Noname manuscript No.
(will be inserted by the editor)

Towards Graph-Anonymization of Software Analytics Data

Empirical Study on JIT Defect Prediction

Akshat Malik · Bram Adams · Ahmed E. Hassan

Received: date / Accepted: date

Abstract As the usage of software analytics for understanding different organizational practices becomes prevalent, it is important that data for these practices is shared across different organizations to build a common understanding of software systems and processes. Yet, organizations are hesitant to share this data and trained models with one another due to concerns around privacy, e.g., because of the risk of reverse engineering the training data of the models. To facilitate data sharing, tabular anonymization techniques like MORPH, LACE and LACE2 have been proposed to provide privacy to defect prediction data. However, said techniques treat data points as individual elements, and lose the context between different features when performing anonymization. We study the effect of four anonymization techniques, i.e., Random Add/Delete, Random Switch, k-DA and Generalization, on the privacy score and performance in six large, long-lived projects. To measure privacy, we use the IPR metric, which is a measure of the inability of an attacker to extract information about sensitive attributes from the anonymized data. We find that all four graph anonymization techniques are able to provide privacy scores higher than 65% in all the datasets, while Random Add/ Delete and Random Switch are even able to achieve privacy scores of 80% and greater in all datasets. For techniques achieving privacy scores of 65%, the AUC and Recall decreased by a median of 1.45% and 5.35%, respectively. For techniques with privacy scores 80% or greater, the AUC and Recall of privatized models decreased by a median of 6.44% and 20.29%, respectively. The state-of-the-art tabular techniques like MORPH, LACE and LACE2 provide high privacy scores (89%-99%); how-

Akshat Malik
MCIS/SAIL, Queen's University, Kingston, ON, Canada
E-mail: akshat.ndun@gmail.com

Bram Adams
MCIS, Queen's University, Kingston, ON, Canada

Ahmed E. Hassan
SAIL, Queen's University, Kingston, ON, Canada

ever, they have a higher impact on performance with a median decrease of 21.15% in AUC and 80.34% in Recall. Furthermore, since privacy scores 65% or greater are adequate for sharing, the graph anonymization techniques are able to provide more configurable results where one can make trade-offs between privacy and performance. When compared to unsupervised techniques like a JIT variant of ManualDown, the GA techniques perform comparable or significantly better for AUC, G-Mean and FPR metrics. Our work shows that graph anonymization can be an effective way of providing privacy while preserving model performance.

Keywords Graph Anonymization · Privacy · Software Defect Prediction · Knowledge Graphs

1 Introduction

Software analytics has become an essential part of any organization's software development process, where organizations use machine learning models to help teams to (amongst others) analyze the probability of bugs in code [9], the time required to fix them [52] and the causes of build failures [53]. Such models are trained on defect metrics, development effort metrics and build artifacts using a variety of machine learning (ML) algorithms.

However, in an increasing number of cases, organizations are unwilling to share their data and models with each other, even within their organizations and across different teams. This unwillingness is primarily due to privacy concerns organizations have about the possibility of extracting underlying data from a trained machine-learning model, as demonstrated in recent academic studies [56, 43]. These works demonstrate how, from a trained model, it is possible to reconstruct the properties of the data used to train the model. To achieve this, attackers train a "shadow model" to imitate the behaviour of a trained model on a dataset. Using this, they can determine whether a data point belongs to the training set. Leakage of such knowledge could be highly detrimental to an organization's security and reputation. For example, revealing that a company fixes a hundred bugs a day while its competitors only fix ten could cause distress among its user base.

One of the most recently studied ways to deal with this threat is to anonymize ML models. There are multiple families of techniques to anonymize ML models, ranging from perturbing the models' data input, where random noise is added into the training data so that the output is private, to perturbing the actual model training process (e.g., the intermediate values of an iterative ML algorithm using federated learning), or adding noise to the model outputs [18, 41]. Each of these families comprises a variety of specific techniques. This paper focuses on the first family, i.e., anonymizing the training data, since anonymized data does not limit the ML algorithms that can be used on it, enabling data to be used with any training algorithm, making our approach flexible.

In terms of preserving privacy in data, several general-purpose anonymization techniques such as k-anonymity, l-diversity, t-closeness, personalized privacy and differential privacy have been proposed [29]. To measure how effectively an anonymization technique provides resistance to the data against attacks by nefarious users, privacy scores can be used to estimate how different the anonymized data is from the non-anonymized data or how effectively information is hidden in the anonymized data. As pointed out by a recent survey [51], over time, several privacy measures have been proposed. For instance, a popular metric for privacy is the Increased Privacy Ratio (IPR) [35]. Based on some background information about the data, it measures what percentage of data has changed after anonymization, measuring how much information is being leaked about the sensitive attributes of the data. An IPR score of, for instance, 80% means that the anonymized dataset in 80% of the cases (or attacks) will result in different, i.e., safer, values for the sensitive attributes than the unanonymized data.

In the world of software analytics, Peters et al. propose the MORPH, LACE and LACE2 [35, 37, 36] algorithms, while Li et al. propose SRDO [26] for anonymization. They empirically validated that it is possible to provide privacy to defect prediction data without affecting its performance. In our replication using JIT defect prediction metrics, we indeed found that the MORPH, LACE and LACE2 algorithms provided high privacy, with scores in the range of 89% to 99%, which exceeds the threshold for adequate data sharing of 65% [36]. However, this high privacy came at the cost of decreased performance of the model, where, across all datasets, the AUC decreased by a median of 21.2%, 17.15% and 21.45%, and Recall by a median of 64.79%, 58.79% and 96.43%, respectively.

In essence, the existing tabular anonymization techniques just treat the software analytics data as a list of metrics, ignoring any logical connection between them. For example, in the case of JIT defect prediction, the number of lines modified by a commit should depend on the number of files modified by that commit. If the number of files modified by the commit changes, it will have a proportional change in the number of lines changed by the commit. The tabular anonymization techniques mostly ignore such dependencies between the various data features, thereby losing context during anonymization.

A promising direction to address this lack of context is the use of graph anonymization techniques, which perform anonymization on a graph representation of the data. Assuming that the latter graph representation can be used to calculate the metrics of interest, graph anonymization promises to not only retain the structural knowledge of the graph better, but also to yield proportional changes to all metrics derived from the graph, preserving the graph's overall utility [23].

While tabular data anonymization for software analytics has been evaluated in the context of traditional defect prediction, graph anonymization techniques have not been considered thus far. Hence, our empirical study evaluates the use of graph anonymization techniques on Just-In-Time (JIT) Defect Prediction data. This popular software analytics task focuses on predicting defective commits using only commit-level information like the number of lines added, number of files changed and author experience. JIT defect prediction data is versatile and can be used for different software analytics tasks, like within-project and cross-project settings. We aim to evaluate to what extent graph anonymization can provide privacy to the JIT data while preserving the predictive power of the models trained on it. This would help us establish whether graph anonymization techniques retain the predictive capability of JIT data to be further used in different analytics tasks.

To evaluate the effect graph anonymization techniques have on the privacy and performance of the data, we adapt four anonymization techniques (Random Add/Delete, Random Switch, k-DA anonymity and Generalization) to be applied to the knowledge graph of the software analytics data collected for six large, long-lived projects in a within-project JIT defect prediction setting: OpenStack, Qt, Apache Flink, Apache Ignite, Apache Groovy and Apache Cassandra. We answer the following research questions:

1. **RQ1: How do different graph anonymization techniques affect the privacy of the JIT defect prediction data?**
   This question evaluates how effectively different graph anonymization techniques provide privacy to the JIT metrics generated from the anonymized knowledge graphs. Using the privacy metric Increased Privacy Ratio (IPR), it was found that all the techniques are able to provide privacy scores greater than 65% (adequate for sharing data) with minimal effort. Furthermore, Random Add/Delete and Random Switch are able to provide privacy scores greater than 80% to all the datasets, while k-DA and Generalization were able to do so for 3 and 4 datasets, respectively. Random Add/Delete is able to provide privacy scores close to 90% to all the datasets. All techniques change different amounts of links in the graph to reach the same privacy levels, with Generalization doing so with the least changes for 7 out of the 12 combinations of 2 privacy levels (Privacy Level I and II) and 6 datasets, followed by Random Add/Delete (3/12) and k-DA (2/12). Hence, graph anonymization techniques are effective in providing privacy to the JIT metrics.
2. **RQ2: How do different graph anonymization techniques affect the performance of JIT defect prediction models?**
   As the graph anonymization techniques change the graph representing software analytics data to provide privacy, it is likely that these changes cause the performance of the metrics generated to degrade. With this second RQ, we want to quantify these techniques' impact on the model performance. Our case study shows that privacy does not come at the cost of performance. All techniques provide privacy scores greater than 65% with a median decrease of 1.45% and 5.35% in AUC and Recall, across all datasets. For 80% and greater privacy scores, the median loss in AUC is 6.44% and in Recall is 20.29%, across all datasets. Generalization performs significantly better than the other techniques in preserving the performance of the model. Graph anonymization techniques are able to preserve the

model's performance while providing enough privacy to share the defect prediction data.

3. **RQ3: How do the graph anonymization techniques compare to the state-of-the-art anonymization techniques?**
   Since using graph anonymization techniques in software analytics is a novel approach, we want to compare how our attempt to graph-anonymize JIT defect prediction metrics compares to the state-of-the-art techniques that aim to provide privacy for JIT metrics. Across all projects, we find that MORPH, LACE and LACE2 were able to provide privacy scores greater than 89%; however, their AUC and Recall fell by a median of 21.15% and 80.35%, respectively, compared to the median loss in AUC of 6.44% and Recall of 20.29% for graph anonymization techniques. Furthermore, graph anonymization techniques provide greater control, where scores of 65% and greater (adequate for sharing) can be achieved with a 1.45% decrease in AUC, scores of 80% and greater (higher privacy threshold) can be achieved with 6.4% decrease in AUC and scores of 88% (with Random Add/Delete) can be achieved with AUC reduced by 7.75%. Lastly, when compared to Manual Down (MD50), the GA techniques perform significantly better or similar in the majority of the cases for the AUC, G-Mean and FPR metrics.

Our study shows that it is possible to use graph anonymization techniques to provide privacy while maintaining performance. These techniques offer fine control over the trade-offs between privacy and performance. When compared to state-of-the-art techniques, MORPH, LACE and LACE2 provide more privacy than required for adequate sharing [36], with graph anonymization techniques having a lesser impact on the performance of the metrics.

Our paper makes the following contributions:

1. Proposing a novel way to represent the software analytics data, in this case, JIT defect prediction data, as a knowledge graph of interconnected entities.
2. Proposing a methodology to generate JIT defect prediction metrics from the knowledge graph.
3. Empirically evaluating 56 configurations of 4 graph anonymization techniques to provide privacy to defect prediction data and measuring how they can be applied to provide privacy to the JIT defect prediction metrics.
4. Empirically comparing to tabular anonymization techniques (MORPH, LACE and LACE2) and unsupervised learning methods (Manual Down) to evaluate the effectiveness of graph anonymization techniques.

2 Background and Related Work

In this section, we give an overview of how different graph anonymization techniques work and discuss existing works for anonymizing defect prediction data. We also discuss JIT defect prediction and how literature has shown that ML models can leak information.

2.1 Reverse Engineering Information from ML Models

Fredrikson et al. [31] demonstrate a way of reconstructing the data on which a model is trained based on the confidence score that is assigned to every prediction. Using limited information about the training data and access to the ML model, the authors were able to extract sensitive information about the data used in training decisions. The authors demonstrate the attacks on image and feature classification problems.

A similar possibility of attack was shown by Shokri et al. [44], who demonstrate how it is possible to extract information about the training data from a black box machine learning model. To do this, the authors use a machine learning model that aims to distinguish differences in model behaviour depending on whether the input was encountered by a model in its training or not. The paper treats the ML model as a black box, only interacting with its API. It was able to perform a membership inference attack on classification models provided by machine learning service providers like Google and Amazon, with an accuracy of 94% and 74%. This highlights the significant privacy implications that ML models can have.

These papers form the basis for our motivation to anonymize the data ML models are trained on to prevent attackers from learning about the sensitive attributes of the underlying software analytics data.

2.2 Existing Anonymisation Methods

Our work extends the work of Peters et al. with MORPH [35], LACE [37] and LACE2 [36] and the works by Li et al. on SRDO [26]. MORPH is an instance mutator algorithm that modifies all data points by a small random amount. The authors further extend this work in LACE, adding an instance pruner algorithm called CLIFF that removes instances based on how frequently an attribute occurs in a class (in defect prediction data, a class would be "defective" or "not defective"). To further enable sharing of data among different projects, they propose LACE2, a multi-party privacy policy. It allows different data owners to work together to create a privatized cache where each owner can add data based on what has already been added. Finally, SRDO is a privacy-preserving algorithm that uses a sparse representation based on double obfuscation to improve the techniques underlying MORPH's data mutation.

The fundamental difference between the above works and ours is that the former view data mostly as rows of attributes. However, we represent the data as a graph of interconnected entities, retaining the structural dependencies between the entities. When performing graph anonymization, the metrics derived from the graph should gain privacy without sacrificing their performance, because we believe that the graph anonymization techniques preserve context and information in the graph when altering it. Furthermore, we are evaluating the existing and proven graph anonymization techniques' ability to provide privacy to the graph of software analytics data. To our knowledge, this is the first instance of using graph anonymization on software analytics data to provide privacy.

The prior work [35, 37, 36, 26] also conduct their study on file-based bug prediction, i.e., using CK metrics on the PROMISE dataset. The CK metrics are used to evaluate various aspects of the complexity and quality of object-oriented software systems [12], whereas we use JIT change metrics collected from the GitHub repositories of the projects. Lastly, these state-of-the-art techniques use the data elements' class labels, i.e., buggy or not, when performing anonymization operations to provide privacy, while the graph anonymization techniques do not use class labels, i.e., they are unsupervised.

Another approach is ManualDown [16], an unsupervised learning model that considers the top $x\%$ largest files that have been changed in a current time period as buggy files. An interpretation of ManualDown in the context of JIT (inspired by other unsupervised JIT approaches [21, 55]) would consider the top $x\%$ commits with the most lines added as buggy. Basically, such a JIT variant of ManualDown would sort all commits in a studied time period by the "number of lines added" feature, then mark the top $x\%$ instances as buggy commits and the remaining ones as not buggy. ManualDown models do not require actual commit data other than the "number of lines added".

Yamamoto et al. [54] propose using federated learning to build a JIT defect prediction model. The study shows that building models using a federated learning approach is possible, not requiring any data to be shared among projects. However, recent works have shown that federated learning models are still open to ML model leaks [39, 34, 10] according to which researchers have been able to perform a membership inference attack. Basically, exploiting the cyclic nature of model learning and the exchange of weights between the different learners and central aggregation nodes. This would expose the un-anonymized training data to malicious users. Essentially, the repeated sharing of model weights via a central node that needs to be trusted by all parties is also the weak point of federated learning approaches from a privacy point of view. It does make attacks harder, but it does not remove the requirement for privatizing the models' input data. We believe that future work can use graph anonymized data on each learner node in a federated learning setting to provide even more privacy to the trained model.

2.3 Overview of Graph Anonymisation Techniques

A graph [1] is a set of nodes (vertices) connected via edges (links). Graphs can be directed, where the relationship goes from one edge to another, or undirected, where the relationship is bidirectional. They provide a unique way to represent interconnected information between different entities. Information in a graph can be extracted by querying the nodes and their links. For example, a graph could link commit information (git information) to the pull request (code review) and, in turn, the report data (bug information) that led to the commit, providing the context for that commit.

Graph anonymization can be performed in two ways, i.e. online or offline. Online techniques anonymize data in real-time, where data is anonymized while querying it. Offline techniques anonymize all of the data together before releasing it for further use. This causes the offline techniques to be slower and computationally more intensive than the online techniques. However, offline techniques are better able to provide privacy and preserve utility as they consider the entire graph, allowing them to consider the overall properties of the underlying analytics data. For this reason, we focus on the offline method, anonymizing all the data at once.

Graph anonymization techniques are considered to be effective in providing privacy to the different entities of a graph while preserving the knowledge and context embedded in the graph [32, 17]. Many graph anonymization techniques have been researched, and they can be divided into the following broad classification [23]:

1. **Naive ID Removal:** This technique removes the ID of all the graph entities. This is easy to apply; however, it does not provide much security to the graph entities as they can be easily re-identified [33, 45]. We do not consider these techniques.
2. **Edge Editing (EE) based techniques:** These techniques operate on a graph by changing its edges. The two ways these techniques can be implemented are Add/Delete and Switch. With Add/Delete, x random links are deleted from the graph and randomly added to other nodes. With Switch, the links of two randomly chosen nodes are swapped [57].
3. **k-Anonymity-based techniques:** k-Anonymity [14] techniques are most commonly used for anonymizing graphs. In this technique, a number k is selected, then the graph is modified until there are at least $k - 1$ similar entities in the graph. k-anonymity techniques have various variants, which include: k-Neighborhood Anonymity [59], k-Degree Anonymity [27], k-Automorphic [60] and k-Isomorphic technique [11].
4. **Aggregation/Class/Cluster-based techniques:** These techniques group entities with similar structures into clusters, then modify the nodes in the clusters to achieve the same structure to provide them anonymity [49].
5. **Differential Privacy (DP) based techniques:** Differential privacy techniques guarantee the amount of privacy provided to a graph. It does so by capturing the graph structure in terms of certain statistics, then accurately regenerating the graph by adding a certain amount of noise in the statistics [42].
6. **Random Walk (RW):** This technique changes the edges between two nodes in the graph, which are a random i distance apart [30].

3 Graph Anonymization for JIT Defect Prediction

In this section, we provide a detailed overview of the different graph anonymization techniques introduced previously in the specific context of JIT defect prediction.

Similar to other software analytics data, JIT defect prediction metrics are linked to one another, wherein a change in the number of files changed by a commit will have a proportional change in the number of lines modified by the commit and the commit author's experience. Hence, instead of anonymizing individual metric values, we propose representing the underlying software analytics data as a graph and anonymizing the graph before calculating the JIT metrics. A change in any entity of the graph will have a proportional change in all the related entities. Therefore, we believe using graph anonymization techniques would preserve the structural knowledge between different graph entities while providing sufficient privacy.

To derive graph-anonymized change metrics for JIT defect prediction, we perform the following steps:

1. Creation of a knowledge graph of the involved repositories.
2. Anonymization of the graph using different algorithms.
3. Calculation of the change metrics from the anonymized graph.

3.1 Step 1: Knowledge Graph Creation

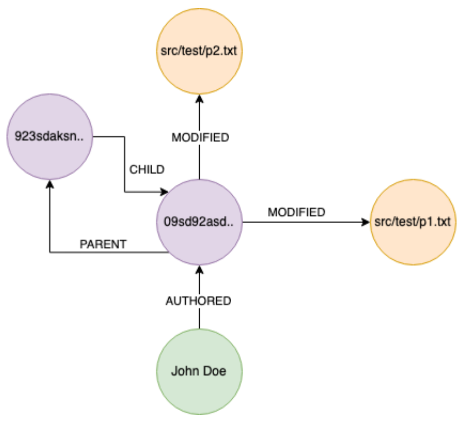


Fig. 1: Example of a Knowledge Graph. The purple nodes are commits, the beige nodes are files and the green node is a person.

A knowledge graph refers to a graph of data that represents information about different nodes representing real-world entities, and links that define how those entities are related to one another [20]. A knowledge graph can be

Entity	Node Properties	Links to other entities (Linked Entity - Link Name - Link Properties)
Commit	ID - The hash of the commit Author-Date - The date the commit was created Bugcount - The number of bugs introduced by the commit Fixcount - The number of bugs fixed by the commit	Files - MODIFIED - Lines Added, Lines Deleted Commit - PARENT Commit - CHILD
File	ID - The complete path of the file File Name - The name of the the file Directory - The directory in which the file is stored Subsystem - The top level directory of a file	
Person	ID - The unique identifier for a person	Commit - AUTHORED

Table 1: Entities and their properties and links

made for different domains like web search, social media, risk assessment and more. In the context of JIT defect prediction, we focus on a knowledge graph made from the information available in the git repository of the datasets to create a graph that we call "git-graph", as shown in Figure 1. The knowledge graph contains the minimum set of entities needed to calculate the change metrics needed for JIT defect prediction. The graph's nodes and links help us understand the relationship between various entities and allow us to derive the JIT defect metrics (see Step-3 in Section 3.3). The various entities and their links are listed below:

1. **Commit:** This represents a commit in the git repository. This entity is linked to the *File* nodes, representing the files changed by the commit. The commit is also linked to the *Person* node that authored the specific commit.
2. **File:** This represents a file present in the code base. Each file is unique to the graph database, i.e., there cannot be duplicate nodes representing the same file. Multiple commits in the graph can modify the same file. The changes a commit makes to a file are represented by the *Lines Added* and *Lines Deleted* properties of the link, representing the number of lines added and the number of lines deleted.
3. **Person:** A *Person* node represents a developer who worked on the repository. A person can be the one who authored a particular commit. In a graph, there is always one unique node for each person.

Table 1 lists all the node entities along with their properties and the links they have. The entities used to create a git-graph are sufficient to compute all the change metrics needed for our study, as highlighted in Table 2. It is possible to create more complex knowledge graphs that utilize bug reports, reviewer experience, review comments, etc. However, we limit our empirical study to the data needed to calculate the most commonly studied features for JIT models [25, 46, 24].

3.2 **Step 2: Graph-Anonymization Algorithms**

The graph anonymization techniques detailed below work on the *Commit* and/or *Person* nodes of a git-graph. We do not operate on *File* nodes, as they do not have any entity dependent on them, i.e., they are leaf nodes.

3.2.1 ***Selected Techniques***

Out of the six categories for graph anonymization techniques mentioned in Section 2.3, we have selected the following four techniques (out of three categories) for our study:

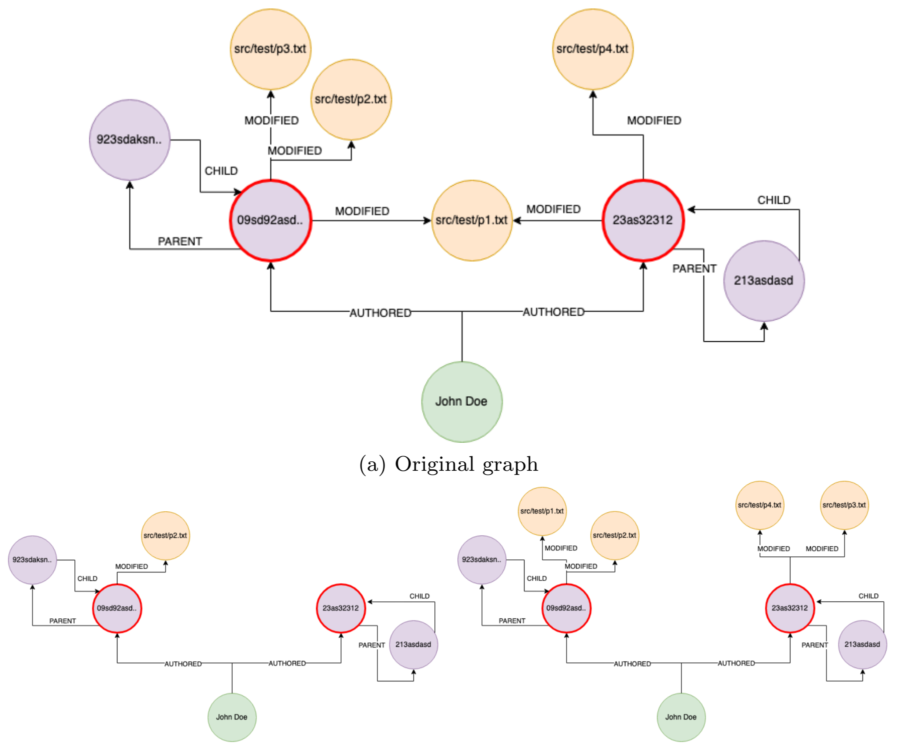
1. **Edge Editing (EE) based techniques [57] :** The paper proposes two variants of Edge Editing techniques, i.e., Add/Delete and Switch. We are selecting the *Random Add/Delete* (RAD) and the *Random Switch* (RS) variant, because the latter is similar to the Random Walk technique and would also cover the Random Walk category.
2. **k-Anonymity [27]:** For k-anonymity, we have chosen *k-Degree Anonymity* (k-DA) as it represents a more modern variant that protects against the re-identification of nodes from attackers that have prior knowledge about the graph.
3. **Aggregation/Class/Cluster-based techniques [49]:** We selected *Generalization* (Gen) from this category as it was the only listed variant.

As the "Naive ID Removal" category offers no privacy to the nodes, therefore, we do not implement it [33]. "Differential privacy" offers mathematical guarantees of privacy, but it has only been recently adopted to graphs. The algorithm proposed by Sala et al. [42] converts the graph to a set of structural statistics, which are converted back into a graph after adding a random amount of noise. However, when the graph is converted to a set of statistics, details like author date, directory, and subsystem are lost. These details are essential in computing the JIT defect metrics, leading us to not select techniques from this category. We leave the adaptation of Differential Privacy for JIT prediction for future work.

In the next section, we describe in detail how each of the four selected anonymization techniques - *Random Add/Delete, Random Switch, k-Degree Anonymity* and *Generalization* operate and how they are applied to the software analytics data's knowledge graph.

3.2.2 ***Random Add/Delete***

In Random Add/Delete, we anonymize x percent of the nodes of a particular type by modifying the node's links to another entity. The algorithm used is specified in Algorithm 1. We run the algorithm for each *Commit* node, where we modify the link it has to the modified *File* nodes (via the MODIFIED link), and for each *Person* node, for which we modify its *Commit* nodes (via the AUTHORED link).



(a) Original graph

(b) After Delete Operation

(c) After Add Operation

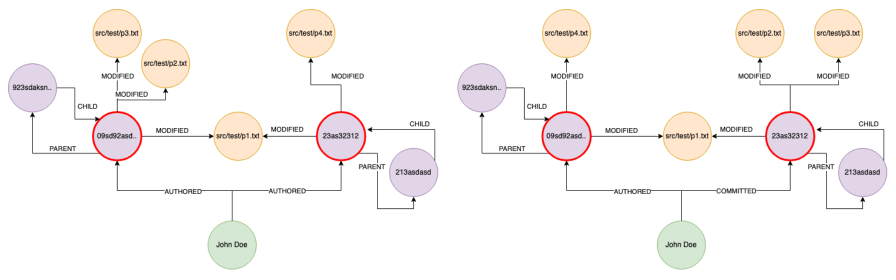
Fig. 2: Random Add/Delete operation. The nodes in red (Commit) are the ones on which the operation is being carried out.

From the x percent of nodes, the algorithm each time selects two nodes randomly to perform the anonymization. In Figure 2a, we see the unaltered graph to which the algorithm will be applied. Using the $randomlyDeleteAndGetNodes$ function, it first randomly deletes nodes of the $dependNodeType$ (e.g., File nodes of a Commit node) linked to the selected node and stores them in $nodeStore$. After randomly deleting the *File* nodes from the *Commit* nodes, we get Figure 2b.

As seen in Figure 2c, all the deleted nodes from the $nodeStore$ are then randomly linked back to the two nodes using the $randomlyAddNodes$ function, leaving no node in the $nodeStore$. This operation is repeated until x percent of the nodes in the graph are anonymized. We select two nodes at a time to ensure that the structure of the nodes after the change is not dramatically different.

Algorithm 1 Algorithm for Random Add/Delete Technique

```
1: procedure MAIN(percentToAnon)
2:     $randomAddDelete(CommitNodeType, FileNodeType, percentToAnon)$
3:     $randomAddDelete(PersonNodeType, CommitNodeType, percentToAnon)$
4: end procedure
5: procedure RANDOMADDDELETE($mainNodeType, dependNodeType, percentToAnon$)
6:     $modifyCount \leftarrow count(mainNodeType) \times percentToAnon$
7:     for $i \leftarrow 0$ to $modifyCount; i \leftarrow i + 2$ do
8:         // Select two random node of $mainNodeType$ from the graph
9:         $firstNode \leftarrow getRandomNode(mainNodeType)$
10:        $secondNode \leftarrow getRandomNode(mainNodeType)$
11:        // Randomly delete linked nodes and return them
12:        $nodeStore.add(randomlyDeleteAndGetNodes(firstNode, dependNodeType))$
13:        $nodeStore.add(randomlyDeleteAndGetNodes(secondNode, dependNodeType))$
14:        // Randomly select nodes from the store and link them to the mainNodeType
15:        $randomlyAddNodes(firstNode, nodeStore)$
16:        $randomlyAddNodes(secondNode, nodeStore)$
17:    end for
18: end procedure
```



(a) Original (b) After Switch Operation

Fig. 3: Random Switch operation. The nodes in red (Commit) are the ones on which the operation is being carried out.

3.2.3 ***Random Switch***

The Random Switch algorithm (Algorithm-2) works similarly to the Random Add/Delete algorithm. The algorithm selects two nodes randomly and switches all the links between them, unlike Random Add/Delete, which randomly changes some links. Figure 3 shows an example of the algorithm when applied to the *Commit* node and the *File* nodes they have modified. After randomly selecting two nodes, in Figure 3a, all the links have been switched around, as seen in Figure 3b. This algorithm is also applied to the *Person* nodes and the links that they have authored.

3.2.4 ***k-Degree Anonymity***

This technique requires that for every node, there are at least $k-1$ other nodes (excluding the current node) that have the same degree, yielding k nodes in

Algorithm 2 Algorithm for Random Switch technique

```
1: procedure MAIN(percentToAnon)
2:     $randomSwitch(CommitNodeType, FileNodeType, percentToAnon)$
3:     $randomSwitch(PersonNodeType, CommitNodeType, percentToAnon)$
4: end procedure
5: procedure RANDOMSWITCH($mainNodeType, dependNodeType, percentToAnon$)
6:     $modifyCount \leftarrow count(mainNodeType) \times percentToAnon$
7:     for $i \leftarrow 0$ to $modifyCount; i \leftarrow i + 2$ do
8:         // Randomly select node from graph
9:         $firstNode \leftarrow getRandomNode(mainNodeType)$
10:        $secondNode \leftarrow getRandomNode(mainNodeType)$
11:        // Delete all $dependentNode$ linked to the node
12:        $firstNodeStore \leftarrow deleteAndGetNodes(firstNode, dependNodeType)$
13:        $secondNodeStore \leftarrow deleteAndGetNodes(secondNode, dependNodeType)$
14:        // Switch the links, first to second node, and vice versa
15:        $addNodes(firstNode, secondNodeStore)$
16:        $addNodes(secondNode, firstNodeStore)$
17:    end for
18: end procedure
```

total with that degree. Degree refers to the number of nodes a particular node is linked to. In our context, when operating on a *Commit* node, the degree is the number of *File* nodes it is linked to and for a *Person* node, it is the number of *Commit* nodes it is linked to. For example, when operating on the *Commit* nodes, in Figure 3a, the *Commit* in red on the left has a degree of 3, while the right one in red has a degree of 2.

In this case, if the value of k is chosen to be 3, then the graph should have at least 2 more commits that modify 3 files. If this requirement is met, then the graph satisfies the condition of k-Degree anonymity, making data 3-degree anonymous. k-DA anonymity yields a $1/k$ probability of identifying which commit is modifying which file. It is important to note that it is not necessary that there should be k nodes for all the degrees from 1 to the max degree present in the graph, but rather that each degree that is present should have at least k nodes. For example, if a graph has degrees 2 and 4, k-DA ensures that degrees 2 and 4 have at least k nodes in them.

The algorithm for this technique is highlighted in Algorithm 3. It is applied on all *Commit* nodes and their linked *File* nodes, and on all *Person* nodes and their authored *Commit* nodes. The first step of the algorithm is to group the nodes based on the degree they have using the *GetDegree* function, which is stored in the *degreeStore* variable. The *degreeStore* variable has all the degrees that are present in the graph as keys (e.g., 2, 4, 10, etc.), and the value corresponding to a key is the list of nodes with that corresponding degree. The number of values that a corresponding key has helps us determine whether a key has k nodes in it or not.

Using the *GetDegreeBelowK*, we get all the degrees that do not have the required k number of nodes in them. Let's call a degree from this set that does not have the required k nodes, as m, with a difference of x from k. Then for each such degree m, we find another degree l that has more than $k + x$ nodes in it.

This ensures that the degree l should still have at least more than k nodes in it after the operation so that it still meets the condition of k-DA anonymity. From this selected degree l, we remove x nodes randomly and change their degree to match the degree m. This is carried out by the *changeDegree* function, which either randomly adds or deletes links to other nodes from the selected x nodes. By doing this operation, the nodes being modified will now belong to degree m, increasing the number of nodes in the degree m to k. Unlike Random Add/Delete and Random Switch, this technique works for all the nodes of a type in the graph until each node satisfies the k-degree constraint.

Algorithm 3 Algorithm for K-Degree Anonymous

```
1: procedure MAIN(k)
2:     $KDegreeAnon(k, CommitNodeType, FileNodeType)$
3:     $KDegreeAnon(k, PersonNodeType, CommitNodeType)$
4: end procedure
5:
6: procedure KDEGREEANON(k, nodeType, linkedNodeType)
7:     // degreeStore is a mapping of the degree to the list of nodes of that degree
8:     $degreeStore \leftarrow GetDegree(nodeType, linkedNodeType)$
9:     $degreeBelowK \leftarrow GetDegreeBelowK(k, degreeStore)$
10:    for $i \leftarrow degreeBelowK$ do
11:        $numNodesToAdd \leftarrow k - count(degreeStore[i])$
12:        $surplusDegree \leftarrow$
13:        $degreeWithCountGreaterThan(k + numNodesToAdd, degreeStore)$
14:        for $m \leftarrow 0$ to $numNodesToAdd$ do
15:            $randomNode \leftarrow getRandomNode(degreeStore[surplusDegree])$
16:            $changeDegree(randomNode, i)$
17:        end for
18:    end for
19: end procedure
20:
21: procedure GETDEGREE($nodeType, linkedNodeType$)
22:    $allNodes \leftarrow getAllNodes(nodeType)$
23:    for $node \leftarrow allNode$ do
24:        $linkedNodes \leftarrow getLinks(node, linkedNodeType)$
25:        $degreeStore[count(linkedNodes)].add(node)$
26:    end for
27:    return $degreeStore$
28: end procedure
29:
30: procedure GETDEGREEBELOWK($k, degreeStore$)
31:    for $i \leftarrow keys(degreeStore)$ do
32:        if $count(degreeStore[i]) < k$ then
33:            $lessThanKDegreeStore.add(i)$
34:        end if
35:    end for
36:    return $lessThanKDegreeStore$
37: end procedure
```

3.2.5 ***Generalization***

The generalization technique requires us to capture the structure of the graph and group nodes into clusters based on their structure. To capture the structure of the graph, we use the *Commit* node as a reference. The structure of a commit is represented by the number of *File* nodes it is linked to and the number of commits the author of the commit has created. Nodes with the same structure are grouped together into *cluster*. If a cluster has n nodes in it, it is called a full cluster; else, it is called an under-filled cluster.

The algorithm for capturing the structure of the graph and ensuring that there are enough elements in each cluster is specified in Algorithm 4. Using the *GetStructure* function, we capture the structure of each commit in the graph. We now attempt to create m clusters with at least n nodes in them, i.e., the graph should now have m new full clusters. The structure that is being targeted to have n nodes is found using the $GetStructWithLessThanKElem$ function that finds the clusters that do not have n nodes in them, let's call this set z. Using the $countMoreThan$ function, we find all the clusters that have more than n nodes in them. Then, for the nodes above n, we call the *modifyStructure*, which ensures that the node *nodeToModify* is changed to match the cluster structure z that is currently being targeted by randomly adding links or deleting links to nodes.

Algorithm 4 Algorithm for Generalisation Technique

```
 1: procedure MAIN(targetCount)
 2:     $structStore \leftarrow GetStructure()$
 3:     $underFilledStruct \leftarrow GetStructWithLessThanKElem(targetCount, structStore)$
 4:     for $underFillStruct\_i \leftarrow keys(underFilledStruct)$ do
 5:         $currCount \leftarrow count(underFilledStruct[underFillStruct\_i])$
 6:         $overFillStructKey \leftarrow countMoreThan(currCount + targetCount, structStore)$
 7:         for $j \leftarrow 0$ to $targetCount - currCount$ do
 8:             $nodeToModify \leftarrow structureStore[overFillStructKey]$
 9:             $modifyStructure(nodeToModify, underFillStruct_i)$
10:         end for
11:     end for
12: end procedure
13: procedure GETSTRUCTURE
14:     for $node \leftarrow getAllNode(CommitNodes)$ do
15:         $countFileNodes \leftarrow count(getLinkedNodes(node, FileNodes))$
16:         $author \leftarrow getAuthor(node)$
17:         $countCommitNodes \leftarrow count(getLinkedNodes(author, CommitNodes))$
18:         $structureKey \leftarrow countFileNodes + "," + countCommitNodes$
19:         $nodeStructure[structureKey].add(node)$
20:     end for
21: end procedure
```

Attribute	Description	Calculation based on anonymized graph
la	Number of lines added by a commit	For all the *File* nodes modified by a *Commit* node, we add the 'la_link' property
ld	Number of lines deleted by a commit	For all the *File* nodes modified by a *Commit*, we add the 'ld_link' property
ent	Entropy or the spread of changes across file	This is calculated by taking the log of the proportion of change to a file as compared to all the files
nf	Number of files	This is the number of unique files changed by a commit. This is calculated by counting the number of *File* nodes a *commit* node has modified
nd	Number of directories	For all the files found in the 'nf', we make a unique set of the directory of each file
ns	Number of subsystem	For all the files found in the 'nf', we make a unique set of subsystems for each file. The subsystem is the root directory for a file
nuc	Number of unique changes	For each modified file for the current *Commit* node, we find the number of commits that have previously modified those *File* nodes
ndev	Number of unique developers that changed the file	For all the commits found in the 'nuc' attribute, we find all the unique authors for all the unique modifying authors
age	Average time interval between current changes and the last changes made to the file	For all the commits found in the 'nuc' attribute, we then find the latest commit for each file. After that, we take the average of the time that commit happened and the current commit
aexp	Experience of the author	The number of nodes the author of the commit has modified before this commit
asexp	Subsystem experience of the author	For all the commits found in 'aexp', we find the unique subsystems (which are same as the subsystems modified by the current commit) those commits have modified
arexp	Relative experience of the author	This is calculated by taking the 'aexp' commits, and then dividing them by their age [28]

Table 2: Description and the method to derive the JIT defect prediction metrics (used by Kamei et al. [24]) calculated from the git-graph.

3.3 Step 3: Generation of JIT metrics

The knowledge graph for the git repository has all the information needed to generate the typical change metrics needed for JIT defect predictions [25, 46, 24]. In Table 2, we show all JIT defect prediction metrics used for model training and how we calculate them from the git-graph generated for each project. The calculation of the metrics for each commit is done as defined in the last column of Table 2. For example, to calculate the *la* attribute for a commit, we take the sum of the *la_link* link property for each *File* node attached to the commit. Similarly, we generate all other metrics for a commit using this approach and repeat it for all commits in the graph.

4 Empirical Study Design

In this section, we give details about the design of the empirical study aiming to address the RQs of the introduction, i.e., the datasets used for our case study, the configuration of the various graph anonymization techniques, and our approach to answering the three research questions.

4.1 **Subject System Selection**

As our study focuses on graph anonymizing JIT metrics and on evaluating the impact on their privacy and performance, we use datasets that were made available by other JIT defect prediction research. One of the most common JIT defect prediction datasets was shared by McIntosh et al. [28], and has been used in later research [38, 40, 19]. It features JIT data for OpenStack and Qt. The other dataset we have used is ApacheJIT [25], which makes data available for popular 14 Apache projects.

In Table 3, we have listed all the projects selected for our study. As graph anonymization changes the structure of the code, we need projects whose trained models have high AUC and have a sizeable number of commits. To calculate the AUC of the models, we split the data into a chronological 80/20 train/test split. We perform out-of-sample bootstrapping training 100 models by randomly selecting 500 elements from the training split and testing on 500 randomly selected testing elements. We then take the average of these scores to find the average AUC. Because of this, we place two conditions on the datasets: a) have an average AUC greater than 60%, and b) the number of commits should be greater than 5,000. We want average AUC to be above 60% so that when anonymization is applied a model does not immediately become random (AUC 50%) but will retain some predictive power. Having more than 5,000 commits helps us ensure the graph has enough data points to be anonymized.

Based on this, we select the top 6 projects with the most commits. However, two selected Apache Hadoop projects recently went through a refactoring change that led the Hadoop repository to have 0 buggy commits, while the Hadoop-HDFS only had a total of only 603 commits (as opposed to 10,000). As such, we excluded these repositories. Although our criteria exclude the Qt dataset (the model has an AUC of less than 60%) we did include it in our study because of the massive size of this dataset and its important role as a benchmark in prior work [38, 40, 19].

4.2 **Data Collection**

The original datasets [25, 28] provide labels indicating which commits are buggy or not and the change metrics used for training the model. To determine the bug labels, both papers follow a similar methodology. The Apache

Dataset	Time Period	No. of Commits	Buggy Commits	Language
Qt	6/2011 - 3/2014	25,150	2002 (8%)	C++
OpenStack	11/2011 - 2/2014	12,374	1616 (13%)	Python
Cassandra	2003 - 2019	8,159	3,117 (38%)	Java
Flink	2003 - 2019	11,691	2,811 (24%)	Java
Groovy	2003 - 2019	8,059	1,614 (20%)	Java/Groovy
Ignite	2003 - 2019	12,036	2,439 (20%)	Java

Table 3: Summary of projects selected for the study.

dataset selects 14 popular Apache projects, based on stars on GitHub, while the OpenStack and Qt datasets were selected where at least 90% of the commits were part of a code review process. The respective authors then extract all the resolved or closed bugs filed in the projects' respective issue-tracking platforms, i.e., Gerrit for OpenStack/Qt and GitHub for Apache projects.

For these resolved bug reports, they obtain all commits referring to the corresponding bug report identifier, then consider the most recent such commit as the report's bug-fixing commit. On these bug-fixing commits, they apply filters to remove any erroneously detected commit, for example, commits changing trivial code (modifying only comments or removing white space) or changing over 100 files and/or 10,000 lines of code. Utilizing SZZ [50] on the bug-fixing commits, they were able to pinpoint the bug-inducing commits, which correspond to the labelled data points in our study.

However, the datasets did not provide the information needed by us to create the File and Person entities in the git-graph. To fetch those entities, we had to use the respective Gerrit (OpenStack [2] and Qt [3]) and PyDriller (Apache projects) APIs of the studied projects to extract the data needed to create the git-graph specified in Section 3.1. For each commit, we had to mine the number of files and the number of lines changed in them, as well as the identifier of the person who authored that commit.

Gerrit API: We call the Gerrit API for all the commits in the OpenStack and Qt datasets to obtain the details needed to form the git graph. After the knowledge graph has been created, we then label the *Commit* nodes as buggy or not according to the labels provided [28].

Git repository: Using the py-driller library [4], we mine each selected Apache repository and collect all the details needed for creating the git-graph. Using the labels provided by the ApacheJIT dataset, we mark the data as buggy or not.

4.3 **Measuring privacy**

4.3.1 ***Defining Privacy***

Privacy refers to how much information can be disclosed about the members of the data. Ideally, we do not want information about sensitive attributes data to leak. However, it is hard to avoid any leak 100%, hence, we want to

commit_id	**la**	**nf**	**nd**	**ns**
commit-1	[1-10]	[0-3]	[0-2]	[0-1]
commit-2	[11-20]	[4-6]	[3-5]	[2-3]
commit-3	[1-10]	[0-3]	[0-2]	[2-3]
commit-4	[1-10]	[7-9]	[0-2]	[0-1]
commit-5	[21-30]	[4-6]	[6-8]	[0-1]
commit-6	[11-20]	[4-6]	[6-8]	[4-5]
commit-7	[11-20]	[7-9]	[6-8]	[2-3]

Table 4: Example for Query Generation with Equal Frequency Binning

measure the amount of information that is leaking. We use the *la* feature as the sensitive feature because it has been observed that for defect prediction, the *number-of-lines-modified* attribute of a commit is closely related to bugs [24]. Furthermore, earlier work by Peters et al. on MORPH, LACE and LACE2 also used the number of source code lines feature as the main sensitive feature. We, therefore, use the lines added (*la*) and lines deleted (*ld*) attributes of a commit as the sensitive attributes in our study.

To measure privacy, we use the metric Increased Privacy Ratio (IPR) as used by Peters et al. [36, 26], similar to Adversarial Accuracy Gain [8]. This measure defines the ability of the attacker to extract information about sensitive attributes of the data, comparing the information gained by the attacker in the anonymized version to the non-anonymized version of the data. IPR can also be represented as a query made on the data before and after anonymization and whether this gives the same results or not.

A few other important definitions before defining IPR:

1. *Sensitive Attributes:* Attributes that we do not want attackers to identify in our dataset. These attributes can reveal sensitive information about the data. For our dataset, they are *la* and *ld*. When data is shared, it is assumed that the attacker does not have access to the sensitive attributes and is attempting to guess them.
2. *Quasi-identifier (QID's):* They, by themselves or along with other attributes from a dataset, can be used to identify entities in a dataset. For our dataset, it would be all the attributes except *la* and *ld*, therefore, *ns, nf, nd, etc.*

To calculate IPR in the dataset, we assume that the attacker has some knowledge about the value of the quasi-identifier of the entities they want to identify in the dataset. This knowledge becomes a query that we will perform on the data to calculate IPR.

4.3.2 ***Query generation***

Similar to Peters et al. [36], before generating the queries to model an attacker's knowledge (created for IPR), we apply Equal Frequency Binning to all attributes of the dataset under consideration. To explain the steps taken to create the query, we use Table 4 as an illustration. In this table, we take a small subset of all features and a few rows to explain how the queries used to calculate the IPR scores are constructred. Given this, we perform the following steps:

1. **Select the query size** to create queries. In this paper, we construct queries of sizes 1,2 and 4. For this example, we only show results for size 2. This means that this query will use 2 QIDs to query the data.
2. Randomly **select the quasi-identifiers** for query generation from all the available QIDs. In Table 2, we can see all the attributes available for selection. For this illustrative example, we use the sample shown in Table 4, from which we choose the attributes *nf* and *nd* as the two attributes.
3. For all selected attributes, **select one random sub-value range** from the possible ranges. In the sample Table 4, we randomly choose the range $[0-3]$ for the *nf* attribute and $[0-2]$ for the *nd* attribute.

In the example mentioned before, we hence generate a query with the conditions: $nf = [0-3]$ and $nd = [0-2]$. For each query, we ensure that the query at least returns one element and that the attribute value pairs are unique, i.e., there are no duplicate queries. If the query does not satisfy this condition, it is removed.

We generate queries with 1, 2 and 4 quasi-identifiers, and for each size, the number of queries is limited until all unique queries are exhausted or the number of queries generated reaches 1000 [36]. When generating queries, we only use the quasi-identifiers because the sensitive attributes are the ones the attacker is trying to identify.

4.3.3 ***Calculating IPR:***

For each query generated previously, we calculate if the query resulted in a breach. A query causes a breach if the most common value of the sensitive attributes (S^*_{max}) for the rows returned from the privatized dataset (G^*_i) matches that of the non-privatized dataset (returning rows G_i with the most common value as S_{max}). If this condition is met, the query is said to cause a breach.

$$\text{Breach}(G_i^*) = \begin{cases} 1, & \text{if } s_{\max(G_i)} = s^*_{\max(G_i^*)} \\ 0, & \text{otherwise.} \end{cases}$$

To calculate IPR, for all the queries, we find the percentage of queries that do not cause a breach. Using this, IPR is then calculated as:

$$\text{IPR} = (1 - (\text{Total number of breaches}/\text{Total number of queries run}) * 100)$$

4.4 Empirical Evaluation

4.4.1 Overall Approach

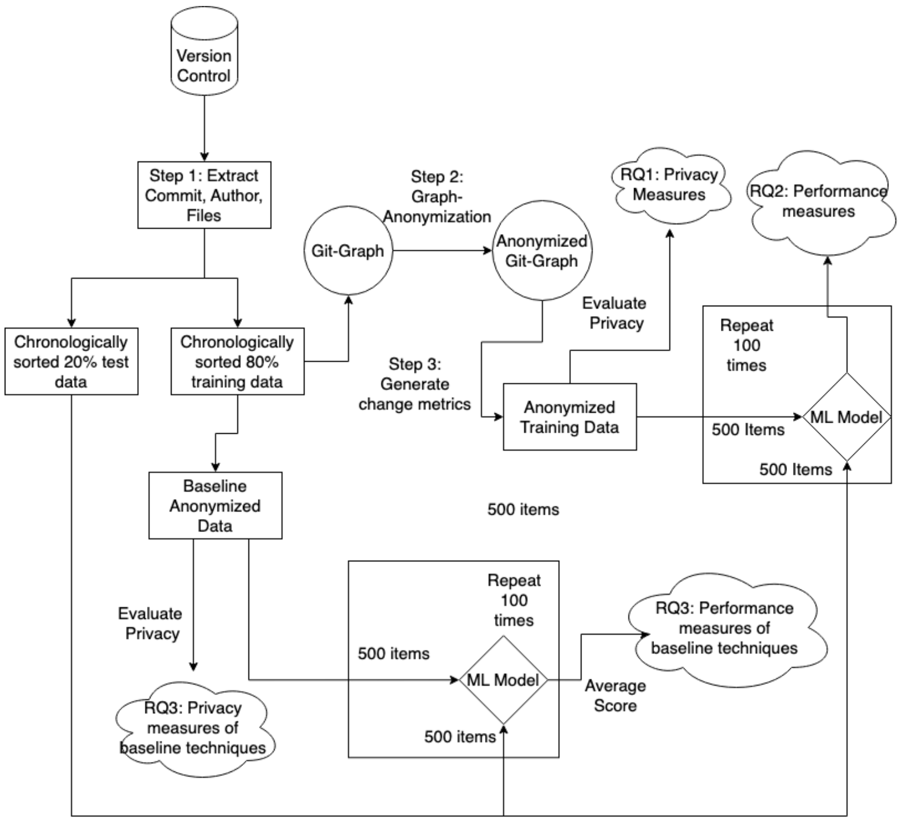

Version Control
Step 1: Extract Commit, Author, Files
Chronologically sorted 20% test data
Chronologically sorted 80% training data
Git-Graph
Step 2: Graph-Anonymization
Anonymized Git-Graph
Step 3: Generate change metrics
Anonymized Training Data
Evaluate Privacy
RQ1: Privacy Measures
RQ2: Performance measures
Repeat 100 times
500 Items
ML Model
500 Items
Baseline Anonymized Data
500 items
Evaluate Privacy
RQ3: Privacy measures of baseline techniques
Repeat 100 times
500 items
ML Model
500 items
Average Score
RQ3: Performance measures of baseline techniques

Fig. 4: Approach for answering the research questions

Tan et al. [47] highlighted that training/testing splits should be done chronologically. They highlighted that in a realistic scenario, at a time t, information will be available only about the commits that were created before it. Therefore, only previous commits should be used to predict whether a new commit is defective. The authors further highlighted how techniques like cross-validation use information from future data to determine if the current commit is buggy, which yields abnormally high precision. To combat these problems, we sort each project's data into a chronological 80/20 train/test split.

Figure 4 shows the general approach taken to answer the different research questions. As highlighted before, the first 80% of the training data is used to construct the git-graph (Step 1 in Section 3.1). The graph generated from the training data is then anonymized using one of the graph anonymization techniques (Step 2 in Section 3.2). Using this graph, we generate the anonymized JIT defect metrics (Step 3 in Section 3.3), which are used to train the models and calculate the privacy gained. We train a large number of models on the subsets of the data using the out-of-sample bootstrapping technique [48], which produces more robust models with less bias and variance. Each bootstrap iteration randomly samples 500 items with replacement from the training and testing dataset to evaluate the model's accuracy. We repeat this 100 times to ensure robustness and take the average of all the performance scores. This would ensure that the technique obtains consistent results.

4.4.2 ***Baseline Techniques***

Similar to Li et at., [26] and Yamamoto et al.,[54] to evaluate how our graph anonymization techniques compare with state-of-the-art privacy-preserving techniques, we use the different techniques provided by Peters et al., i.e., the MORPH, LACE and LACE2 algorithms [35, 37, 36]. We were unable to find the replication package for SRDO [26], therefore are not able to compare to it.

Similar to the approach for graph anonymization evaluation, we divide the baseline approach's data into chronologically sorted 80/20 train/test sets, then anonymize 80% of the training data. This anonymized data is used to calculate privacy and train models using out-of-sample bootstrapping. We train the model using 500 samples from the train data and use 500 samples from the test data to evaluate its performance. This process is repeated 100 times to get models with fewer biases and variance. To ensure a fair comparison, the test bootstraps used are the same across the graph anonymization and baseline techniques. To tune the performance of the baseline techniques we perform parameter optimization by testing all value combinations between 0 to 1 with 0.2-point increments. For the LACE and LACE2 experiments, we keep the CLIFF prune rate parameter as 0.4, which is the default value in the replication package. We also add the default values provided by the package for all the baseline techniques. Based on this, we report the baseline techniques' privacy and performance.

Inspired by other unsupervised learning approaches for JIT [21, 55], in order to adopt the ManualDown technique of Fan et al. [16] for JIT, we mark the top 50% of commits in the test set as buggy, referring to this as "MD50". For a fair comparison, we use the same test split as used for the GA and NGA techniques to measure the performance of the MD50 model.

4.5 **Graph Anonymization Configuration**

Each graph anonymization technique can be applied in different amounts (levels) or configurations, allowing finer-grained control of the trade-off between privacy and utility. In this section, we will define how different graph anonymization techniques will be applied in terms of configuration levels.

Random Add/Delete: In our initial experiments, we observed that Random Add/Delete was able to provide a high amount of anonymization when applied in increments of 20% (20, 40, 60, 80, 100). However, this did not allow us to observe how small amounts of anonymization changed the privacy and AUC metrics. Therefore, to see fine-grained changes in privacy, we apply anonymizations in increments of 2% up to 20%, then in increments of 10%.

Therefore, the configuration levels are 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 30, 40, 50, 60, 70, 80, 90, and 100. When anonymization is applied, we select x% of the *Commit* nodes (x represents the level chosen) and perform the technique on the *File* nodes to which they are linked. We then select the *Person* nodes and change the *Commit* nodes for which they are the authors.

Random Switch: Similar to Random Add/Delete, we apply random switch at levels: 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 30, 40, 50, 60, 70, 80, 90, and 100. This is also applied to the *Commit* nodes and the *Person* nodes in a configuration similar to Random Add/Delete.

k-DA Anonymization: In k-DA, we specify a number k, which becomes the minimum number of nodes each degree in the graph should have. For this, we specify k to have values of 2, 3, 5, 7, 10, 13, 15, 18, 21 and 24 [27]. The k-DA anonymity operations are applied to each *Commit* node and the *File* nodes to which it is connected, and for the *Person* nodes, the *Commit* nodes they have authored.

Generalization: Generalization creates clusters based on the graph structure. When analyzing the structure of a *Commit* node, we use the *File* node it is connected to and the author's experience, which is the number of nodes that the author has authored. Based on this structure, we ensure that each cluster has at least 10 commits. The number of clusters that need to be created are +10, +15, +20, +25, +30, +40, +50, +65, +80 and +100. This means that, for example, for a value of +20, the algorithm will ensure that it creates 20 new full clusters, each cluster containing at least 10 nodes. The clustering operation is performed on the *Commit* nodes alone. However, to make a cluster of a particular structure, we have to modify the *File* nodes that a *Commit* is connected to and the *Commit* nodes that its author has worked on.

4.6 Methodology for the Research Questions

In this section, we outline our approach to address the individual research questions we have posed.

4.6.1 ***RQ1: How do different graph anonymization techniques affect the privacy of the JIT defect prediction data?***

Motivation: The main aim of our work is to evaluate the extent to which graph anonymization techniques can provide privacy to the JIT metrics generated from the git-graph. As the anonymization techniques change the graph data, we want to evaluate if they are effective in providing privacy to the data.

What amount of graph anonymization needs to be applied to gain a sufficient amount of privacy? Does applying more anonymization always result in more privacy for the metrics?

Approach: To assess how different graph anonymization techniques provide anonymity to the git-graph, we first create the graph for each dataset in Neo4j. After the graph has been created, we make a backup of the graph. Then, for each dataset, we perform the following steps:

1. **Restore:** Using the backup created earlier, we recreate the database from the backup. This ensures that the graph is unaltered.
2. **Graph anonymize:** Depending on the anonymization technique currently being applied, we select the level (amount) of anonymization that needs to be applied to the graph. We then apply the anonymization techniques on the graph with the selected configuration value.
3. **Compute JIT metrics:** After anonymization, we compute the JIT metrics for each commit that is present in the graph. We repeat this and the above two steps for all the configuration levels of the anonymization technique before moving to the next step.
4. **Non-anonymized baseline:** As the non-anonymized version acts as the baseline to calculate the performance and measure privacy, we compute the JIT metrics from a graph where no anonymization has been applied.
5. **Measure privacy:** Using the various files generated in the previous steps, we compute the privacy metrics for each file using the non-anonymized version as the baseline.

The above steps are repeated for each technique and dataset. After collecting all the files, we run scripts to analyze the privacy provided by each technique. All the privacy measures for different datasets are aggregated per technique to understand how different techniques provide privacy to different datasets.

For the IPR metric, we divide the privacy attained into different levels. We found that Peters et al. [36] called any privacy score above 65% adequate, while Li et al. [26] used an adequacy limit of 80%. Therefore, our privacy levels are as follows:

1. **Privacy Level I:** 65 - 80%
2. **Privacy Level II:** 80 - 100%

If any technique provides Privacy Level I to a dataset, it is deemed to be adequate for sharing and the technique is feasible to use. In case data owners may wish to anonymize data further, we discuss Privacy Level II.

Each anonymization technique alters the dataset differently for its configuration values. This makes it hard to compare the effect different values have on the dataset since a configuration level of 20 for, say, k-DA does not reflect the same amount of graph anonymization as configuration level 20 of Random Switch. To allow for better comparison, we also calculate the number of graph links that were changed by each technique at each level when anonymizing the data. This helps to understand the number of links that were changed by each configuration value of a technique and how many link changes were needed to reach a particular privacy level. The number of links changed acts as a normalized scale to compare different techniques, instead of using their values.

4.6.2 ***RQ2: How do different graph anonymization techniques affect the performance of the JIT defect prediction model?***

Motivation: Graph anonymization techniques augment the graph to add privacy. With this research question, we want to quantify how augmentation changes the performance of the metrics. Are the changes the graph anonymization techniques make to the graph always destructive, or do they, in any case, improve performance as well? This will also help us understand if the performance loss and privacy gains can be balanced. Do different performance measures degrade at the same rate with increasing anonymization levels?

Approach: Using the anonymized JIT metrics generated in the previous research question, we bootstrap the training of multiple Random Forest Classifiers [5]. The details of how the model is trained are highlighted in Section 4.4 above. Once the model has been trained, we calculate the AUC, Recall, FPR and G-Mean metrics of the model to evaluate the impact graph anonymization has on these metrics [28, 24].

The Area Under the Curve (AUC) has been used in many JIT studies to describe a model's predictive power and is useful when there is an imbalanced training dataset. The value typically ranges from 0.5, equivalent to random guessing, to 1, which is perfect prediction power. The ROC curve, whose area is measured by AUC, plots a model's recall (Y axis) vs false positive rate (FPR), across different configurations of a model. Recall is the measure of the model's ability to correctly predict if a commit is buggy or not. It is important to understand how anonymization techniques change the ability of the data to detect bugs correctly. FPR is the measure of how many bugs are incorrectly identified as buggy. It is calculated as $\frac{FP}{FP+TN}$. The lower this measure, the better the model performance, as that would signify fewer commits are getting incorrectly classified as buggy, thereby wasting less time in fixing the wrong commits. Finally, we also use the G-Mean as it is useful in the evaluation of imbalanced datasets. It is the geometric mean of sensitivity (recall) and specificity ($\frac{TN}{FP+TN}$) of the model. The higher this metric, the better the model performance.

It is important to study the changes that anonymization brings to the models' performance as any anonymization applied without any utility function, i.e., model performance metric, can be harmful, as there would be no way to detect the effect of privacy. Privacy and utility cannot be considered in isolation [13].

As explained above in RQ1 design, the number of graph links changed acts as a normalized way to compare different configuration values for different techniques. Therefore, here too, we study the effect of graph link changes on the performance of the datasets. This is important as it will help us understand how destructive particular anonymization techniques are and establish a relation between the number of graph links changed and changes in the AUC and Recall.

We conduct a Kruskall-Wallis test ($\alpha = 0.05$) with Dunn's posthoc test to find the best-performing technique in terms of preserving AUC, Recall, G-Mean and FPR. For each dataset, we take all the configuration values for each technique that are able to provide privacy scores greater than 65%. These filtered values are then compared to one another to find the best-performing technique in terms of AUC and Recall. We also perform the Spearman correlation between increasing configuration values and the performance metrics to quantify the possible relation between how the performance changes with increasing anonymization levels.

4.6.3 ***RQ3: How do the graph anonymization techniques compare to the state-of-the-art anonymization techniques?***

Motivation: We want to compare how the graph anonymization techniques work in comparison to the state-of-the-art anonymization techniques for JIT models. Do the graph anonymization techniques achieve privacy scores that are comparable to the state-of-the-art techniques? Furthermore, we want to compare the difference in performance that comes with using these techniques when compared to graph anonymization techniques.

Approach: As highlighted in 4.4.2, to evaluate how the state-of-the-art techniques compare to graph anonymization techniques, we were able to find the replication packages for MORPH, LACE and LACE2 [6] (collectively referred to as non-graph anonymization (NGA) techniques) for evaluation on the JIT defect prediction metrics. As fine-tuning the NGA techniques results in different models, we compare the data points that are able to provide Privacy Level I and Privacy Level II and their median AUC, Recall, FPR and G-Mean scores for different datasets.

Furthermore, to understand the statistical difference between the AUC, Recall and IPR scores of the NGA and GA (graph anonymization) techniques, we compare them in two different ways. All the comparisons are performed per dataset, and for each comparison, we consider the configuration values that have an IPR score greater than 65%. First, for the AUC, Recall, and IPR scores separately, we combine all of the different NGA techniques' values into one distribution (for one metric at a time, e.g., AUC) and compare that with all the GA technique values combined into one distribution (for one metric at a time, e.g., AUC) to understand, overall which group performs better using the Mann-Whitney U test. Second, we compare all of the techniques (NGA+GA) among themselves to evaluate how they compare to one another using the Kruskall-Wallis test with Dunn's posthoc test. This is to identify the technique that works best among all. For all tests, we use the $\alpha = 0.05$.

We compare the performance of the model trained using GA-anonymized data with the MD50 Manual Down model as well. For this MD50 model, as it requires no training, we only need the test data set [21, 55], then mark the top 50% of the test set commits with the highest “number of lines added” feature values as buggy, before comparing with the oracle for that test set. This process is repeated 100 times for each bootstrap iteration to obtain the performance distribution for the MD50 model.

The graph techniques have multiple configurations, each with its own bootstrapped model performance. Hence, for each dataset anonymized with a GA technique, we find the best performing configuration value (at least Privacy Level I) out of all the configurations. We do this by statistically comparing the bootstrap performance distributions across each configuration, giving us the best performing configuration for each GA technique and dataset combination. The performance of each such combination is then compared to the AUC, G-Mean, Recall and FPR performance metrics of MD50 using the Wilcoxon signed-rank test ($\alpha = 0.05$) with Bonferroni correction.

5 Empirical Study Results

In this section, we discuss the results of the research questions we set out to answer.

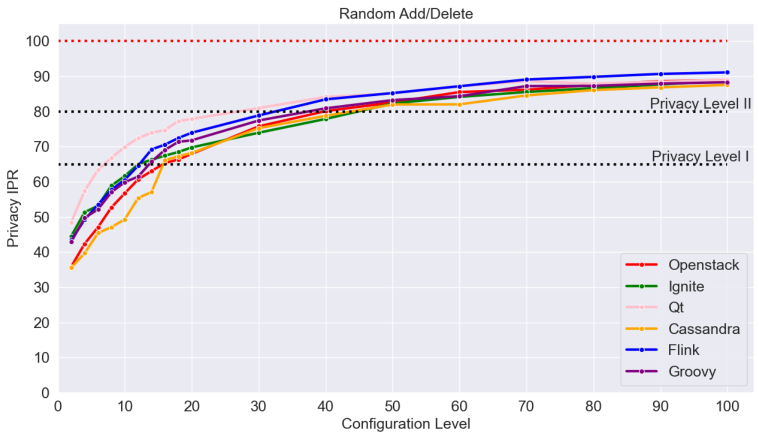
5.1 RQ1: How do different graph anonymization techniques affect the privacy of the JIT defect prediction data?

Overall, our results show that graph anonymization techniques are effective in providing privacy to the JIT metrics. Each technique is able to provide Privacy Level I (IPR scores of 65% or greater) to all the datasets. Furthermore, for every dataset, there is more than one technique that is able to provide Privacy Level II (IPR scores of 80% or greater).

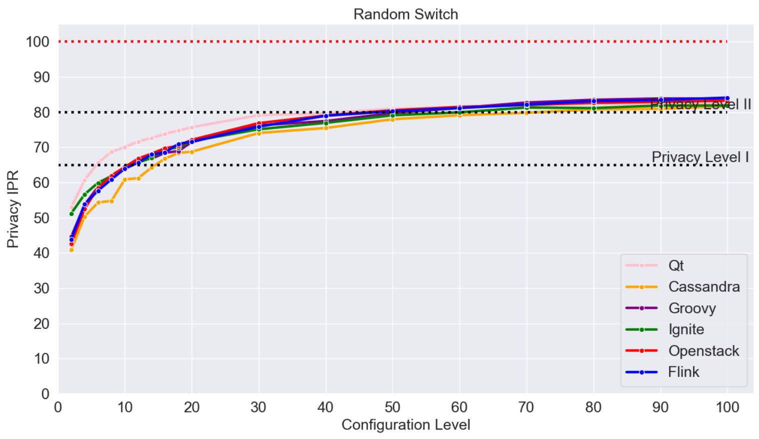
Figure 5 shows that the Random Switch and Random Add/Delete techniques are able to provide both Privacy Levels I and II to all the datasets. We see that both techniques have a similar effect, where with increasing levels of anonymization, the privacy scores keep increasing. At 100% anonymized nodes, Random Add/Delete is able to provide privacy scores of almost 90%.

However, for both techniques, we see an elbow-shaped curve in the amount of privacy gained with increasing levels, indicating diminishing returns in privacy gains with increasing anonymization levels. This is because even when increasingly more nodes are changing, the anonymization may affect nodes that have already been altered and therefore do not dramatically increase the privacy scores anymore. We see this similar trend for all techniques, where with increasing anonymization levels, they gain lesser amounts of privacy.

With little effort spent on anonymization, both Random Switch and Random Add/Delete techniques are able to achieve high privacy scores. To gain Privacy Level I, less than 20% of the nodes are to be



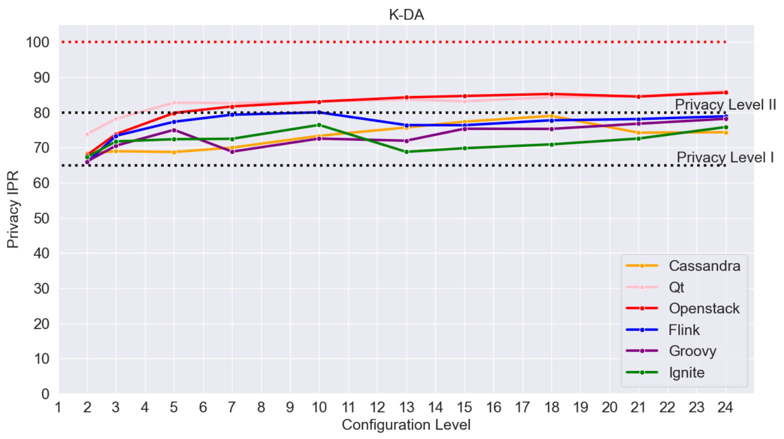
(a) Random Add/Delete



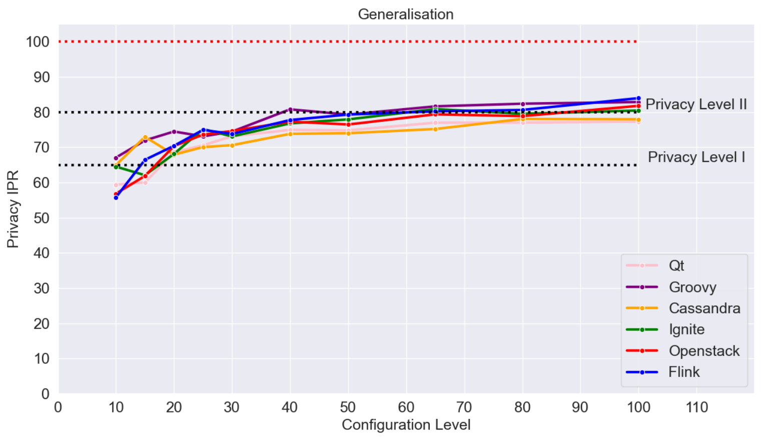
(b) Random Switch

Fig. 5: IPR scores of all the datasets for Random Add/Delete and Random Switch. The vertical-dotted lines are the two privacy levels.

anonymized for both. However, there is a difference in the number of nodes to be anonymized to gain Privacy Level II, where Random Switch requires anonymization levels of 60/70% as compared to Random Add/Delete, which does for 40/50%, i.e., with lesser nodes anonymized. This difference is because Random Add/Delete causes more change in the graph as it destroys and redistributes the links for the various nodes. This is in contrast to Random Switch, which simply switches the nodes around, keeping the same distribution of links.



(a) k-DA



(b) Generalisation

Fig. 6: IPR scores of all the datasets for k-DA and Generalization. The vertical-dotted lines are the two privacy levels.

While k-DA only reaches Privacy Level II for half of the datasets, it is able to provide Privacy Level I to all the datasets at the lowest configuration level ($k = 2$). Figure 6a shows how all projects quickly increase their IPR above 65%, yet taper off relatively quickly. This is because, unlike Random Add/Delete and Random Switch, k-DA operates on all the nodes in the graph that do not satisfy the k-DA anonymity condition. As all the nodes in the graph are modified, it leads the graph to gain privacy quicker than Random Add/Delete or Random Switch, which only operates on $x\%$ of the nodes at a time.

	Random Add Delete		Random Switch		k-DA		Generalization	
	Level I	**Level II**	**Level I**	**Level II**	**Level I**	**Level II**	**Level I**	**Level II**
Cassandra	16/10492	50/24136	16 /18048	80 /83594	2 /6081	NA	15 /9424	NA
Flink	14 /11904	40 /36966	12 /27840	50 /113154	2 /8060	10 /40255	15 /6813	65 /32996
Groovy	14 /5140	40 /11610	12 /7120	60 /41296	2 /5757	NA	10 /3218	40 /13082
Ignite	12 /7996	50 /36306	12 /22762	70 /129756	2 /9449	NA	20 /5668	65 /41083
OpenStack	16 /14712	50 /48366	12 /36540	60 /171220	2 /9725	7 /74104	20 /12155	100 /47938
Qt	8 /46936	30 /144314	6 /82972	40 /491580	2 /119411	5 /514410	20 /12912	NA

Table 5: "x/y" represents a configuration value x and its corresponding number of link changes y, for different IPR Privacy Levels. The cells in green represent the technique achieving that Privacy Level with the least number of links changed for a given project.

With increasing values of k, the gains in privacy of k-DA are limited. One of the possible reasons is that at higher values of k, the same characteristics of the graph are further amplified, causing a limited change in the metrics. For example, at $k = 5$, there are 5 commits that modify 10 files, making the nf attribute of those commits 10. At $k = 21$, there are now 21 commits that modify 10 files. While according to the IPR metrics, some query results still do change as there is an increase in privacy measures. However, overall the benefits from higher values of k are limited as the same characteristics of the graph are retained. This is the reason why only 3/6 datasets gain Privacy Level II.

Similar to k-DA, in Figure 6b, we see that Generalization provides higher privacy scores than Random Add/Delete and Random Switch at the lowest configuration level. At configuration level 20, it is able to provide Privacy Level I to all the datasets. Furthermore, at configuration level 100, it provides privacy scores greater than 80% to 4 out of the 6 datasets.

Across all techniques, Generalization requires changing the least amount of links to achieve Privacy Level I (4/6), followed by k-DA (2/6). In Table 5, we can see the number of links changed by each technique for each dataset to achieve a particular Privacy Level. For Privacy Level II, it is able to provide privacy with the least link changes for 3 out of the 6 datasets, with Random Add/Delete doing so for the remaining three datasets. For Privacy Level II, we see that Generalization and Random Add/Delete perform equally well, providing privacy with the least links changed to three datasets each. Every technique changes a different number of links to reach a particular privacy level. Across all the datasets, Random Switch changes the most links to achieve Privacy Level I (Table 5). For instance, in Figure 7, we see the number of links changed to achieve different privacy levels for the Apache Ignite dataset. Generalization is able to provide almost Privacy Level I for 4,285 links changed in the Apache Ignite datasets. At that same number of links changed, privacy provided by Random Add/Delete (59%), and Random Switch (51%) is lower.

To understand how invasively graph anonymization changes the distribution of values of a given feature between the non-anonymized data and

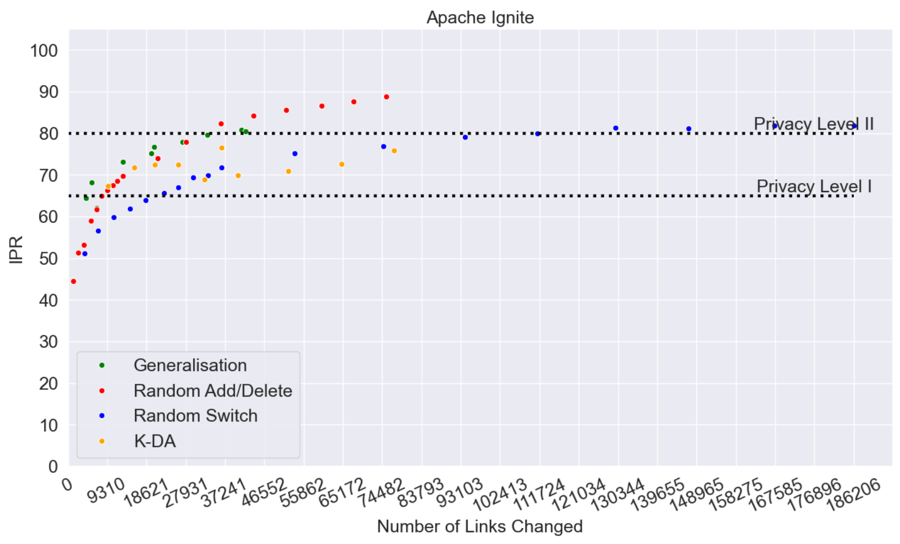


Fig. 7: The number of links changed in Apache Ignite to reach different privacy levels

anonymization configuration level 1, we compare, for each feature, its non-anonymized distribution of feature values to the anonymized distribution of feature values. For example, we compare OpenStack's k-DA anonymized *la* feature's distribution ($k = 2$) with the non-anonymized distribution. Wilcoxon tests (with $\alpha = 0.01$) on each dataset and for each feature show that each technique at its lowest configuration level changes at least 3 features' distribution in a significant manner. This shows that anonymization causes either the quasi-identifiers or the sensitive feature distribution to change, leading the query to give different results and an increase in IPR scores.

Summary of Research Question 1

Graph anonymization techniques are effective in providing privacy to the JIT metrics. All techniques are able to provide privacy scores greater than 65% to all the datasets. For each dataset, there is one technique that achieves a privacy score greater than 80%. Random Switch and Random Add/Delete are able to provide Privacy Level II to all the datasets, with only Random Add/Delete being able to provide privacy scores close to 90% to all datasets. Generalization is able to provide both privacy levels with the least number of links changed in (7/12) cases, followed by Random Add/Delete (3/12) and k-DA (2/12).

	Random Add Delete		**Random Switch**		**k-DA**		**Generalization**	
	Level I	**Level II**	**Level I**	**Level II**	**Level I**	**Level II**	**Level I**	**Level II**
Cassandra	65.85 (-2.53)	64.78 (-4.11)	63.1 (-6.67)	57.98 (-14.24)	67.6 (0.13)	NA	67.52 (0.01)	NA
Flink	70.68 (-1.92)	69.24 (-3.93)	68.68 (-4.7)	64.54 (-10.46)	71.04 (-1.45)	70.26 (-2.52)	72.19 (0.36)	71.81 (-0.17)
Groovy	60.63 (-5.06)	59.89 (-6.22)	60.34 (-5.63)	57.65 (-9.84)	61.54 (-3.36)	NA	62.8 (-1.73)	62.26 (-2.56)
Ignite	72.0 (-2.94)	68.38 (-7.82)	67.5 (-8.88)	59.76 (-19.32)	73.46 (-0.61)	NA	73.73 (-0.51)	71.5 (-3.52)
OpenStack	61.19 (-3.85)	60.23 (-5.36)	59.72 (-6.28)	56.18 (-11.84)	59.6 (-6.41)	58.62 (-7.95)	63.95 (0.39)	62.76 (-1.48)
Qt	55.42 (-3.37)	53.77 (-6.24)	54.62 (-5.0)	51.8 (-9.89)	54.69 (-4.84)	54.58 (-5.02)	56.42 (-1.46)	NA

Table 6: Median AUC score with the relative change in AUC from the non-anonymized baseline in parentheses at different Privacy Levels.

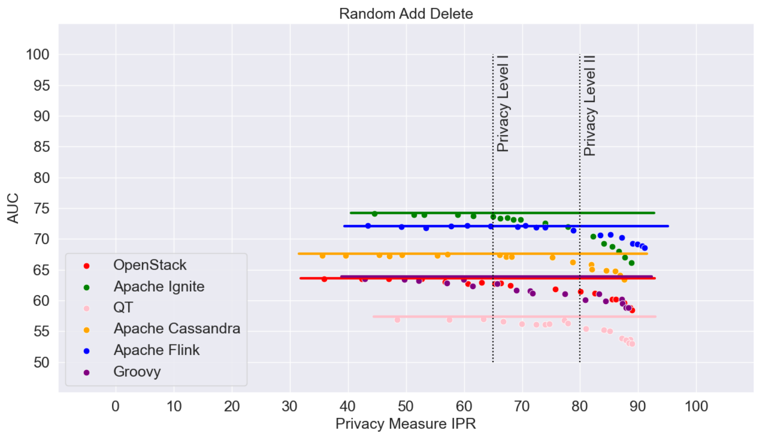
	Random Add Delete		**Random Switch**		**k-DA**		**Generalization**	
	Level I	**Level II**	**Level I**	**Level II**	**Level I**	**Level II**	**Level I**	**Level II**
Cassandra	85.39 (-1.43)	83.49 (-3.62)	85.88 (-1.17)	78.06 (-10.17)	85.18 (-2.02)	NA	84.45 (-2.65)	NA
Flink	66.26 (-6.31)	65.72 (-7.07)	67.2 (-5.41)	66.08 (-6.98)	61.84 (-11.88)	59.78 (-14.82)	66.43 (-6.2)	66.4 (-6.24)
Groovy	32.82 (-16.9)	30.49 (-22.81)	35.56 (-9.7)	32.77 (-16.79)	33.72 (-13.04)	NA	37.27 (-6.12)	36.94 (-6.96)
Ignite	72.8 (-4.24)	66.14 (-13.0)	71.65 (-5.38)	59.61 (-21.28)	68.06 (-9.76)	NA	64.95 (-14.38)	57.79 (-23.82)
OpenStack	45.42 (-10.78)	42.98 (-15.58)	44.93 (-11.83)	39.1 (-23.28)	32.72 (-35.73)	30.3 (-40.48)	52.06 (1.96)	46.36 (-9.2)
Qt	20.32 (-21.85)	16.96 (-34.75)	18.97 (-27.82)	13.75 (-47.68)	13.21 (-49.64)	12.72 (-51.49)	21.4 (-17.29)	NA

Table 7: Median Recall score with the relative change in Recall from the non-anonymized baseline in parentheses at different Privacy Levels.

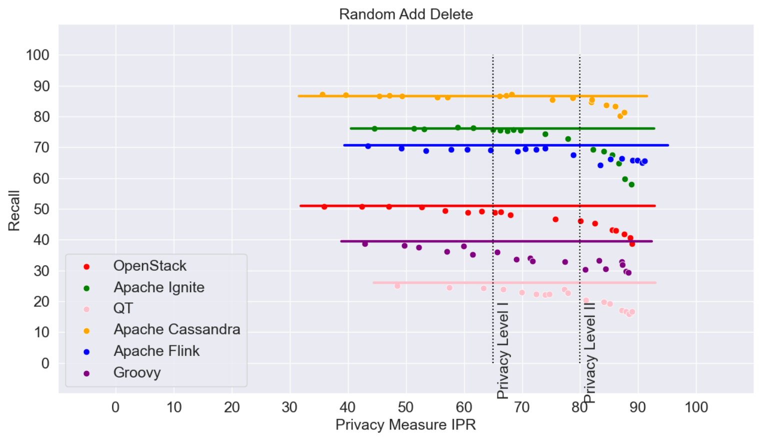
5.2 **RQ2: How do different graph anonymization techniques affect the performance of the JIT defect prediction model?**

Across all projects, graph anonymization techniques that reach Privacy Level I, see a median decrease in AUC of 1.45% and in Recall of 5.35% (average of 1.98% AUC and 7.71% Recall). Furthermore, reaching Privacy Level II involves a median drop in AUC of 6.44% and in Recall of 20.29% (average of 7.28% AUC and of 22.42% Recall). In Table 6 and Table 7, we can see the effect of the median AUC and Recall to achieve a particular privacy level for all the datasets. The best-performing techniques at their best-performing configuration are able to reach Privacy Level I, on average, with a 1.3% increase in AUC and a 4.45% increase in Recall across all datasets. The best-performing techniques at their best-performing configuration reach Privacy Level II across all datasets with an average decrease of 1.37% in AUC and 5.51% in Recall.

Random Add/Delete is able to provide Privacy Level I with a median decrease of 1.35% and 3.93% in AUC and Recall (on average 1.63% AUC and 6.05% Recall), and Privacy Level II with a median decrease of 5.36% and 15.58% in AUC and Recall (average 5.54% AUC and 16.65% Recall). As seen in Figure 8, Random Add/Delete provides an IPR score of almost 90% for all the datasets at 100% anonymized nodes. However, across projects, at this privacy score (near 90%), on average, the AUC and Recall decrease by 7.59% and 20.45%, respectively. Since any dataset with privacy scores of 65% or greater can be further shared, the graph anonymization techniques provide users with the ability to tweak the amount of anonymization performed to find a good trade-off between performance and privacy.



(a) AUC Score and Privacy



(b) Recall and Privacy

Fig. 8: AUC/Recall vs IPR with Random Add/Delete applied. The horizontal, coloured solid line is the performance of the non-anonymized (baseline) model. The vertical-dotted lines are the different privacy levels.

Random Switch has a higher impact on the performance than Random Add/Delete for all the datasets, where AUC and Recall decrease by a median value of 3.25% and 5.06% for Privacy Level I and 11.25% and 20.12% for Privacy Level II across all datasets[1]. One reason for this difference could be that Random Add/Delete is able to provide

[1] To save space, we do not show the corresponding plots for Random Switch. They can be found in the online replication package [7].

privacy with fewer link changes (as seen in Table 5) than Random Switch. This implies that Random Add/Delete causes less change to the graph structure and therefore does not degrade performance as severely as Random Switch.

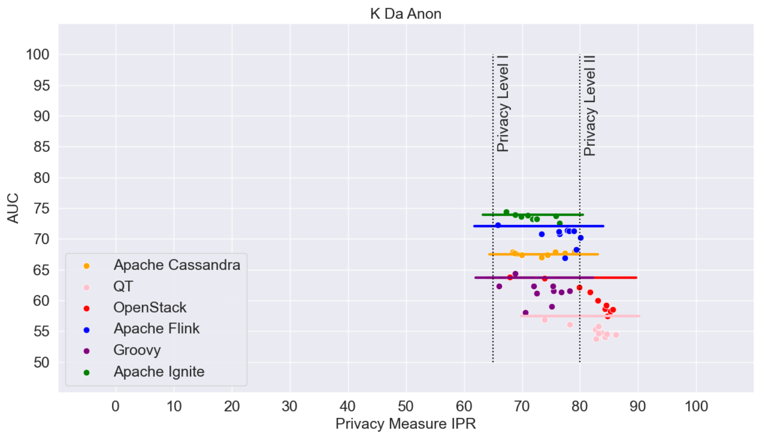
For Random Add/Delete and Random Switch, we find that there is always a strong negative linear correlation (Spearman correlation $\geq$ 0.8) between the anonymization level and the AUC and Recall. As seen in Figure 8 for Random Add/Delete, we see that with increasing privacy scores, the AUC and Recall keep on falling. At 100% anonymized nodes, on average, the AUC is reduced by 7.59%, and Recall by 20.45% across all datasets. We observe similar trends for Random Switch as well, where, at 100% anonymization level, on average, the AUC decreases by 15.18% and Recall decreases by 27.54%. Random Switch has a more dramatic drop in AUC and Recall with an increase in privacy. The k-DA and Generalization techniques show either a weak or moderate negative correlation (with the exception of k-DA for OpenStack and Flink, which show a strong correlation).

We believe that the AUC and Recall keep falling with increasing anonymization because, as Random Add/Delete and Random Switch randomly changes the structure of the data, they break the context that is present in the graph. Changes to this context progressively cause the model to be confused and lead to poor performance. At lower configuration levels, as the changes are less invasive, the performance of the model is not significantly impacted. As the amount of anonymization increases, more changes are made to the graph, which comes at the cost of performance.

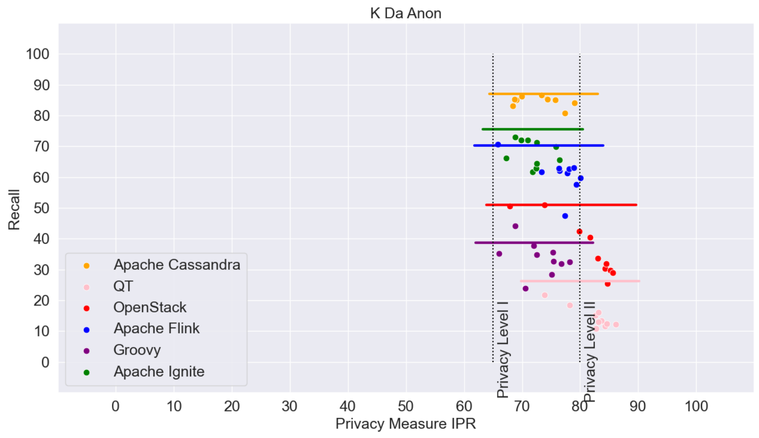
The k-DA technique's impact on Recall (Figure 9b) is the highest of all the techniques, with a median decrease in Recall of 9.78% and 42.83% for the two privacy levels across all datasets. For Privacy Level II, even the best-performing configuration for k-DA decreases Recall by up to 25% on average for OpenStack, QT and Flink. At the same privacy level, other techniques incur less loss in Recall; for example, Random Add/Delete drops recall by 12% on average for the same three datasets. However, k-DA's impact on AUC (Figure 9a) is not the highest, with AUC decreasing by a median of 0.9% and 5.49% across the two privacy levels. k-DA is only able to provide Privacy Level II to 3 datasets (Flink, OpenStack and QT), with a median AUC loss in the range of 2.52 - 7.95%.

Similar to k-DA, Generalization is able to provide Privacy Level I with a median increase of 0.1% AUC and a median decrease of 5.15% Recall. For Privacy Level II, it sees a median decrease of 1.89% in AUC and 8.57% in Recall. As seen in Figure 10, Generalization is unable to provide Privacy Level II to Cassandra and Qt projects even at configuration level 100. However, the privacy scores reach about 77%, with the AUC being impacted by 0.19% and -2.03%, respectively.

Generalization preserves significantly more AUC and Recall than other techniques. For a particular project, when Generalization is used for anonymization, there is a statistical difference in the AUC when compared to the other anonymization techniques, with AUC being higher for Generalization in 4/6 projects, followed by k-DA, which performs better in 2/6 projects.



(a) AUC Score and Privacy



(b) Recall and Privacy

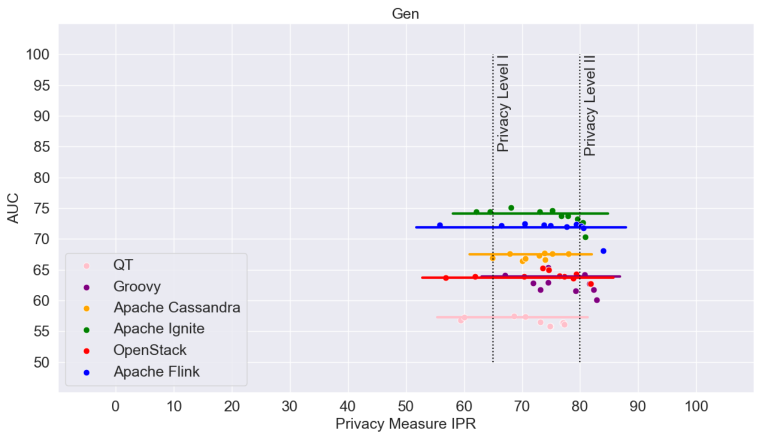
Fig. 9: AUC/Recall vs IPR with k-DA applied. The horizontal, coloured solid line is the performance of the non-anonymized (baseline) model. The vertical-dotted lines are the different privacy levels.

For Recall, Generalization's scores are statistically significantly higher when compared to other techniques in 3/6 projects, with Random Add/Delete being in 1/6 projects. In the remaining two projects, we observe insignificant differences. One reason why Generalization is able to provide privacy with the least change in performance is that it provides the different Privacy Levels with the least number of changes (Table 5). Therefore, the data derived from the anonymized graph undergoes fewer changes compared to others and loses its predictive power.

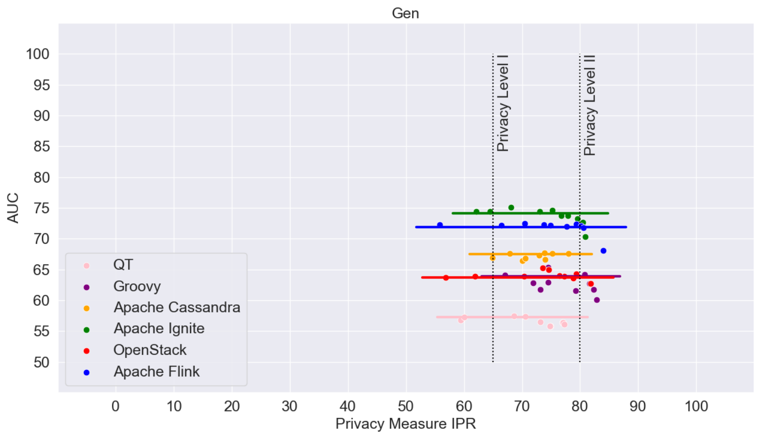
The graph anonymization techniques across all projects and configuration points decrease the G-Mean by a median of 2.29% and 11.76% for Privacy Level I and Level II. G-Mean shows similar trends as AUC and Recall, where with increasing anonymization levels, the model's performance G-Mean score decreases. Generalization and k-DA perform the best in terms of preserving G-Mean, being significantly better in 4 and 2 out of 6 cases, respectively. The FPR decreases (i.e., improves) by a median of 4.42% and 3.13% for Privacy Level I and Level II. However, we should keep in mind that at the same time, the Recall score decreases by 5.31% and 20.29% for Privacy Level I and Privacy Level II, reducing the performance of the model as a whole. For example, we see that k-DA statistically performs the best for FPR, reducing it (i.e., improving it) by a median of 11% and 52% for Privacy Level I and Privacy Level II. Yet, simultaneously, the Recall decreases by a median of 9.78% and 42.83%, negating the benefits. We see similar behaviour of the FPR metric for other techniques. The graphs and tables for these two metrics can be found in our online appendix [7].

In Figure 11, we can see how many links are changed by the techniques for the Apache Groovy dataset and the effect on the AUC and Recall of the model. For a value of 12k changed links in Figure 11, Generalization and Random Switch have less impact on the AUC as compared to Random Add/Delete and k-DA. Both Generalization (0.52%) and Random Switch (1.06%) change the AUC negligibly. However, the IPR score provided by Generalization at the same level is 80% compared to 71% for Random Switch. Furthermore, at 12k changed links, Random Add/Delete is able to provide IPR scores of 80%, but the loss in AUC is almost 6%. We see similar trends for the other datasets, where for a similar number of changed links, Generalization is able to provide high privacy scores with the least impact on performance, followed by Random Add/Delete. The graphs for the remaining datasets can be found in our online replication package [7].

We note that the recall metric observes a higher impact in certain cases, especially Qt. To calculate how much the recall metric changes with changing anonymization level, for each technique, we calculate the effect size of each configuration point when compared to the non-anonymized baseline. To calculate this for each dataset, we compare the distribution of the 100 bootstrapped models for each configuration point (for example, RAD 30) to the non-anonymized performance and calculate the effect size. We find that the change in Qt's recall is not significantly different than for the other projects. For example, in Table 7 we see that Random Add/Delete decreases the Qt project's Recall more when compared to the other projects (-21% for Privacy Level II). Yet, for Qt, when statistically comparing the 18 configuration points with Random Add/Delete to the non-anonymized baseline, the effect size for the recall is insignificant, negligible or small for 13 configurations. This is similar to OpenStack's 12, Flink's 13, Groovy's 14 and Ignite's 12 configuration points. We see similar behaviour for other techniques, where the relative



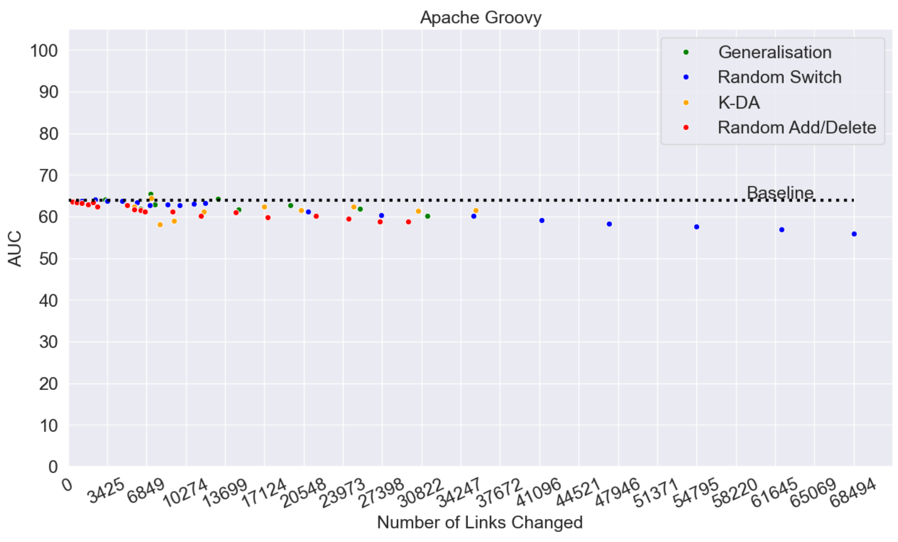
(a) AUC



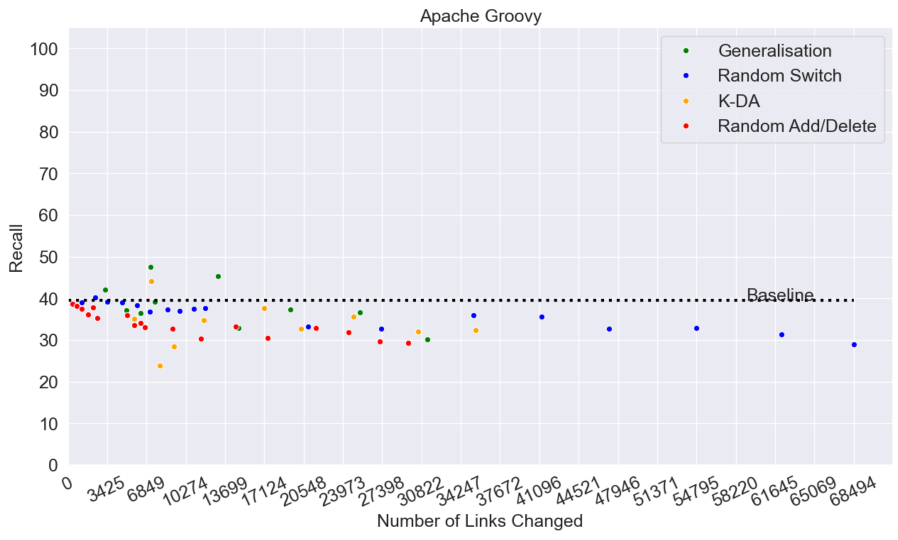
(b) Recall

Fig. 10: AUC/Recall with IPR with Generalization. The horizontal, coloured solid line is the baseline (non-anonymized). The vertical-dotted lines are the different privacy levels.

change in Qt's recall score is similar to other projects. For example, for k-DA it is small in 1/10 cases same as Flink and OpenStack. Only for Random Switch the change in recall is higher compared to others, being small in 9/18 cases compared to the others being small in 12 or more. To conclude, even though the effect on Qt's recall may seem higher, the change is not substantially different from what other projects observe and depends on the technique being used.



(a) AUC Score



(b) Recall

Fig. 11: AUC/Recall versus the number of links changed for Apache Groovy, for all the techniques.

Summary of Research Question 2

The graph anonymization techniques, across all datasets, are able to provide Privacy Level I with a median drop in AUC of less than 2% and Recall of less than 6%, and Privacy Level II median drop in AUC of less than 7% and Recall of 21%. Random Add/Delete and Random Switch see a linear decrease in performance with increasing anonymization. k-DA sees the largest drop in Recall out of all the techniques. Generalization is able to provide all privacy levels with the least significant drop in AUC and Recall.

5.3 RQ3: How do the graph anonymization techniques compare to the state-of-the-art anonymization techniques?

For each project, with statistical significance, the MORPH, LACE and LACE2 (NGA) techniques provide higher privacy scores when compared to the graph anonymization techniques, with the NGA privacy scores in the range of 89% - 99% (Privacy Level II). As seen in Figure 12 for Apache Ignite, the NGA techniques are able to provide privacy scores in the range of 91% to 94% for all configurations. The graphs for other projects are available in the online replication package [7].

In Table 10, we see the aggregated results for the different GA techniques when compared to the three NGA techniques across the six datasets. The results are reported as win/tie/loss with respect to the row header technique, where win indicates that the distribution was statistically significantly higher for the row header technique (e.g., Random Add Delete), tie indicates that there was no statistical difference, and loss means that there was a statistically significant drop for the row header technique (e.g., Random Add Delete).

GA techniques provide significantly less privacy than MORPH, LACE and LACE2, with the latter techniques having significantly higher privacy scores in more than 16/18 cases. Only Random Add/Delete has comparable privacy results to MORPH, where the privacy results are statistically insignificant to MORPH in 5/6 cases.

On the other hand, the impact of NGA techniques on the AUC (Table 8), and Recall (Table 9), is significantly higher than that of the GA techniques. As seen in Table 10, for AUC, the results are either insignificant or favour GA techniques, with Generalization performing better than NGA techniques in 17/18 cases. For Recall, we see similar results where each GA technique performs better in at least 14/18 cases, except k-DA, which does so in 10/18 cases. Interestingly, only for the Qt dataset, MORPH (NGA) performs better for Recall than k-DA, Random Add/Delete and Random Switch (GA techniques), however, it still does not perform better than Generalization in that case.

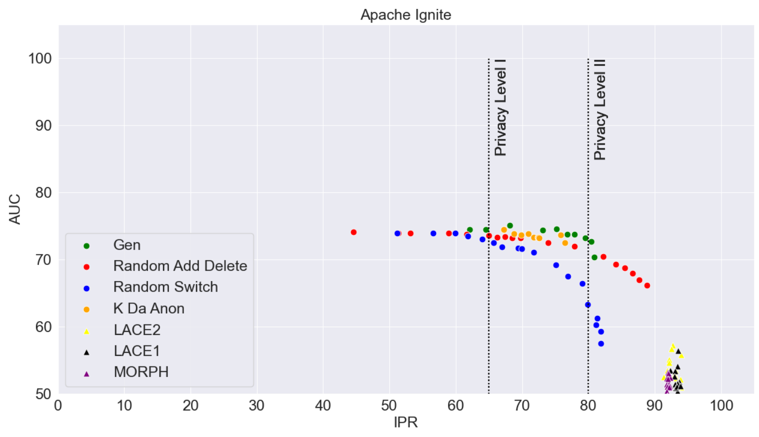
Across all datasets, the NGA techniques provide Privacy Level II with AUC and Recall dropping by a median of 21.15% and 80.34% (on average 19.81% and 66.63%). In contrast, for the same privacy level, the GA techniques suffer only a median loss of 6.44% and 20.29% for AUC and Recall, respectively. As NGA techniques always provide Privacy Level II, they have the same impact on performance for the lower privacy levels. The GA techniques provide Privacy Level I with a 1.45% and 5.35% median decrease in AUC and Recall. For example, as seen in Figure 12, the best-performing GA technique for Apache Ignite provides Privacy Level II with a 5% decrease in AUC and a 9% decrease in Recall. In contrast, the best-performing NGA techniques do so with a 24% and 73% decrease in AUC and Recall. Furthermore, for Privacy Level I, the impact of the best-performing GA technique on both performance metrics is less than 1%, for instance, Random Add/Delete.

The NGA techniques decrease the G-Mean by a median of 70.07% across all projects and configuration points. For the FPR metric, the NGA techniques decrease it by 82.43% (i.e., improving it). However, as highlighted in RQ2, FPR needs to be considered along with Recall, where the gains in FPR are negated by a corresponding decrease in Recall, making the model's performance degrade as a whole. As seen in Table 10, statistical tests show that for both G-Mean and FPR metrics, the GA techniques perform significantly better than the NGA techniques in the majority of cases (80% and higher). The plots for each dataset can be found in our online replication package [7].

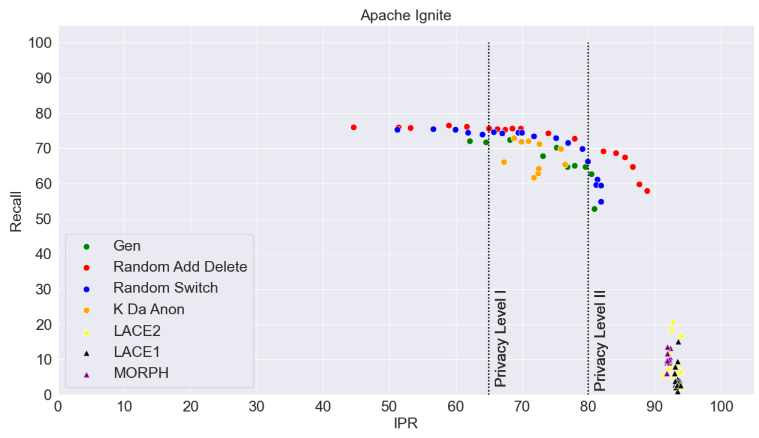
The NGA techniques do not offer any choice over the amount of anonymization and privacy. They provide high privacy with severe performance degradation and do not offer any control over either. In contrast, GA techniques offer choices where the end users can choose Privacy Level I with almost no change in performance or Privacy Level II with slight changes in performance, simply by varying the configuration level values. Graph anonymization techniques are able to provide privacy scores close to 90%, with a lower impact on performance. For example, the Random Add/Delete technique is able to provide the Apache Groovy dataset with an IPR score of 88% and an AUC score of 60% (-5.7% from the baseline). This is in contrast to non-graph anonymization techniques where the best performing NGA technique provides a privacy score of 97.5% but the AUC is reduced to 52% (-17.7% decrease from baseline).

It is important to highlight that even though GA techniques provide statistically lower privacy, they are still able to provide Privacy Level I and Privacy Level II to all the datasets, which is private enough to be shared with others [36]. GA techniques achieve the same privacy levels as NGA techniques, with significantly higher AUC, Recall and G-Mean scores.

In Table 11, we show the comparison of the graph techniques to MD50. The results are reported as the aggregation of #win/tie/loss for the six studied projects, where a win means that a best-performing GA configuration performed significantly better than MD50 for a project, a tie means that a best-performing GA configuration and MD50 technique performed equally well



(a) AUC Score



(b) Recall

Fig. 12: AUC/Recall versus IPR score of Apache Ignite with graph and non-graph anonymization techniques.

	RAD	**RS**	**K-DA**	**Gen**	**MORPH**	**LACE**	**LACE2**
Cassandra	64.785 (-4.11)	57.98 (-14.24)	NA	NA	53.87 (-20.32)	64.89 (-4.02)	54.5 (-19.39)
Flink	69.24 (-3.93)	64.535 (-10.46)	70.26 (-2.52)	71.81 (-0.17)	51.46 (-28.46)	55.69 (-22.58)	52.26 (-27.33)
Groovy	59.89 (-6.22)	57.65 (-9.84)	NA	62.265 (-2.56)	49.66 (-22.28)	51.155 (-19.95)	50.00 (-21.74)
Ignite	68.38 (-7.82)	59.765 (-19.32)	NA	71.505 (-3.52)	51.4 (-30.64)	51.465 (-30.56)	52.76 (-28.81)
OpenStack	60.23 (-5.36)	56.175 (-11.84)	58.62 (-7.95)	62.76 (-1.48)	52.73 (-17.14)	58.06 (-8.77)	50.0 (-21.43)
Qt	53.78 (-6.24)	51.80 (-9.89)	54.58(-5.02)	NA	50.32 (-12.47)	52.11 (-9.35)	50.00 (-13.02)

Table 8: Median AUC score at Privacy Level II with the relative change compared to the non-anonymized baseline in parentheses.

	RAD	**RS**	**K-DA**	**Gen**	**MORPH**	**LACE**	**LACE2**
Cassandra	83.49 (-3.62)	78.06 (-10.17)	NA	NA	27.57 (-68.27)	57.75 (-33.54)	14.665 (-83.12)
Flink	65.72 (-7.07)	66.08 (-6.98)	59.78 (-14.82)	66.4 (-6.24)	6.66 (-90.6)	17.14 (-75.8)	7.735 (-89.08)
Groovy	30.49 (-22.81)	32.77 (-16.79)	NA	36.935 (-6.96)	18.32 (-53.85)	9.87 (-75.14)	0.255 (-99.36)
Ignite	66.14 (-13.0)	59.61 (-21.28)	NA	57.79 (-23.82)	10.11 (-86.67)	3.755 (-95.05)	7.92 (-89.56)
OpenStack	42.98 (-15.58)	39.095 (-23.28)	30.3 (-40.48)	46.36 (-9.2)	37.69 (-25.97)	36.32 (-28.66)	0.025 (-99.95)
Qt	16.965 (-34.75)	13.75 (-47.68)	12.725 (-51.49)	NA	31.66 (20.47)	19.16 (-27.09)	0.035 (-99.87)

Table 9: Median Recall score at Privacy Level II with the relative change compared to the non-anonymized baseline in parentheses.

	Overall AUC	**Overall Recall**	**Overall G-Mean**	**Overall FPR**	**Overall Privacy**
Random Add Delete	15/3/0	14/3/1	17/1/0	3/3/12	0/5/13
Random Switch	12/6/0	14/3/1	17/1/0	2/4/12	0/1/17
k-DA	16/2/0	10/7/1	15/3/0	3/7/8	0/2/16
Gen	17/1/0	14/4/0	18/0/0	2/6/10	0/1/17

Table 10: Overall statistical test results for each GA technique on 6 datasets compared to all three NGA techniques with at least Privacy Level I (i.e., 18 cases). They are presented as win/tie/loss.

for a project, and a loss means that a best-performing GA configuration performed significantly worse than the MD50 technique for a project. The last row in Table 11 shows the results of the MD50 model when compared to a JIT model trained on non-anonymized data.

For the AUC metric, we find that all the GA-anonymized metrics perform equally well when compared to the MD50 model, being significantly better in 2/6 cases and comparable in 3/6 cases. For the G-Mean metric, all the GA techniques perform better in at least 4/6 cases, being insignificant in the other two cases (except Random Switch). The performance for AUC and G-Mean is similar to the performance achieved for models trained on non-anonymized data. For recall, we observe that the graph techniques perform better in just 1 or 2 cases, being worse in others. However, the FPR metric shows that even though MD50 has a better recall, its FPR rate is very high, with all the GA techniques performing at least as good in terms of FPR as MD50 for 5 cases, with k-DA and Gen even outperforming MD50 in 4 or more cases. This difference is due to the MD50 model marking 50% of the commits as buggy, which ends up marking a substantial number of commits as buggy while they are not actually buggy. For MD, the benefits of recall are counteracted by the increase in FPR.

Table 11 also shows that the performance of a model trained on graph anonymized metrics relative to MD50 is similar to that of models trained using non-anonymized data, i.e., the best performing GA anonymized model for a dataset and technique performs similar to the non-anonymized JIT defect prediction models, incurring no additional performance penalty (except FPR), while providing an IPR score of more than 65%. Overall, the models trained on graph anonymized data outperform the MD50 models in terms of FPR and G-Mean, while in more than 50% of the cases performing better or similar to MD50 models in terms of AUC.

	Overall AUC	Overall Recall	Overall G-Mean	Overall FPR
Random Add Delete	2 / 3 / 1	1 / 1 / 4	4 / 2 / 0	2 / 3 / 1
Random Switch	2 / 3 / 1	2 / 0 / 4	4 / 1 / 1	2 / 3 / 1
k-DA	2 / 3 / 1	0 / 2 / 4	4 / 2 / 0	4 / 1 / 1
Gen	2 / 3 / 1	1 / 1 / 4	4 / 2 / 0	5 / 0 / 1
Non-anonymized data	2 / 3 / 1	1 / 1 / 4	4 / 2 / 0	4 / 1 / 1

Table 11: Overall statistical test results for each GA technique compared to the Manual Down technique on the 6 datasets (i.e., 6 cases). They are presented as win/tie/loss.

Summary of Research Question 3

NGA techniques are able to provide significantly higher IPR scores when compared to GA techniques, with scores ranging between 89% to 98% for all datasets. However, this privacy (Privacy Level II) comes at the cost of a significant performance decrease, where NGA techniques, reduce AUC and Recall by a median of 19.82% and 66.64%, compared to 7.28% and 22.43% for GA techniques, respectively. Furthermore, NGA offers no ability to control the trade-offs between privacy and performance. When compared to MD50, GA techniques perform better or similar for at least 5/6 datasets for AUC, G-Mean and FPR.

6 Discussion

Implications for Researchers: Our work shows that graph anonymization techniques are able to provide increased privacy to JIT defect prediction data without impacting the trained models' performance. Unlike tabular techniques, the graph anonymization techniques are able to maintain performance, because, upon a change to the graph, they do not randomly alter the values of other features, but retain their logical context. Future research should explore the use of graph anonymization techniques to provide privacy for other types of software analytics data and models, for example, build status prediction [53], effort estimation [52], and more.

We would also recommend future work to combine multiple anonymization techniques together, for example, combining Random Add/Delete and Generalization techniques to observe how they change the performance of the models when applied one after the other on the data. Further exploration of whether techniques from the same graph anonymization category exhibit similar results would be useful in understanding how differently the techniques of the same category help data gain privacy.

Implications for Developers: Developers can use graph anonymization to provide privacy without sacrificing performance. For IPR scores greater than 65%, any of the graph anonymization techniques can be used without significant change in performance. In our case studies, we observed that applying the Random Add/Delete and Random Switch techniques to 20% of the nodes and for k-DA and Gen the configuration values 3 and 20, respectively, can be chosen to provide IPR scores 65% and greater.

For higher privacy requirements, we would recommend applying Random Add/Delete and Random Switch to 50% and 80% of the nodes, with Random Add/Delete having a lesser impact on performance. There is no consistent configuration value for k-DA and Gen that provides IPR scores of 80% or greater. We would recommend evaluating with higher values, k¿10 and Gen¿50. If the task is performance-sensitive, we suggest using Generalization, as our study shows that it provides significantly more performance as compared to the other techniques. Future work can explore if there are values where the behaviour of the Gen and k-DA techniques becomes consistent.

We also observe that privacy and performance have an almost inverse relation, since when privacy scores increase the performance stays constant for some time before decreasing, and vice versa. Higher privacy levels provided by RAD and RS are always accompanied by higher performance degradation. Trade-offs need to be made between both, where projects that have higher performance can be anonymized to a higher degree without the models becoming unusable. However, the same should not be done for projects with lower performance, where anonymization can make the models unusable.

Challenges: The main challenge of GA techniques is the additional setup required to create the underlying data graph and generate the metrics. Since drift might require retraining models on future versions of the underlying git data, future work should look into a way to incrementally update the graph, allowing more swift re-training or fine-tuning of models. This would ease the use and adoption of graph anonymization techniques for providing privacy. Finally, we believe that future research should explore the integration of the three families of anonymizing ML models, i.e., (1) graph anonymizing input data (our focus), (2) adopting specialized learning algorithms like federated learning [54], and (3) anonymizing the output of model predictions [18].

7 Threats to Validity

In this section, we highlight the threats to the validity of our paper are:

7.1 Construct Validity

Git-Graph specific results: It is possible that if the graph anonymization techniques are used on a different knowledge graph, the result of our study would be different. However, as graph anonymization techniques have been able to provide privacy to knowledge graphs of different domains, we believe that they would be able to provide privacy to knowledge graphs for different software analytics tasks made up of different entities (Reviewers, Comments, etc.). Verification of such applicability has been left for future work.

7.2 **Internal Validity**

Input threat: The process of graph anonymization and generating the JIT defect prediction metrics is time-consuming. Therefore, we do not repeat the graph anonymization at the same configuration level multiple times. However, we mitigate this threat by repeating the anonymization for multiple different configuration levels of the anonymization techniques. In doing so, we are able to establish that the results of anonymization are not an anomaly, as we see similar behaviour trends across different datasets.

Dependent metric selection: Similar to previous work for anonymization of defect prediction metrics by Peters et al. [35, 37, 36], we only used bug-proneness as the target-dependent variable for the model. The results might change if a different dependent variable is used, for example, effort-aware bug-proneness [22]. However, we hypothesize that the results will not dramatically change by graph anonymization of the metrics, based on findings like the fact that anonymization improves the FPR of the JIT model by about 3%. This indicates that graph anonymized metrics do not dramatically change the models' ability to avoid classifying non-buggy commits as buggy, thereby avoiding wasting developers' effort. We propose future work to evaluate the performance of the models with effort-aware (and other) dependent variables.

Hyperparameter tuning: It is possible that tuning the hyperparameters of the model would result in different performance metrics. Falessi et al., [15], in their work for understanding the effect of dormant defects, do not perform hyperparameter turning. This is because their aim is not to produce the most optimal models, but instead to highlight the fact that dormant defects decrease the recall of defect prediction. Similarly, the aim of our study is not to produce the most optimal models but rather to assess the change that comes with privatizing the datasets. We measure the change in model performance anonymization brings compared to the non-anonymized baseline. A model that has been tuned can be expected to show a better performance, albeit with a similar trend of change in performance metrics as our current results.

Testing/training split: We use an 80/20 chronologically sorted training/testing split to train the models.[47]. We then use out-of-sample bootstrapping to ensure that the models have less bias and variance [48]. It would also ensure that the model's performance is good or bad based on the random draw of elements but rather consistent overall.

7.3 External Validity

Privacy metrics: We only use IPR for measuring privacy results, which has been used in multiple previous works [36, 37, 35, 26]. However, it is possible that a different privacy metric may produce different results. We leave the evaluation of privacy with different metrics to future work.

Project-specific: It is possible that our results might change if different projects were studied or projects in languages other than Java, C++ and Python. However, the graph anonymization techniques and the git-graph generated are language-independent. They only operate on the knowledge graph generated from the repositories, regardless of the underlying programming language used. We used projects of different sizes and use cases to ensure wider applicability. Therefore, we believe that the findings should be valid even for different projects.

Random Forest Classifier: It is possible that the results might change if a different classifier is used. However, Random Forest is commonly used for JIT defect prediction models and has been found to perform at least as good as deep learning models [58].

8 Conclusion

Our case study on 6 large, long-lived projects with 4 graph anonymization techniques shows that they are an effective way to provide privacy to the data while maintaining performance. All 6 datasets under study were able to gain privacy scores greater than 65% with no significant loss in performance and greater than 80% with slight changes in performance. Out of the four graph-anonymization techniques, Generalization was the best performing in terms of preserving AUC and Recall, followed by Random Add/Delete. All techniques were able to provide Privacy Level II at lower configuration levels without significant anonymization effort.

When compared to state-of-the-art anonymization techniques like MORPH, LACE and LACE2, the graph-anonymization techniques were able to provide privacy scores greater than 80% with significantly lesser change in performance metrics. Compared to MD50, all the GA techniques perform better in at least 4/6 cases for G-Mean and FPR, (except RS and RAD, which perform better in 2/6 cases for FPR) while performing similar or better for AUC in 5/6 cases. This shows that graph-anonymization techniques are an effective way to provide privacy to data.

As of now, we have been able to demonstrate that graph-anonymization techniques are effective in preserving the data's within-project predictive capabilities; however, in future work, we would like to test this in a cross-project defect prediction setting where we can verify that even after graph-anonymization, the data is able to retain its predictive power in a cross-project setting. We would also like to extend our work by applying graph-anonymization techniques to other software analytics tasks than JIT prediction.

9 Conflict of Interests

All authors declare that they have no conflicts of interest.

10 Data Availability Statement

The replication package for this project which contains the code and data used can be found here [7].

11 Acknowledgements

We want to thank Filipe Côgo for the initial discussions regarding the topic of this paper.

References

[1] en. Page Version ID: 1131061188. Jan. 2023. URL: https://en.wikipedia.org/w/index.php?title=Graph_(abstract_data_type)&oldid=1131061188.
[2] URL: https://review.opendev.org/q/status:open+-is:wip.
[3] URL: https://codereview.qt-project.org/.
[4] URL: https://pydriller.readthedocs.io/en/latest/.
[5] URL: https://scikit-learn/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.
[6] URL: http://lace.readthedocs.io/en/latest/readme.html.
[7] URL: https://www.dropbox.com/scl/fo/qztc7clj321gnw8qtgl69/h?rlkey=iu3hn2jivf3ot8o43kfnje7zz&dl=0.
[8] Justin Brickell and Vitaly Shmatikov. "The cost of privacy: destruction of data-mining utility in anonymized data publishing". In: *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2008, pp. 70–78.
[9] Saikat Chakraborty et al. "Deep Learning Based Vulnerability Detection: Are We There Yet?" In: *IEEE Transactions on Software Engineering* 48.9 (2022), pp. 3280–3296.
[10] Jiale Chen et al. "Beyond Model-Level Membership Privacy Leakage: an Adversarial Approach in Federated Learning". In: *2020 29th International Conference on Computer Communications and Networks (ICCCN)*. 2020, pp. 1–9.
[11] James Cheng, Ada Wai-chee Fu, and Jia Liu. "K-isomorphism: privacy preserving network publication against structural attacks". In: *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. 2010, pp. 459–470.

[12] S.R. Chidamber and C.F. Kemerer. “A metrics suite for object oriented design”. In: *IEEE Transactions on Software Engineering* 20.6 (1994), pp. 476–493.

[13] Aritra Dasgupta, Min Chen, and Robert Kosara. “Measuring Privacy and Utility in Privacy-Preserving Visualization”. In: *Computer Graphics Forum*. Vol. 32. 8. Wiley Online Library. 2013, pp. 35–47.

[14] Khaled El Emam and Fida Kamal Dankar. “Protecting privacy using k-anonymity”. In: *Journal of the American Medical Informatics Association* 15.5 (2008), pp. 627–637.

[15] Davide Falessi, Aalok Ahluwalia, and Massimiliano DI Penta. “The Impact of Dormant Defects on Defect Prediction: A Study of 19 Apache Projects”. In: *ACM Trans. Softw. Eng. Methodol.* 31.1 (Sept. 2021). ISSN: 1049-331X. URL: `https://doi.org/10.1145/3467895`.

[16] Yi Fan et al. “The Utility Challenge of Privacy-Preserving Data-Sharing in Cross-Company Defect Prediction: An Empirical Study of the CLIFF-MORPH Algorithm”. In: *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 2017, pp. 80–90.

[17] Tomas Feder, Shubha U. Nabar, and Evimaria Terzi. *Anonymizing Graphs*. ADS Bibcode: 2008arXiv0810.5578F type: article. Oct. 2008. URL: `https://ui.adsabs.harvard.edu/abs/2008arXiv0810.5578F`.

[18] Moritz Hardt and Eric Price. “The noisy power method: A meta algorithm with applications”. In: *Advances in neural information processing systems* 27 (2014).

[19] Thong Hoang et al. “CC2Vec: Distributed representations of code changes”. In: *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. 2020, pp. 518–529.

[20] Aidan Hogan et al. “Knowledge Graphs”. In: *ACM Comput. Surv.* 54.4 (July 2021). ISSN: 0360-0300. URL: `https://doi.org/10.1145/3447772`.

[21] Qiao Huang, Xin Xia, and David Lo. “Revisiting supervised and unsupervised models for effort-aware just-in-time defect prediction”. In: *Empirical Software Engineering* 24 (2019), pp. 2823–2862.

[22] Qiao Huang, Xin Xia, and David Lo. “Revisiting supervised and unsupervised models for effort-aware just-in-time defect prediction”. In: *Empirical Software Engineering* 24 (2019), pp. 2823–2862.

[23] Shouling Ji, Prateek Mittal, and Raheem Beyah. “Graph Data Anonymization, De-Anonymization Attacks, and De-Anonymizability Quantification: A Survey”. In: *IEEE Communications Surveys Tutorials* 19.2 (2017), pp. 1305–1326.

[24] Yasutaka Kamei et al. “A large-scale empirical study of just-in-time quality assurance”. In: *IEEE Transactions on Software Engineering* 39.6 (2012), pp. 757–773.

[25] Hossein Keshavarz and Meiyappan Nagappan. “ApacheJIT: A Large Dataset for Just-in-Time Defect Prediction”. In: *Proceedings of the 19th International Conference on Mining Software Repositories*. MSR ’22. Pittsburgh, Pennsylvania: Association for Computing Machinery, 2022,

pp. 191–195. ISBN: 9781450393034. URL: `https://doi.org/10.1145/3524842.3527996`.

[26] Zhiqiang Li et al. “On the Multiple Sources and Privacy Preservation Issues for Heterogeneous Defect Prediction”. In: *IEEE Transactions on Software Engineering* 45.4 (2019), pp. 391–411.

[27] Kun Liu and Evimaria Terzi. “Towards identity anonymization on graphs”. In: *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. 2008, pp. 93–106.

[28] Shane McIntosh and Yasutaka Kamei. “Are Fix-Inducing Changes a Moving Target? A Longitudinal Case Study of Just-In-Time Defect Prediction”. In: *IEEE Transactions on Software Engineering* 44.5 (2018), pp. 412–428.

[29] Ricardo Mendes and João P. Vilela. “Privacy-Preserving Data Mining: Methods, Metrics, and Applications”. In: *IEEE Access* 5 (2017), pp. 10562–10582.

[30] Prateek Mittal, Charalampos Papamanthou, and Dawn Song. “Preserving link privacy in social network based systems”. In: *arXiv preprint arXiv:1208.6189* (2012).

[31] “Model Inversion Attacks that Exploit Confidence Information and Basic Countermeasures”. In: CCS ’15 (2015), pp. 1322–1333. URL: `https://doi.org/10.1145/2810103.2813677`.

[32] R. Mortazavi and S. H. Erfani. “GRAM: An efficient (k, l) graph anonymization method”. en. In: *Expert Systems with Applications* 153 (Sept. 2020), p. 113454. ISSN: 0957-4174.

[33] Arvind Narayanan and Vitaly Shmatikov. “De-anonymizing social networks”. In: *2009 30th IEEE symposium on security and privacy*. IEEE. 2009, pp. 173–187.

[34] Milad Nasr, Reza Shokri, and Amir Houmansadr. “Comprehensive Privacy Analysis of Deep Learning: Passive and Active White-box Inference Attacks against Centralized and Federated Learning”. In: *2019 IEEE Symposium on Security and Privacy (SP)*. 2019, pp. 739–753.

[35] Fayola Peters and Tim Menzies. “Privacy and utility for defect prediction: Experiments with MORPH”. In: *2012 34th International Conference on Software Engineering (ICSE)*. 2012, pp. 189–199.

[36] Fayola Peters, Tim Menzies, and Lucas Layman. “LACE2: Better Privacy-Preserving Data Sharing for Cross Project Defect Prediction”. In: *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*. Vol. 1. 2015, pp. 801–811.

[37] Fayola Peters et al. “Balancing Privacy and Utility in Cross-Company Defect Prediction”. In: *IEEE Transactions on Software Engineering* 39.8 (2013), pp. 1054–1068.

[38] Chanathip Pornprasit and Chakkrit Kla Tantithamthavorn. “JITLine: A Simpler, Better, Faster, Finer-grained Just-In-Time Defect Prediction”. English. In: IEEE Computer Society, May 2021, pp. 369–379. ISBN: 978-1-72818-710-5. URL: `https://www.computer.org/csdl/proceedings-article/msr/2021/871000a369/1tB7jJnFTi0`.

[39] Anastasia Pustozerova and Rudolf Mayer. "Information leaks in federated learning". In: *Proceedings of the Network and Distributed System Security Symposium*. Vol. 10. 2020.

[40] Gema Rodríguez-Pérez, Meiyappan Nagappan, and Gregorio Robles. "Watch Out for Extrinsic Bugs! A Case Study of Their Impact in Just-In-Time Bug Prediction Models on the OpenStack Project". In: *IEEE Transactions on Software Engineering* 48.4 (2022), pp. 1400–1416.

[41] Mohammad Al-Rubaie and J. Morris Chang. "Privacy-Preserving Machine Learning: Threats and Solutions". In: *IEEE Security Privacy* 17.2 (2019), pp. 49–58.

[42] Alessandra Sala et al. "Sharing graphs using differentially private graph models". In: *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*. 2011, pp. 81–98.

[43] Reza Shokri et al. "Membership Inference Attacks Against Machine Learning Models". In: *2017 IEEE Symposium on Security and Privacy (SP)*. 2017, pp. 3–18.

[44] Reza Shokri et al. "Membership Inference Attacks Against Machine Learning Models". In: *2017 IEEE Symposium on Security and Privacy (SP)*. 2017, pp. 3–18.

[45] Mudhakar Srivatsa and Mike Hicks. "Deanonymizing mobility traces: Using social network as a side-channel". In: *Proceedings of the 2012 ACM conference on Computer and communications security*. 2012, pp. 628–637.

[46] Sadia Tabassum, Leandro L. Minku, and Danyi Feng. "Cross-Project Online Just-In-Time Software Defect Prediction". In: *IEEE Transactions on Software Engineering* 49.1 (2023), pp. 268–287.

[47] Ming Tan et al. "Online Defect Prediction for Imbalanced Data". In: *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*. Vol. 2. 2015, pp. 99–108.

[48] Chakkrit Tantithamthavorn et al. "An empirical comparison of model validation techniques for defect prediction models". In: *IEEE Transactions on Software Engineering* 43.1 (2016), pp. 1–18.

[49] Brian Thompson and Danfeng Yao. "The union-split algorithm and cluster-based anonymization of social networks". In: *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security*. 2009, pp. 218–227.

[50] Jacek undefinedliwerski, Thomas Zimmermann, and Andreas Zeller. "When Do Changes Induce Fixes?" In: *Proceedings of the 2005 International Workshop on Mining Software Repositories*. MSR '05. St. Louis, Missouri: Association for Computing Machinery, 2005, pp. 1–5. ISBN: 1595931236. URL: `https://doi.org/10.1145/1083142.1083147`.

[51] Isabel Wagner and David Eckhoff. "Technical Privacy Metrics: A Systematic Survey". In: *ACM Comput. Surv.* 51.3 (June 2018). ISSN: 0360-0300. URL: `https://doi.org/10.1145/3168389`.

[52] Peter A Whigham, Caitlin A Owen, and Stephen G Macdonell. “A baseline model for software effort estimation”. In: *ACM Transactions on Software Engineering and Methodology (TOSEM)* 24.3 (2015), pp. 1–11.

[53] Timo Wolf et al. “Predicting build failures using social network analysis on developer communication”. In: *2009 IEEE 31st International Conference on Software Engineering*. 2009, pp. 1–11.

[54] Hiroki Yamamoto et al. “Towards Privacy Preserving Cross Project Defect Prediction with Federated Learning”. In: *2023 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 2023, pp. 485–496.

[55] Yibiao Yang et al. “Effort-Aware Just-in-Time Defect Prediction: Simple Unsupervised Models Could Be Better than Supervised Models”. In: *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. FSE 2016. Seattle, WA, USA: Association for Computing Machinery, 2016, pp. 157–168. ISBN: 9781450342186. URL: `https://doi.org/10.1145/2950290.2950353`.

[56] Samuel Yeom et al. “Privacy Risk in Machine Learning: Analyzing the Connection to Overfitting”. In: *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*. 2018, pp. 268–282.

[57] Xiaowei Ying and Xintao Wu. “Randomizing social networks: a spectrum preserving approach”. In: *proceedings of the 2008 SIAM International Conference on Data Mining*. SIAM. 2008, pp. 739–750.

[58] Zhengran Zeng et al. “Deep Just-in-Time Defect Prediction: How Far Are We?” In: *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*. ISSTA 2021. Virtual, Denmark: Association for Computing Machinery, 2021, pp. 427–438. ISBN: 9781450384599. URL: `https://doi.org/10.1145/3460319.3464819`.

[59] Bin Zhou and Jian Pei. “Preserving privacy in social networks against neighborhood attacks”. In: *2008 IEEE 24th International Conference on Data Engineering*. IEEE. 2008, pp. 506–515.

[60] Lei Zou, Lei Chen, and M Tamer Özsu. “K-automorphism: A general framework for privacy preserving network publication”. In: *Proceedings of the VLDB Endowment* 2.1 (2009), pp. 946–957.