	0.887	0.893	0.898
	SRDO	0.896	0.920	0.904	0.907
	QK-hidden	**0.999**	**0.999**	**0.999**	**0.999**
	IK-hidden	0.979	0.971	0.970	0.973
2	MORPH	0.912	0.906	0.903	0.907
	LACE	0.907	0.902	0.900	0.903
	SRDO	0.895	0.904	0.910	0.903
	QK-hidden	**0.999**	**0.999**	**0.999**	**0.999**
	IK-hidden	0.996	0.994	0.994	0.995
3	MORPH	0.905	0.896	0.900	0.900
	LACE	0.903	0.895	0.904	0.901
	SRDO	0.904	0.900	0.905	0.903
	QK-hidden	**0.999**	**0.999**	**0.999**	**0.999**
	IK-hidden	**0.999**	0.998	0.998	0.998

significantly enhances privacy protection compared to MORPH, LACE, and SRDO.

2) When using *IK*-hidden and *QK*-hidden as *NDB* generation algorithms in the NegCPARBP method, the probability of effectively preventing attackers from accessing feature values consistently surpasses 0.97. Based on the average privacy protection rates across the three datasets, although the *IK*-hidden algorithm demonstrates inferior privacy protection capability compared to *QK*-hidden, its overall performance remains highly satisfactory.

3) After processing with the *IK*-hidden method, it becomes exceedingly difficult for attackers to obtain the original value when the attacker uses one feature (k=1) for simulating attacks, with privacy protection performance reaching 0.973, i.e., the attacker fails 973 out of 1000 attempts to access the original value of the target feature in the target file. Meanwhile, when k=2 (0.995) or k=3 (0.998), the attack becomes even more challenging. If the attacker simultaneously uses k features for simulating attacks, the probability of success is $P_{target} \times P_1 \times \ldots \times P_k$. It is evident that the probability of a successful attack will increase exponentially with the number of features used by the attacker. Hence, the *IK*-hidden method demonstrates robust privacy protection effectiveness.

Answer to RQ1: While the privacy protection capability of the *IK*-hidden algorithm is slightly lower than that of the *QK*-hidden algorithm, it still exceeds 0.97. Moreover, our method achieves better privacy protection capabilities than existing privacy protection methods in CPDP (their privacy protection capabilities range from 0.898 to 0.907).

B. RQ2: Can NegCPARBP Better Maintain Data Utility Compared to Existing Methods?

Methods: We analyze the results of CPARBP using the original data and five methods in terms of the four evaluation metrics

TABLE VI
THE *PD* VALUES OF THE SIX METHODS (ORIGINAL DATA, IK: NEGCPARBP WITH IK-HIDDEN, QK: NEGCPARBP WITH QK-HIDDEN, MORPH, LACE, AND SRDO) ON THE THREE DATASETS (LINUX, MYSQL, AND NETBSD) UNDER THE THREE CLASSIFIERS (NB, SVM, AND RF)

Classifier	Dataset	Original	IK	QK	MORPH	LACE	SRDO
NB	Linux	0.750	0.805	**0.945**	0.145	0.265	0.470
	MySQL	0.308	0.756	**0.854**	0.003	0.077	0.190
	NetBSD	0.524	**0.786**	0.310	0.038	0.138	0.343
	Average	0.527	**0.782**	0.703	0.062	0.160	0.334
SVM	Linux	0.800	0.930	**0.950**	0.000	0.325	0.285
	MySQL	0.744	0.826	**0.849**	0.136	0.244	0.364
	NetBSD	0.667	**0.805**	0.671	0.200	0.286	0.400
	Average	0.737	**0.853**	0.823	0.112	0.285	0.350
RF	Linux	**0.750**	0.655	0.625	0.025	0.160	0.400
	MySQL	0.436	0.641	**0.872**	0.097	0.262	0.392
	NetBSD	0.476	**0.681**	0.362	0.129	0.238	0.310
	Average	0.554	**0.659**	0.620	0.084	0.220	0.367

TABLE VII
THE *PF* VALUES OF THE SIX METHODS ON THE THREE DATASETS UNDER THREE CLASSIFIERS

Classifier	Dataset	Original	IK	QK	MORPH	LACE	SRDO
NB	Linux	0.236	0.260	0.322	**0.140**	0.201	0.455
	MySQL	0.084	0.253	0.336	0.091	**0.058**	0.184
	NetBSD	0.183	0.335	0.115	**0.081**	0.141	0.216
	Average	0.168	0.283	0.258	**0.104**	0.133	0.285
SVM	Linux	0.304	0.327	0.335	**0.127**	0.195	0.310
	MySQL	**0.260**	0.296	0.318	0.587	0.427	0.312
	NetBSD	0.287	0.372	0.271	0.277	0.245	**0.207**
	Average	0.284	0.331	0.308	0.330	0.289	**0.277**
RF	Linux	0.289	0.186	0.136	**0.091**	0.159	0.308
	MySQL	**0.114**	0.167	0.358	0.508	0.236	0.365
	NetBSD	0.200	0.282	0.143	**0.104**	0.191	0.154
	Average	0.201	0.212	0.212	0.234	**0.196**	0.276

TABLE VIII
THE *G-MEASURE* VALUES OF THE SIX METHODS ON THE THREE DATASETS UNDER THREE CLASSIFIERS

Classifier	Dataset	Original	IK	QK	MORPH	LACE	SRDO
NB	Linux	0.757	0.770	**0.789**	0.236	0.359	0.466
	MySQL	0.461	**0.751**	0.746	0.005	0.138	0.278
	NetBSD	0.638	**0.720**	0.456	0.072	0.233	0.459
	Average	0.619	**0.747**	0.664	0.104	0.243	0.401
SVM	Linux	0.744	0.781	**0.782**	0.000	0.390	0.365
	MySQL	0.742	**0.760**	0.756	0.204	0.324	0.429
	NetBSD	0.689	**0.705**	0.698	0.309	0.380	0.501
	Average	0.725	**0.749**	0.745	0.171	0.365	0.432
RF	Linux	**0.730**	0.719	0.720	0.038	0.233	0.488
	MySQL	0.584	0.724	**0.733**	0.159	0.354	0.432
	NetBSD	0.597	**0.697**	0.505	0.221	0.348	0.402
	Average	0.637	**0.713**	0.653	0.140	0.312	0.441

under three classifiers. Tables VI, VII, VIII, and IX show the detailed *PD*, *PF*, *G-measure*, and *Balance* values across Linux, MySQL, and NetBSD datasets. The column "O" signifies using the original training set without involving any privacy protection process. The columns "IK" and "QK" represent NegCPARBP methods with *IK*-hidden and *QK*-hidden algorithms, respectively. Meanwhile, the columns "M", "L", and "S" correspond to MORPH, LACE, and SRDO methods, respectively. Furthermore, the bold part represents the best value among all comparison methods. The results highlighted in green indicate that *IK*-hidden performs significantly better than the corresponding methods. Conversely, the results highlighted in red indicate that *IK*-hidden is significantly worse than the corresponding methods. For results without color annotation, *IK*-hidden is not significantly better or worse than the corresponding methods.

(1) *PD Result:* As shown in Table VI, it is evident that the *IK*-hidden method consistently demonstrates significantly better *PD*

TABLE IX
THE *BALANCE* VALUES OF THE SIX METHODS ON THE THREE DATASETS UNDER THREE CLASSIFIERS

Classifier	Dataset	Original	IK	QK	MORPH	LACE	SRDO
NB	Linux	0.757	0.768	**0.769**	0.386	0.455	0.489
	MySQL	0.507	**0.751**	0.739	0.290	0.345	0.408
	NetBSD	0.639	**0.718**	0.505	0.317	0.380	0.507
	Average	0.634	**0.746**	0.671	0.331	0.393	0.468
SVM	Linux	0.743	**0.763**	0.761	0.287	0.488	0.444
	MySQL	0.742	**0.757**	0.750	0.261	0.372	0.489
	NetBSD	0.689	**0.703**	0.698	0.398	0.453	0.544
	Average	0.724	**0.741**	0.736	0.315	0.437	0.493
RF	Linux	**0.730**	0.720	0.715	0.307	0.390	0.511
	MySQL	0.593	0.720	**0.723**	0.266	0.437	0.490
	NetBSD	0.604	**0.697**	0.537	0.379	0.439	0.493
	Average	0.642	**0.712**	0.658	0.317	0.422	0.498

values compared to the method using original data, except when using the RF classifier on the Linux dataset. Across all three classifiers, the average *PD* values of *IK*-hidden outperform that of the original method by 15.7%–48.4%. Furthermore, the *IK*-hidden method exhibits significantly better *PD* values compared to MORPH, LACE, and SRDO. On the NB classifier, the average *PD* value of *IK*-hidden across all three datasets surpasses those of MORPH, LACE, and SRDO by 134.1%–1161.3%. Similarly, on the SVM classifier, *IK*-hidden outperforms MORPH, LACE, and SRDO by 143.7%–661.6%. On the RF classifier, *IK*-hidden achieves an average *PD* value higher than those of MORPH, LACE, and SRDO by 79.6%–684.5%. Compared to the *QK*-hidden method, *IK*-hidden may exhibit lower *PD* values in some cases, but its average performance across all three datasets is superior to *QK*-hidden. Specifically, *IK*-hidden surpasses *QK*-hidden by 11.2%, 3.6%, and 6.3% on the NB, SVM, and RF classifiers, respectively.

(2) *PF Result:* For the *PF* metric, a higher value indicates a higher likelihood of misidentifying ARB-free files as ARB-prone. Therefore, lower *PF* values indicate better results. As shown in Table VII, for the NB classifier, *IK*-hidden has the second worst average *PF* value across the three datasets. *IK*-hidden only significantly outperforms SRDO on Linux and is superior to *QK*-hidden on both Linux and MySQL. In most other cases, *IK*-hidden performs significantly worse. For the SVM classifier, *IK*-hidden achieves the worst average *PF* value. *IK*-hidden is only significantly superior to *QK*-hidden and MORPH on MySQL. In most other cases, *IK*-hidden performs significantly worse. For the RF classifier, *IK*-hidden achieves the third worst average *PF* value. *IK*-hidden method demonstrates significant superiority over the original method and SRDO on Linux, as well as over the *QK*-hidden, MORPH, and SRDO on MySQL. Additionally, *IK*-hidden exhibits no significant difference compared to LACE across all three datasets. In other cases, *IK*-hidden performs significantly worse on the RF classifier. Overall, across all three datasets, *IK*-hidden consistently exhibits significantly worse performance when compared to other methods.

(3) *G-measure and Balance Result:* In Tables VIII and IX, the *IK*-hidden method consistently shows significantly better *G-measure* and *Balance* values compared to the original method, except when using the NB and RF classifiers on the Linux dataset. Across all three classifiers, the average *G-measure* and *Balance* values of *IK*-hidden outperform the original method by 3.3%–20.7% and 2.3%–17.7%, respectively. Additionally, *IK*-hidden exhibits significantly better *G-measure* and *Balance* values compared to MORPH, LACE, and SRDO. On the NB classifier, the average *G-measure* value of *IK*-hidden across all three datasets outperforms those of MORPH, LACE, and SRDO by 86.3%–618.3%, and the average *Balance* value of *IK*-hidden outperforms those of MORPH, LACE, and SRDO by 59.4%–125.4%. Similarly, on the SVM classifier, *IK*-hidden outperforms MORPH, LACE, and SRDO by 73.4%–338.0% on *G-measure* and surpasses MORPH, LACE, and SRDO by 50.3%–135.2% on *Balance*. On the RF classifier, *IK*-hidden achieves an average *G-measure* value higher than those of MORPH, LACE, and SRDO by 62%–409%, and achieves an average *Balance* value higher by 42%–125%. Compared to the *QK*-hidden method, *IK*-hidden exhibits lower *G-measure* or *Balance* values in some scenarios, but its average *G-measure* and average *Balance* performance across all three datasets are superior to *QK*-hidden. Specifically, for *G-measure*, *IK*-hidden surpasses *QK*-hidden by 12.5%, 0.5%, and 9.2% on NB, SVM, and RF classifiers. For *Balance*, *IK*-hidden outperforms *QK*-hidden by 11.2%, 0.7%, and 8.2% on the NB, SVM, and RF classifiers, respectively.

(4) *Summary:* Although the *IK*-hidden method tends to predict more ARB-free files as ARB-prone compared to other methods based on the analysis of *PF* values, the primary objective of CPARBP is to accurately identify ARB-prone files to enhance system reliability. Therefore, when considering the combined analysis of *PD*, *PF*, *G-measure*, and *Balance*, the increase in *PF* value by the *IK*-hidden method is deemed acceptable. Overall, in terms of maintaining data utility, we can conclude that the *IK*-hidden method outperforms *QK*-hidden, MORPH, LACE, and SRDO. The main reason for better data utility of *IK*-hidden can be attributed to its negative database generation process, which aims to generate the complement of the original feature vector that can perform operations similar to those of the original feature vector. In addition, *IK*-hidden can capture more information about important features (measured by information gain) for CPARBP. By doing so, it ensures that the overall distribution and characteristics of the features in the original data are preserved. This preservation of feature distribution ensures that the privacy-protected data retains the essential information for the accurate prediction of ARBs.

Answer to RQ2: NegCPARBP can better maintain data utility compared to existing methods in terms of *PD*, *G-measure*, and *Balance*.

VI. DISCUSSION

A. The Impact of Feature Selection

Since the ARB dataset contains 82 features to describe each software file, there may be irrelevant or redundant features that affect the accuracy of predicting software aging bugs. Therefore, we use the feature selection method SVMF [43], which is shown to perform the best among 22 feature selection methods for the ARB dataset, according to Zhang et al.'s study [44]. Following Zhang et al.'s approach, when using two datasets for training

TABLE X
THE *PD*, *PF*, *G-MEASURE*, AND *BALANCE* VALUES WHEN TRAINING ON THE DATASETS PROCESSED WITH SVMF AND THE NEGCPARBP

Classifier	Dataset	*PD*	*PF*	*G-measure*	*Balance*
NB	Linux	0.850 ↑	0.257 ↓	0.793 ↑	0.790 ↑
	MySQL	0.641 ↓	0.149 ↓	0.731 ↓	0.725 ↓
	NetBSD	0.190 ↓	0.153 ↓	0.311 ↓	0.417 ↓
	Average	0.560 (↓28.4%)	0.186 (↓34.1%)	0.612 (↓18.1%)	0.644 (↓13.6%)
SVM	Linux	0.850 ↓	0.257 ↓	0.793 ↑	0.789 ↑
	MySQL	0.846 ↑	0.347 ↑	0.737 ↓	0.732 ↓
	NetBSD	0.571 ↓	0.315 ↓	0.623 ↓	0.624 ↓
	Average	0.756 (↓11.5%)	0.306 (↓7.6%)	0.718 (↓4.1%)	0.715 (↓3.5%)
RF	Linux	0.650 ↓	0.161 ↓	0.732 ↑	0.727 ↑
	MySQL	0.513 ↓	0.102 ↓	0.653 ↓	0.648 ↓
	NetBSD	0.286 ↓	0.135 ↓	0.430 ↓	0.486 ↓
	Average	0.483 (↓26.7%)	0.133 (↓37.3%)	0.605 (↓15.2%)	0.620 (↓12.9%)

and another for testing, we apply SVMF separately to each of the two training datasets. SVMF ranks features based on their classification performance, with higher-ranked features being considered more important. We select the top 20 features from each of the two training datasets, and the union of these features forms the final set selected by SVMF. We then apply the NegCPARBP method to the feature-selected training set for privacy protection. The test dataset is also restricted to these selected features. We repeat the entire process 10 times. Table X presents the average *PD*, *PF*, *G-measure*, and *Balance* values when training on the datasets processed with feature selection and NegCPARBP. These results are compared with those in Section B, where feature selection is not used. In Table X, an upward arrow indicates an increase in the metric value with SVMF feature selection, while a downward arrow indicates a decrease. Data highlighted in green indicate that using NegCPARBP together with SVMF significantly outperforms the case without SVMF. Conversely, data highlighted in red indicate that NegCPARBP with SVMF performs significantly worse than NegCPARBP without SVMF. For methods without any color annotation, NegCPARBP with SVMF shows no significant difference in performance compared to NegCPARBP without SVMF.

The *PD* values across all three classifiers decrease significantly, with two exceptions: the *PD* values for the NB classifier on the Linux dataset and the SVM classifier on the MySQL dataset. The *PF* values for all three classifiers decrease, except for the SVM classifier on the MySQL dataset, which shows an increase. For *G-measure* and *Balance*, the values across all three classifiers also decrease significantly, except on the Linux dataset, where both metrics increase.

Overall, the results suggest that using the SVMF feature selection method only improves the *PD*, *G-measure*, and *Balance* values for a small subset of datasets. The average *PD*, *G-measure*, and *Balance* values across the three datasets decrease. The primary reason might be that retaining more features helps NegCPARBP preserve greater data utility after privacy protection. Therefore, we recommend not using feature selection and instead applying privacy protection to all features.

B. Implications

The findings of our study offer several key implications.

(1) *Enhancing Privacy Protection in Cross-Project ARB Data Sharing:* In an era where privacy concerns are increasingly prominent, especially in cross-project ARB data sharing, NegCPARBP addresses these issues by generating privacy-preserving data using an *NDB* generation algorithm. This method represents an advancement in protecting sensitive information without sacrificing data utility. Additionally, we suggest retaining all software aging-related features in the training set rather than performing feature selection, to maximize data utility.

(2) *Practical Utility for Industry and Academia:* For both researchers and practitioners, the proposed method provides a practical solution for maintaining high levels of data utility while enhancing privacy. This makes NegCPARBP an appealing choice for organizations reluctant to share cross-project data due to privacy concerns, potentially fostering greater collaboration and data exchange in the software industry.

(3) *Foundation for Future Work:* This study represents the first application of the technique "negative representation of information" to the area of ARB prediction for privacy protection. The experiments demonstrate both the privacy-preserving capabilities and the data utility of the NegCPARBP method. Future research can build on this approach, exploring enhanced *NDB* generation techniques to further improve privacy protection and data utility in ARB prediction scenarios.

C. Threats to Validity

(1) This study focuses on three well-known datasets: Linux, MySQL, and NetBSD, which are commonly employed in previous research on CPARBP [1], [6]. However, we still cannot ascertain the generalizability of our method to other datasets. Future work should encompass testing our method on a broader range of datasets.

(2) Classification is a pivotal research domain within machine learning. In this paper, we employ three widely used classifiers in the field of bug prediction: NB, SVM, and RF. These classifiers serve as foundational models in the CPARBP field, each belonging to distinct categories: NB as a probabilistic model, SVM as a margin-based model, and RF as a decision tree model. However, the classifiers we utilized only represent a subset of all possible classifiers; other unused classifiers may yield different outcomes.

(3) We use *PD*, *PF*, *G-measure*, and *Balance* to evaluate the methods used in this study. *PD* and *PF* provide a comprehensive assessment of classifier performance on imbalanced datasets. *PD* measures the classifier's ability to detect positives, while *PF* assesses the risk of false positives. *G-measure*, representing the harmonic mean of the probabilities of correctly predicting positive and negative classes, offers a comprehensive evaluation of classifier performance, especially beneficial for datasets with class imbalance. *Balance* is a performance metric that offers an alternative perspective in assessing the classification ability of imbalanced datasets by jointly considering *PD* and *PF*. Other metrics utilized in software engineering are not reported in this

work. We will continue to explore the performance of additional metrics in future work.

(4) To generate *NDBs* by the NegCPARBP method, we propose an improved algorithm *IK*-hidden based on *QK*-hidden, which involves some random factors. The randomness in the algorithm may lead to variations in the experimental results, affecting the reproducibility and generalizability of our findings. Therefore, we conduct ten repetitions and utilize the mean result of these ten runs as the experimental outcome. Additionally, there are also random factors in the baseline methods. To compare the effectiveness of different methods, we employ the Wilcoxon sign-ranked test and Cliff's δ on the results obtained from ten runs of each method. These tests allow us to determine whether a method exhibits statistically significant superiority over others, providing valuable insights for method selection and evaluation.

VII. Conclusion

Software aging and the associated risks, particularly ARBs, emphasize the need for effective detection and prediction methods. CPARBP has been verified as a promising technique in addressing source-project data limitations but it also raises privacy concerns when utilizing source-project data. Our proposed method NegCPARBP introduces a novel approach inspired by the negative selection mechanism to protect privacy during ARB prediction. It preprocesses data, generates a Negative DataBase (*NDB*) containing significantly different data from the original feature vector, and extracts privacy-protected data for sharing and ARB prediction. The experimental results demonstrate NegCPARBP's superiority over existing methods in achieving high privacy protection rates while maintaining data utility for ARB prediction. As privacy concerns continue to be paramount, NegCPARBP offers a valuable contribution to safeguarding data owners' privacy in the context of CPARBP.

References

[1] X. Wan, Z. Zheng, F. Qin, Y. Qiao, and K. S. Trivedi, "Supervised representation learning approach for cross-project aging-related bug prediction," in *Proc. IEEE 30th Int. Symp. Softw. Rel. Eng.*, 2019, pp. 163–172.

[2] R. Romagnoli, B. H. Krogh, D. de Niz, A. D. Hristozov, and B. Sinopoli, "Runtime system support for CPS software rejuvenation," *IEEE Trans. Emerg. Topics Comput.*, vol. 11, no. 3, pp. 594–604, Third Quarter 2023.

[3] F. Qin, Z. Zheng, X. Wan, Z. Liu, and Z. Shi, "Predicting aging-related bugs using network analysis on aging-related dependency networks," *IEEE Trans. Emerg. Topics Comput.*, vol. 11, no. 3, pp. 566–579, Third Quarter 2023.

[4] E. Marshall, "Fatal error: How Patriot overlooked a Scud," *Science*, vol. 255, no. 5050, pp. 1347–1347, 1992.

[5] M. Nogueira et al., "A large scale characterization of device uptimes," *IEEE Trans. Emerg. Topics Comput.*, vol. 11, no. 3, pp. 553–565, Third Quarter 2023.

[6] F. Qin, Z. Zheng, Y. Qiao, and K. S. Trivedi, "Studying aging-related bug prediction using cross-project models," *IEEE Trans. Rel.*, vol. 68, no. 3, pp. 1134–1153, Sep. 2019.

[7] M. Grottke, A. P. Nikora, and K. S. Trivedi, "An empirical investigation of fault types in space mission system software," in *Proc. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, 2010, pp. 447–456.

[8] C.-Y. Huang and T.-Y. Kuo, "Queueing-theory-based models for software reliability analysis and management," *IEEE Trans. Emerg. Topics Comput.*, vol. 5, no. 4, pp. 540–550, Fourth Quarter 2017.

[9] D. Cotroneo, R. Natella, and R. Pietrantuono, "Is software aging related to software metrics," in *Proc. IEEE 2nd Int. Workshop Softw. Aging Rejuvenation*, 2010, pp. 1–6.

[10] D. Cotroneo, R. Natella, and R. Pietrantuono, "Predicting aging-related bugs using software complexity metrics," *Perform. Eval.*, vol. 70, no. 3, pp. 163–178, 2013.

[11] D. Cotroneo, M. Grottke, R. Natella, R. Pietrantuono, and K. S. Trivedi, "Fault triggers in open-source software: An experience report," in *Proc. IEEE 24th Int. Symp. Softw. Rel. Eng.*, 2013, pp. 178–187.

[12] F. Qin, Z. Zheng, C. Bai, Y. Qiao, Z. Zhang, and C. Chen, "Cross-project aging related bug prediction," in *Proc. IEEE Int. Conf. Softw. Qual. Rel. Secur.*, 2015, pp. 43–48.

[13] B. Xu, D. Zhao, K. Jia, J. Zhou, J. Tian, and J. Xiang, "Cross-project aging-related bug prediction based on joint distribution adaptation and improved subclass discriminant analysis," in *Proc. IEEE 31st Int. Symp. Softw. Rel. Eng.*, 2020, pp. 325–334.

[14] H. Kaur and A. Kaur, "An empirical study of aging related bug prediction using cross project in cloud oriented software," *Informatica*, vol. 46, no. 8, pp. 105–120, 2022.

[15] F. Peters and T. Menzies, "Privacy and utility for defect prediction: Experiments with morph," in *Proc. IEEE 34th Int. Conf. Softw. Eng.*, 2012, pp. 189–199.

[16] F. Peters, T. Menzies, L. Gong, and H. Zhang, "Balancing privacy and utility in cross-company defect prediction," *IEEE Trans. Softw. Eng.*, vol. 39, no. 8, pp. 1054–1068, Aug. 2013.

[17] F. Peters, T. Menzies, and L. Layman, "LACE2: Better privacy-preserving data sharing for cross project defect prediction," in *Proc. IEEE/ACM 37th IEEE Int. Conf. Softw. Eng.*, 2015, pp. 801–811.

[18] Z. Li, X.-Y. Jing, X. Zhu, H. Zhang, B. Xu, and S. Ying, "On the multiple sources and privacy preservation issues for heterogeneous defect prediction," *IEEE Trans. Softw. Eng.*, vol. 45, no. 4, pp. 391–411, Apr. 2019.

[19] M. Shepperd and S. MacDonell, "Evaluating prediction systems in software project estimation," *Inf. Softw. Technol.*, vol. 54, no. 8, pp. 820–827, 2012.

[20] O. Jalali, T. Menzies, and M. Feather, "Optimizing requirements decisions with keys," in *Proc. 4th Int. Workshop Predictor Models Softw. Eng.*, 2008, pp. 79–86.

[21] P. E. Hart, D. G. Stork, and R. O. Duda, *Pattern Classification*. Hoboken, NJ, USA: Wiley, 2000.

[22] J. Wright, A. Y. Yang, A. Ganesh, S. S. Sastry, and Y. Ma, "Robust face recognition via sparse representation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 31, no. 2, pp. 210–227, Feb. 2009.

[23] F. Esponda, S. Forrest, and P. Helman, "Enhancing privacy through negative representations of data," DTIC Document, Tech. Rep. A667894, 2004.

[24] F. Esponda, E. S. Ackley, P. Helman, H. Jia, and S. Forrest, "Protecting data privacy through hard-to-reverse negative databases," *Int. J. Inf. Secur.*, vol. 6, no. 6, pp. 403–415, 2007.

[25] D. Zhao et al., "K-means clustering and kNN classification based on negative databases," *Appl. Soft Comput.*, vol. 110, 2021, Art. no. 107732.

[26] H. Zhu, X. Wang, J. Zhao, S. Geng, and L. Wang, "Multi-client authenticated intersection protocol from a negative database server," *J. Internet Technol.*, vol. 21, no. 6, pp. 1659–1669, 2020.

[27] D. Zhao, P. Zhang, J. Xiang, and J. Tian, "NegDL: Privacy-preserving deep learning based on negative database," in *Proc. IEEE 4th Int. Conf. Data Intell. Secur.*, 2022, pp. 118–126.

[28] H. Jia, C. Moore, and D. Strain, "Generating hard satisfiable formulas by hiding solutions deceptively," *J. Artif. Intell. Res.*, vol. 28, no. 1, pp. 107–118, 2007.

[29] R. Liu, W. Luo, and L. Yue, "The P-hidden algorithm: Hiding single databases more deeply," *Immune Comput.*, vol. 2, no. 1, pp. 43–55, 2014.

[30] D. Zhao, W. Luo, R. Liu, and L. Yue, "A fine-grained algorithm for generating hard-to-reverse negative databases," in *Proc. Int. Workshop Artif. Immune Syst.*, 2015, pp. 1–8.

[31] Y. Yang and J. O. Pedersen, "A comparative study on feature selection in text categorization," in *Proc. 14th Int. Conf. Mach. Learn.*, D. H. Fisher, Ed., Nashville, TN, USA, Morgan Kaufmann, Jul. 8–12, 1997, pp. 412–420.

[32] SciTools, "Understand." Accessed: Mar. 2, 2025. [Online]. Available: https://scitools.com/

[33] S. Kotsiantis and D. Kanellopoulos, "Discretization techniques: A recent survey," *GESTS Int. Trans. Comput. Sci. Eng.*, vol. 32, no. 1, pp. 47–58, 2006.

[34] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," *IEEE Trans. Softw. Eng.*, vol. 33, no. 1, pp. 2–13, Jan. 2007.

[35] S. Wang and X. Yao, "Using class imbalance learning for software defect prediction," *IEEE Trans. Rel.*, vol. 62, no. 2, pp. 434–443, Jun. 2013.

[36] Y. Jiang, B. Cukic, and Y. Ma, "Techniques for evaluating fault prediction models," *Empirical Softw. Eng.*, vol. 13, no. 5, pp. 561–595, 2008.

[37] D. Rey and M. Neuhäuser, *Wilcoxon-Signed-Rank Test*. Berlin, Germany: Springer, 2011, pp. 1658–1659.

[38] G. Macbeth, E. Razumiejczyk, and R. D. Ledesma, "Cliff's Delta Calculator: A non-parametric effect size program for two groups of observations," *Universitas Psychologica*, vol. 10, no. 2, pp. 545–555, 2011.

[39] Y. Li et al., "Training data debugging for the fairness of machine learning software," in *Proc. 44th Int. Conf. Softw. Eng.*, 2022, pp. 2215–2227.

[40] D. D. Lewis, "Naive (Bayes) at forty: The independence assumption in information retrieval," in *Proc. Eur. Conf. Mach. Learn.*, Springer, 1998, pp. 4–15.

[41] J. C. Platt, "Fast training of support vector machines using sequential minimal optimization," in *Advances in Kernel Methods: Support Vector Learning*. Cambridge, MA, USA: MIT Press, 1998, pp. 41–65, doi: 10.7551/mitpress/1130.003.0016.

[42] M. Belgiu and L. Drăguţ, "Random forest in remote sensing: A review of applications and future directions," *ISPRS J. Photogrammetry Remote Sens.*, vol. 114, pp. 24–31, 2016.

[43] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik, "Gene selection for cancer classification using support vector machines," *Mach. Learn.*, vol. 46, pp. 389–422, 2002.

[44] C. Zhang et al., "The impact of feature selection techniques on aging-related bug prediction models: An empirical investigation," May 2023. [Online]. Available: http://dx.doi.org/10.21203/rs.3.rs-2901295/v1

Dongdong Zhao received the PhD degree from the University of Science and Technology of China (USTC) in 2016. He is currently an associate professor with the School of Computer and Artificial Intelligence of Wuhan University of Technology. His research interests include dependable computing, information security, privacy protection and biometrics.

Zhihui Liu received the bachelor's degree from the School of Computer Science and Artificial Intelligence, Wuhan University of Technology, in 2022. He is currently working toward the MS degree with School of Computer Science and Artificial Intelligence, Wuhan University of Technology. His research interests include privacy protection and machine learning.

Fengji Zhang received the MS degree from School of Computer Science, Wuhan University in 2023. He is currently working toward the PhD degree with the Department of Computer Science, City University of Hong Kong. His research interests include large code models.

Lei Liu received the bachelor's degree from the School of Computer Science and Artificial Intelligence, Wuhan University of Technology, in 2023. He is currently working toward the MS degree with School of Electronic Science and Engineering, Xi'an Jiaotong University. His research interests include software engineering.

Jacky Wai Keung (Senior Member, IEEE) received the BSc(Hons) degree in computer science from the University of Sydney and the PhD degree in software engineering from the University of New South Wales, Australia. He is currently an associate professor with the Department of Computer Science, City University of Hong Kong. His research interests include software and systems engineering, data science, AI, FinTech, machine learning, blockchain systems for FinTech applications, deep learning, LLM applications, defect prediction, and empirical modeling and evaluation of complex systems.

Xiao Yu received the PhD degree from the School of Computer Science, Wuhan University in 2020, and the Department of Computer Science, City University of Hong Kong in 2021. Currently, he is working as a research fellow with the State Key Laboratory of Blockchain and Data Security, Zhejiang University. His research interests include large language models and software engineering.