Detecting Outdated Screenshot from GUI Document

YE TANG, Shanghai Jiao Tong University, China
AOYANG YAN, Shanghai Jiao Tong University, China
HUI LIU, Beijing Institute of Technology, China
NA MENG, Virginia Tech, USA
HAO ZHONG*, Shanghai Jiao Tong University, China

In software development, many documents (*e.g.*, tutorials for tools and mobile application websites) contain screenshots of graphical user interfaces (GUIs) to illustrate functionalities. Although screenshots are critical in such documents, screenshots can become outdated, especially if document developers forget to update them. Outdated screenshots can mislead users and diminish the credibility of documentation. Identifying screenshots manually is tedious and error-prone, especially when documents are many. However, no existing tools are proposed to detect outdated screenshots in GUI documents.

To mitigate manual efforts, we propose DOSUD, a novel approach for detecting outdated screenshots. It is challenging to identify outdated screenshots since the differences are subtle and only specific areas are useful to identify such screenshots. To address the challenges, DOSUD automatically extracts and labels screenshots and trains a classification model to identify outdated screenshots. As the first exploration, we focus on Android applications and the most popular IDE, VS Code. We evaluated DOSUD on a benchmark comprising 10 popular applications, achieving high F1-scores. When applied in the wild, DOSUD identified 20 outdated screenshots across 50 Android application websites and 17 outdated screenshots in VS Code documentation. VS Code developers have confirmed and fixed all our bug reports.

ACM Reference Format:
Ye Tang, Aoyang Yan, Hui Liu, Na Meng, and Hao Zhong. 2024. Detecting Outdated Screenshot from GUI Document. *ACM Trans. Softw. Eng. Methodol.* 1, 1 (July 2024), 25 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Software documents often incorporate screenshots of graphical user interfaces (GUIs) to illustrate software functionalities and the steps of human-computer interactions. For instance, screenshots are prevalent in bug reports [38] and tutorials [52], where they convey information more lucidly and vividly than textual descriptions alone. As a result, many applications provide carefully written documentation with screenshots. For instance, Microsoft alone maintains more than 700 such repositories dedicated to such documentation [3]. Each repository has many documents with screenshots. For instance, VS Code is an open-source IDE implemented by Microsoft. According to the Stack Overflow 2023 developer survey [12], VS Code is the most popular IDE. Among the 590

*Manuscript received July, 2024.
(Corresponding author: Hao Zhong.)

Authors' addresses: Ye Tang, tangye_22@sjtu.edu.cn, Shanghai Jiao Tong University, China; Aoyang Yan, xiaoyan9894@sjtu.edu.cn, Shanghai Jiao Tong University, China; Hui Liu, liuhui08@bit.edu.cn, Beijing Institute of Technology, China; Na Meng, nm8247@cs.vt.edu, Virginia Tech, USA; Hao Zhong, zhonghao@sjtu.edu.cn, Shanghai Jiao Tong University, China.

1049-331X/2024/7-ART $15.00
https://doi.org/XXXXXXX.XXXXXXX

documents of VS Code, 455 documents have screenshots. Besides Microsoft, other companies also recognize the importance of screenshots. For instance, a research lead for UX research at Baymard named Edward Scott wrote an article about screenshots. In this article, he criticized that 35% of SaaS and digital subscription sites fail to provide sufficient visual information, *e.g.*, screenshots.

Although screenshots are critical for GUI documents, they are misleading and harmful if they are outdated. For instance, as an experienced developer, Sarah Moir wrote an article to discuss the benefits and pitfalls of adding screenshots to documentation [64]. On one side, she lists three benefits of screenshots such as making documents easier to read, providing visual references, and supplementing complicated task steps. On the other side, she lists five pitfalls of screenshots. As the first pitfall, she criticizes that outdated screenshots can cause readers to lose trust in documentation. YC is a well-known incubator, and it invested over 5,000 companies (*e.g.*, AirBnb and Dropbox). When this article is posted on YC, it raises a hot discussion [26]. A reader endorses the criticism: "outdated screenshot is one of the fastest ways to lose customer trust". Another reader also agrees with the criticism: "I lose a lot more trust when I've spent 15 minutes trying to find the damn option the documentation insists should be there, only to find it doesn't exist any more". To handle the problem, developers have fixed many outdated screenshots. For instance, by searching the issue tracker of VS Code with the two keywords, "update screenshot" or "outdated screenshot", we find more than 210 bug reports. In these bug reports, programmers complain that documents with outdated screenshots are confusing. Still, we can underestimate the relevance of outdated screenshots, since programmers can report outdated screenshots without mentioning our keywords. For instance, Azure is a cloud solution provided by Microsoft. An Azure bug report [2] complains about an outdated image, and this image is a screenshot. As GUI documents are many, it is tedious to manually identify outdated screenshots.

Although developers have manually fixed many outdated screenshots, to the best of our knowledge, no prior approach has been proposed to detect such bugs, due to two challenges. First, the differences between the latest and outdated screenshots are often subtle. Although outdated screenshots hinder the understanding of documents, it is difficult for document developers to identify outdated screenshots if they do not execute applications step by step as described. Second, only specific areas of screenshots are meaningful for identifying outdated screenshots, but many other differences can be caused by underlying data. If we encode screenshots to pixels, a tool must know which areas of pixels indicate outdated screenshots.

To meet the timely need, in this paper, we propose a novel approach named DOSUD (Detecting Outdated Screenshots in GUI Documents). It is the first approach to detect outdated screenshots. To handle the first challenge, DOSUD integrates multiple discipline techniques including image processing and GUI testing. To resolve the second challenge, DOSUD captures screenshots from multiple versions of applications and trains a classification model upon captured screenshots. Our built model is a random forest, and it is the combination of many `if`-statements. If an area is dynamic generated, it seldom associates with outdated screenshots. As irrelevant areas punish our training process, our built model unlikely has positive `if`-statements to check the pixels of such areas when determining outdate screenshots. This paper makes the following contributions:

- **A new research direction of detecting outdated screenshots in documents.** Despite the efforts of manually fixing outdated screenshots, to the best of our knowledge, no approach can detect outdated screenshots from GUI documents. For the first time, we introduce this problem to the research community. As screenshots are commonly used in various types of software engineering documents (*e.g.*, bug reports, and development emails), the research on this new direction can motivate various useful tools.

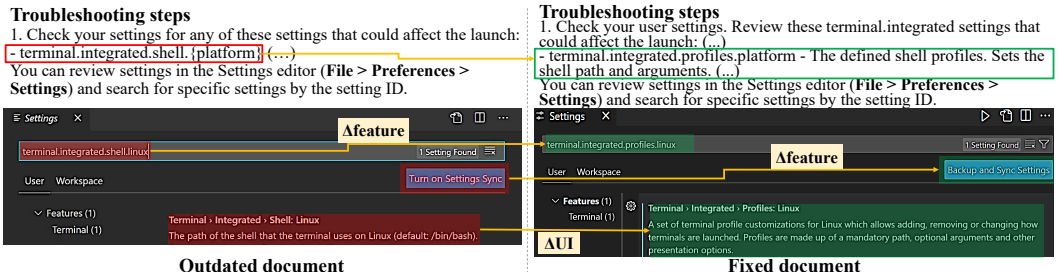


Fig. 1. Our detected bug in VS Code Documentation.

- **A novel approach.** To illustrate our new direction, in this paper, we propose a novel approach, named DOSUD. As the first exploration, we focus on two typical GUI documents such as the website of Android applications and the documents of VS code. It can detect outdated screenshots from the documents of Android applications and VS Code. Specifically, it employs GUI testing to dump screenshots of both the latest and previous versions and then extracts training data with image processing techniques. From the training data of each application, it trains a classifier that can predict whether a screenshot is outdated or not.
- **Previously undetected outdated screenshots in the wild.** To evaluate its effectiveness in the wild, with DOSUD, we applied DOSUD to the latest websites of 50 applications on Google Play and VS Code. We discovered 20 instances of outdated screenshots from the website of 50 Android applications. Moreover, we detected 17 outdated screenshots within VS Code documentation that were not previously reported by anyone. Remarkably, 17 of these instances have been fixed by the developers, following the submission of our issue reports. This result shows the importance and the relevance of our research problem.
- **Positive results on our benchmark.** We construct a benchmark of captured screenshots. DOSUD achieves an F1-score ranging from 87% to 100% on this benchmark.

In total, we evaluated DOSUD on 59 Android applications (9 in the benchmark and 50 in the wild). Additionally, we evaluated DOSUD on a desktop application, VS Code, which was examined both in the benchmark and in the wild. The subjects are sufficient to ensure the reliability of DOSUD.

Besides Android applications and desktop applications, our approach can be extended to detect outdated screenshots from the documents of other applications (such as Azure). Although we need to re-implement our GUI testing tool, it should be feasible given the extensive research on GUI testing [33] and the availability of alternative tools. Moreover, other types of software engineering documents may have different formats from web pages (*e.g.*, pdf files and gif files). For these documents, we need to re-implement our extractors as well. Nevertheless, our approach can be generalized to other types of applications, as long as their screenshots can be extracted. Detecting outdated screenshots in applications is a significant and pertinent issue that affects various kinds of documentation. This paper represents the first exploration of a novel research direction, which entails some unresolved challenges. We discuss several challenges that arise from more complicated applications in Section 8.

2 MOTIVATING EXAMPLE

With extensions, VS Code can support the development of many programming languages. To learn VS Code, Microsoft has a project to maintain its documentation [14]. The documentation has more than 500 documents. Programmers often read the documentation and report their found bugs. Till now, programmers have reported 2,589 bugs, and 2,511 bugs have been fixed.

Although VS code developers fixed many bugs, as shown in Figure 1, DOSUD detects a new bug from a document. The description of this document explains how to search for a specific setting by its ID. When explaining the process, this document introduces an ID, `terminal.integrated.shell.linux`, and provides a corresponding screenshot. If programmers use the latest version of VS Code and follow the document, they will not find the said setting, since this ID is no longer valid. As shown in the right side of Figure 1, if programmers need to search this setting in the latest VS Code, they must use a new ID, `terminal.integrated.profiles.linux` (highlighted in green).

DOSUD detects that this screenshot is outdated. We reported this bug [16], and it was confirmed in one day. A developer fixed this bug in three weeks and commented that "*Thanks for raising this.*". The right side of Figure 1 shows the fixed document. The description introduces the latest ID, and the screenshot shows the latest version. We highlight the new ID in green and show the modification with a yellow arrow. Besides IDs, the obsolete screenshot has more confusing elements. For example, the `Turn on Settings Sync button` (highlighted in red) has been replaced by `Backup and Sync Settings` (highlighted in green) button. The fixed document resolves the confusion caused by the outdated screenshot.

Without DOSUD, developers have to read many documents manually to find this bug. When developers check whether a screenshot is outdated or not, they have to locate the mentioned GUI and manually compare the differences. The process is time-consuming and error-prone. DOSUD can detect this bug, since it extracts screenshots from previous versions of VS Code, and learns a classification model from captured screenshots. The trained model can predict whether screenshots are outdated or not.

Direct comparisons between screenshots are inadequate for dealing with complicated cases. In Section 6.3.2, as a form of direct comparison, the 1-nearest neighbor classifier fails to classify the latest screenshot in Figure 1, because the closest match resembles the outdated screenshot in Figure 1. In contrast, we train an RF classifier for VS Code. We encode screenshots to pixels and train our model on labeled pixels. The trained model is a random forest of decision trees. Each tree can be considered as a set of `if` statements. Taking critical pixels as input, these `if` statements determine whether a screenshot is outdated according to specific patterns in its pixels. In Figure 1, we highlight the differences between the outdated and the latest screenshots. The subtle changes cause many different pixels. Our classifier is sensitive to the different pixels and can predict the outdated screenshot accordingly. We next introduce how DOSUD works.

3 PRELIMINARY STUDY

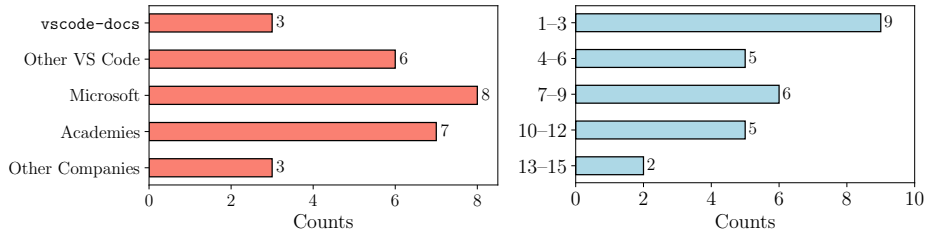
In this section, we conduct a study to explore two research questions:

(RQ1) Who cares about outdated screenshot (Section 3.1)?
(RQ2) Can outdated screenshots be identified by the last modification time (Section 3.2)?

3.1 RQ1. Stockholder

3.1.1 Protocol. We searched the issue tracker of VS Code with the two keywords, "update screenshot" and "outdated screenshot". In total, we obtained 210 bug reports. From them, we randomly sample 20% of the bug reports and select 44 bugs. These outdated-screenshot bugs are reported by 27 users and are fixed by 18 programmers. All the 44 bugs confuse readers since their reporters are real users and programmers. When repairing 24 out of the 44 bugs, VS Code programmers modify both screenshots and their descriptions. The 24 bugs are closely tied to the core content of textural descriptions. These bugs can significantly affect the understanding of readers.

The GitHub profiles record the organizations and programming experience of users. If no organization is provided, we examine their personal websites and emails on GitHub profiles, *e.g.*,



(a) The organizations. **(b)** The coding experience (Years).

Fig. 2. The background of users who report outdated screenshots.

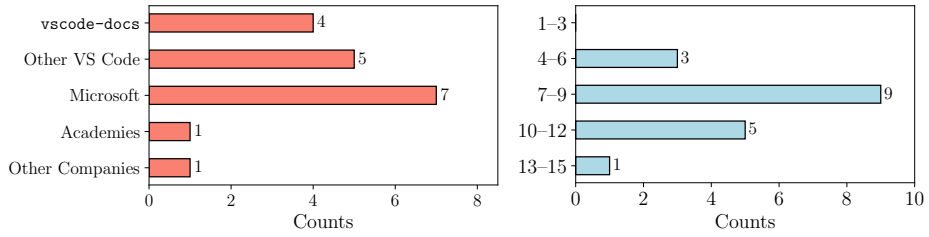
LinkedIn, blogs, and Twitter. If no such links are available, we use Google to search for a personal website associated with the same name and verify whether it contains information consistent with the GitHub profile. We released our inspected bugs on our project website. Other researchers can check our results. For each user, we calculate the duration from the date of the first activity on GitHub to the date of reporting or fixing the outdated screenshot. We use the durations to measure the programming experience of users. We classify users and programmers by their organizations and programming experience. For users, the results are useful for understanding the outreach impact of outdated screenshots. For programmers, the results are useful for understanding the required expertise of checking and repairing reported outdated screenshots.

3.1.2 Results. Figure 2 shows the background of users. In particular, Figure 2a shows that users come from VS Code, Microsoft, academies, and other companies. Here, `vscode-docs` is a project for maintaining the documentation of VS Code. Among the 27 users, only 3 bug reporters come from `vscode-docs`. For instance, Gre* Van* Lie*, a senior content writer of `vscode-docs`, submitted two bug reports about outdated screenshots. Although they are responsible for maintaining the documents of VS Code, only 8 out of the 44 outdated screenshots are reported by the writers of `vscode-docs`. According to the result, most outdated screenshots are not identified by proofreading of documentation writers but by programmers in real development. For instance, Ben* Pas* is a programmer from Microsoft. He is a member of the VS Code project since it was called Monaco. Among our sampled bugs, Ben* Pas* alone reports 7 outdated screenshots. For instance, in one of his bug reports [28], he complains that "*I noticed that (for macOS at least) our screenshot of VS Code is really outdated. Mainly we do not show our nice custom window title bar.*"

Many programmers of VS Code projects come from Microsoft, since VS Code was initially developed and later donated by Microsoft [30]. In Figure 2a, Microsoft denotes bugs reported by Microsoft programmers who are not members of VS code projects. For instance, Luc* Abu* is a Microsoft senior manager, and he leads several Python projects. In our samples bug reports, he reported three outdated screenshots.

Besides Microsoft programmers, outdated screenshots affect users from other industrial and academic organizations. For instance, Odi* Dah* is from Linköping University. In a bug report [19], he complains that "*On the 'Add a JAR' section of the Java projects documentation is using an outdated version of the extension where the Java Projects view is still called 'Java Dependencies'. This could be confusing for new users*". As another example, Ben* Rog* is a programmer from Employment and Social Development, Canada. His bug report [29] complains that "*Observe that the new setting in v1.69 is emmet.useInlineCompletions, however the documentation on the Release Notes indicates the incorrect setting of emmet.inlineCompletions.*"

Figure 2b shows the distribution of programming experience. The result shows that outdated screenshots mainly affect users whose programming experience is less than three years. For instance, Pol* Pra*, a user from Microsoft, has a year of experience. As described in her bug report [17], she



(a) The organizations. **(b)** The coding experience (year).

Fig. 3. The background of programmers who repair outdated screenshots.

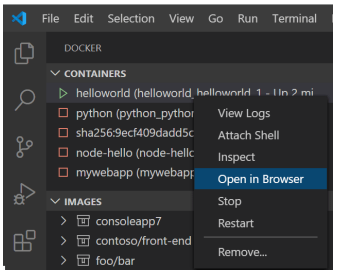


Fig. 4. A false alarm of threshold.

was confused by the missing button: "*for the 'dismiss this update' button in the 'version' info bar, hence they will get confused and face difficulties in accessing the 'dismiss' button*". Meanwhile, even experienced users can be confused. For instance, Pin* is a programmer of the VS Code language model server for Vue.js. He has 12 years of programming experience, and his bug report [31] complains that "*The security tab is changed to the dropdown. Might confuse people.*" The above observations lead to a finding:

> **Finding 1:** Besides Microsoft, outdated screenshots affect users from other organizations, and 33.3% of such users have fewer than 3 years of programming experience.

Figure 3a shows the organizations of programmers. Although only 4 out of the 18 programmers come from `vscode-docs`, they fixed 26 out of 44 outdated screenshots. For instance, Gre* Van* Lie* is a senior content writer of `vscode-docs`. He fixed 17 outdated screenshots. The programmers from other VS code projects and other Microsoft programmers fixed 15 bugs. For instance, Joa* Mor* is a principal software engineer on the VS Code editor. He fixed one outdated screenshot. Jos* Par* is a senoir content writer for Windows. He fixed three outdated screenshots. Only 3 outdated screenshots are fixed by outsiders. For instance, Rac* Mac* is a student from Harvey Mudd College. She fixed an outdated screenshot after she reported the bug [20]. Figure 3b shows the programming experience of programmers. The result shows that fixing outdated screenshots requires many years of programming experience even if they are outsiders. For instance, as introduced before, Rac* Mac* fixed an outdated screenshot. She has five years of programming experience. The above observations lead to a finding:

> **Finding 2:** Outdated screenshots can confuse experienced programmers since 25.9% of reporters have more than 10 years of programming experience, and 83.3% of programmers who fixed outdated screenshots have more than seven years of programming experience.

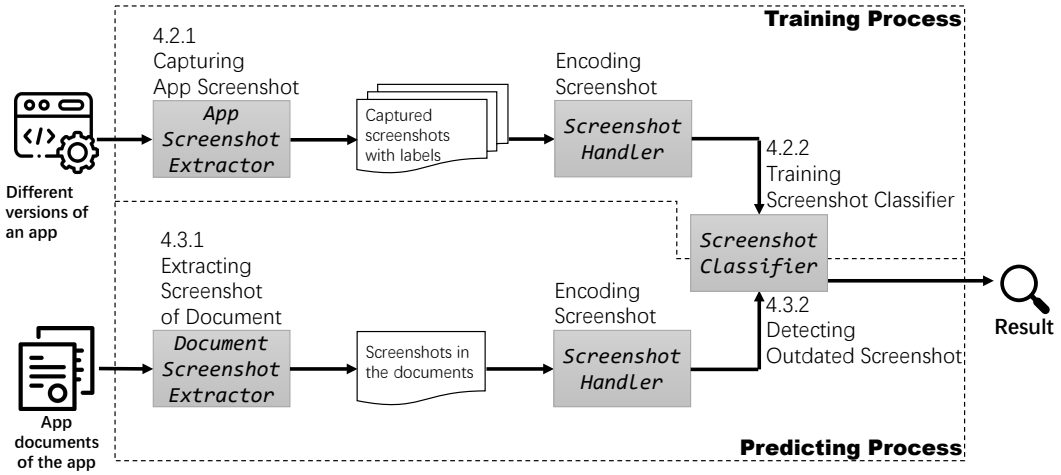


Fig. 5. The overview of DOSUD.

3.2 RQ2: Feasibility of Simple Threshold

3.2.1 Protocols. In this research question, we explore whether it is feasible to accurately identify outdated screenshots with a simple threshold. First, we extract all commits that repair outdated screenshots in RQ1. For each commit, we calculate the time interval from this modification to the previous modification. We find that the median time interval is 297 days. We select this median as the threshold for identifying outdated screenshots. We use this threshold to determine whether the latest screenshots are outdated. Like other bugs, only a small portion of screenshots could be outdated. To build the confidence on this statement, we randomly sample 10% of identified bugs and manually check whether they are indeed outdated.

3.2.2 Results. When we use 297 days as the threshold, 70.3% of screenshots are determined as outdated. As the VS Code documentation is under active maintenance, it should not have so many outdated screenshots. As a result, we suspect that the threshold leads to many false alarms. After our manual inspection, we found that only 8 out of our sampled 154 screenshots are outdated. For instance, the screenshot shown in Figure 4 is a false alarm. Although it has not been updated in 1,281 days, it remains accurate and up to date. The observations lead to the following finding:

> **Finding 3:** It is infeasible to use a simple time interval threshold to accurately determine whether screenshots are outdated.

In summary, outdated screenshots are significant and relevant since they confuse programmers both from Microsoft and other organizations. Although our sample size is limited, our early findings show that both novice and experienced programmers can encounter outdated screenshots. Although novice programmers can identify outdated screenshots, a simple threshold is insufficient to detect outdated screenshots. After they are identified, most outdated screenshots are fixed by professionals with seven to nine years of programming experience. We next introduce how DOSUD works.

4 APPROACH

Figure 5 shows the overview of DOSUD. It consists of a training process and a predicting process. In the training process, DOSUD captures the screenshots of both the latest version and the past

Algorithm 1: Screenshot Extractor and Handler

```
Input: All versions of an app, APPs
Output: trainData
trainData ← [];
foreach app ∈ APPs do
    screenshots ← ScreenshotExtractor(app);
    foreach screenshot ∈ screenshots do
        vector ← ScreenshotHandler(screenshot);
        if app.version is brand-new then
            vector.label ← false;
        else
            vector.label ← true;
        if vector ∈ latest version ∧ vector ∈ old version then
            vector.label ← false;
        trainData.append(vector);
return trainData;
```

versions as inputs, and trains a classification model. In the predicting process, it uses the trained model to detect outdated screenshots from the app websites.

4.1 Problem Definition

Let $v_1, v_2, \cdots, v_n$ denote all versions of an application, where v_n is the latest one, and let $S^i = \{s_1^i, s_2^i, \cdots\}$ represent the set of all screenshots in version v_i. In the latest documents of v_n, a screenshot s is considered outdated, if it satisfies the following condition:

$$s \in \bigcup_{i=1}^{n-1} \left(S^i - S^n\right). \tag{1}$$

According to our definition, if the latest version of the documentation contains a screenshot but the latest implementation does not have such a screenshot, we consider that it is outdated. In our target problem, we consider only outdated screenshots in documents.

Although our definition is precise, when detecting outdated screenshots in the wild in Section 5, we find that outdated screenshots intertwine with descriptions. For instance, even if screenshots are fresh, descriptions can be outdated. In addition, screenshots and descriptions can be outdated at the same time. Zhong and Su [81] detect outdated descriptions in documents, *e.g.*, outdated code references and samples. As outdated screenshots intertwine with descriptions, it could be interesting to integrate our approach with Zhong and Su [81] to detect more types of bugs. Meanwhile, outdated screenshots may not hinder the understanding of documents if updated GUI elements are minor, *e.g.*, changing icons. We do not discuss GUI elements for a concise definition. It could be interesting if follow-up researchers identify and prioritize the obsolete GUI elements according to their relationship with descriptions.

4.2 Training Model

DOSUD takes the latest and past versions of an app as its inputs, and trains a classifier for each app. We train a separate classifier for each app because the features and layouts of each app vary considerably and influence the appearance of the screenshots. To handle this challenge, we train a

Algorithm 2: Random Forest Classifier Training

Input: *trainData*
Output: *trees*
1 $trees \leftarrow []$;
2 **for** $b = 1$ *to* B **do**
    // We set B to 100.
3   $z \xleftarrow{\text{bootstrap } N} trainData$;
4   $T_b \leftarrow \text{tree}(z)$;
5   **while** T_b *has a terminal node with both labels* **do**
6     $V_m \xleftarrow{\text{randomly select } m} V_p$;
      // We set m to $\sqrt{p}$.
7     $V \leftarrow \text{Gini}(V_m, node)$;
8   T_b.split($node, V$);
9   $trees$.append(T_b);
10 **return** *trees*;

classifier for each app individually to capture the specific characteristics of each app and enhance the accuracy of the classification.

4.2.1 Capturing App Screenshot. It is tedious and time-consuming to manually extract screenshots from an app, especially when it has complicated GUIs. To streamline this process, we categorize applications into two categories:

1. *Applications with testing scripts.* Some applications, *e.g.*, Visual Studio Code, have GUI test cases. When we execute test cases, we utilize Open Broadcaster Software (OBS) [11], an open-source software for video recording, to capture screenshots. It captures screenshots at a rate of 60 frames per second.

2. *Applications without testing scripts.* To automate the process, we use a GUI testing tool, called Fastbot [37]. It takes an APK as its input and captures its screenshots by traversing its GUIs with the Upper Confidence Bound (UCB) algorithm [39] and reinforcement learning (RL). UCB favors less-visited actions with higher uncertainty.

After DOSUD extracts screenshots from the apps, it encodes the screenshots as integer vectors and assigns labels to them automatically. Algorithm 1 illustrates the steps of capturing, encoding, and labeling screenshots. In Lines 6 to 9, screenshots from the latest version are labeled as `false`, indicating that they are not outdated, while screenshots from other versions are labeled as `true`, indicating that they are outdated. If a screenshot appears in both the latest version and an old version, it means that it has not changed, so Line 11 labels it as `false` as well. The output of this algorithm is the training set for *Screenshot Classifier*.

Although testing tools can fail to capture complete screenshots, the impact of missing screenshots can be minor, since many GUI documents show typical scenarios. The implications of incomplete screenshots are further discussed in Section 7. In addition, our trained model does not require exactly matched screenshots, and it works when similar screenshots are captured, owing to the utilization of a screenshot encoder.

4.2.2 Training Screenshot Classifier. *Screenshot Classifier* is based on a Random Forest (RF) classifier [36], which is an ensemble learning method that constructs multiple decision trees and aggregates their predictions by majority vote. To enable the classification, DOSUD encodes each screenshot into an integer vector. A screenshot is an image that is composed of a matrix of color

Algorithm 3: Random Forest Classifier Predicting

Input: x
Output: *ans*
1 **for** b = *1 to* B **do**
2 $\quad T_b(x) \leftarrow trees[b]$.predict($x$);
3 $ans \leftarrow$ majority vote $\{T_b(x)\}_1^B$;
4 **return** *ans*;

pixels, and each pixel encodes the color at a specific point with a color mode. A common color mode is RGB, where a pixel is represented by a triple of integers between 0 and 255, indicating the intensity of red, green, and blue components. For example, `(255, 0, 0)` denotes pure red. We convert each screenshot into a 3D matrix by replacing each pixel with its corresponding RGB triple, and flatten the 3D matrix into a fixed-length integer vector.

Screenshots in GUI documents can have different resolutions. To enable the comparison, our *Screenshot Handler* resizes screenshots to a specific target resolution $c \times r$. A higher target resolution preserves more details, but it increases the training and predicting time. Based on our empirical results in Section 6.5, we choose optimal target resolutions of $c = 36$ and $r = 64$ for the Android platform, and $c = 64$ and $r = 36$ for desktop environments.

The screenshots in GUI documents can have other resolutions. In image processing, image scaling [54] is the task of resizing digital images. If the target resolution is higher than the original resolution, image interpolation techniques [68] are used to generate new pixels. If the target resolution is lower than the original resolution, as in our case, sample-rate conversion [46] is applied to reduce the image size. This process involves information loss, and the Whittaker-Shannon interpolation formula [53] can construct a perfectly bandlimited signal. We use Lanczos filtering [44], an approximation to this formula, based on a Lanczos kernel:

$$L(x) = \begin{cases} 1, & x = 0 \\ \frac{2\sin(\pi x)\sin(\pi x/2)}{\pi^2 x^2}, & -2 \le x < 2, x \neq 0 \\ 0, & otherwise \end{cases} \tag{2}$$

Given a one-dimensional signal $\vec{s}$, the value at an arbitrary location x is obtained through the following equation:

$$S(x) = \sum_{i=\lfloor x \rfloor - \alpha + 1}^{\lfloor x \rfloor + \alpha} s_i L(x - i), \tag{3}$$

where α is the filter size parameter, $\lfloor \cdot \rfloor$ is the floor function, and L is the Lanczos kernel. This filter smoothly interpolates the values of a window of α samples. We choose this filter because it is more effective than other filters [74].

Screenshots on Android platforms have status bars at the top and navigation bars at the bottom. These sections display irrelevant or changing information during extracting process, such as battery percentage. Such information introduces noise and does not reflect outdated screenshots. To avoid inaccurate predictions, we crop them out of the images. Based on our empirical trials, DOSUD crops the top pixels and the bottom pixels to remove the status and navigation bars, respectively.

We present the Random Forest Classifier Training algorithm in Algorithm 2. The algorithm takes a set of training data as input and outputs a list of decision trees. Line 1 initializes an empty list of trees. Lines 2 to 9 iterate over a predefined number of trees (B). The bootstrap sample [45] is a classical sampling strategy. It randomly selects an instance from a pool, and returns the selected instance to the pool, before the next selection. With this strategy, Line 3 selects N instances. Line 4

creates an empty tree. Lines 5 to 8 recursively split the nodes of the tree until all terminal nodes have a single label. Line 6 randomly selects m variables from the p variables. The value of m is usually the square root of p or a value that is smaller than p. This makes the trees more general. Line 7 computes the Gini index for each variable and chooses the one with the lowest value as the best variable to split the node:

$$Gini(D) = 1 - \sum_{j=1}^{n} p_j^2, \tag{4}$$

where D is a dataset that is divided into n classes and p_j is the probability of an element being in the j-th class. The best split is the one that minimizes the Gini index. Line 8 splits the node into two child nodes based on the best variable. Line 9 adds the tree to the list of trees. Line 10 returns the list of trees as the output. In our implementation, we use the `RandomForestClassifier` from the `scikit-learn` library [25]. We use its default hyperparameters, *i.e.*, $B = 100$ and $m = \sqrt{p}$.

4.3 Detecting Outdated Screenshot

In this section, we introduce how DOSUD detects outdated screenshots. Given an app document, *Document Screenshot Extractor* first extracts its screenshots, and DOSUD uses its trained classifier to determine whether a screenshot is outdated.

4.3.1 Extracting Screenshot of Document. Our *Document Screenshot Extractor* extracts the screenshots from app documents. Most Android application documents are presented in the form of web pages on Google Play. Similarly, the majority of desktop application documents, such as those for Visual Studio Code, are typically maintained in GitHub repositories. Our extractor crawls the screenshots on the web page or the repository and saves the screenshots to a local directory. If an app document is in different formats (*e.g.*, PDF), a new extractor should be implemented, but that is a one-time effort.

4.3.2 Detecting Outdated Screenshot. After screenshots are extracted from application documents, Algorithm 3 illustrates the prediction process of detecting outdated screenshots. It is a random forest classifier. The input of the algorithm is a vector x that is encoded from a given screenshot. The output of the algorithm is the predicted class label for x, *i.e.*, outdated or not. A trained random forest is a set of built decision trees. The algorithm feeds x to each decision tree and stores the prediction in $T_b(x)$, where b is the index of a current tree. After all predictions are obtained, it takes the majority vote as the final output. If the majority vote is outdated, DOSUD marks the screenshot as outdated.

5 EVALUATION IN THE WILD

On benchmarks, it is infeasible to evaluate the usefulness of a tool. In addition, even if benchmarks are all real data, their settings can be different from the real situations. As a result, the effectiveness of an approach can be significantly reduced if it is deployed in the wild [69]. To resolve these concerns, in this section, we use DOSUD to detect outdated screenshots for 50 Android apps and VS Code in the wild.

Meanwhile, as we cannot identify all bugs of an application, in the wild, we cannot calculate the measures like f-score. We present a study on benchmarks in Section 6 and report such measures.

5.1 Setup

Pendlebury *et al.* [69] criticize that many approaches are not effective when they are evaluated in the wild. This evaluation is to answer this concern. To evaluate its effectiveness in the wild, we apply DOSUD to detect outdated screenshots from the latest app websites on VS Code documentation and

Table 1. Outdated screenshots detected by DOSUD.

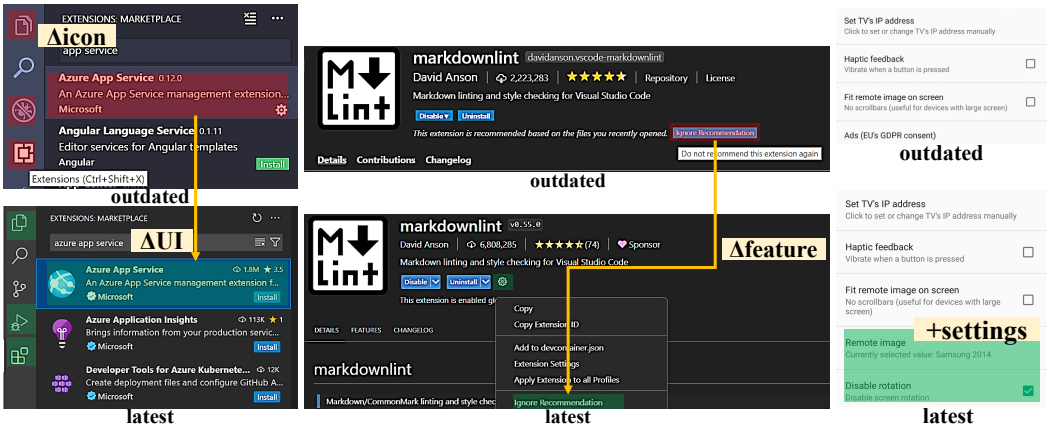
App Name	ΔScreenshot	Category			Issue URL	Status	PR URL	Text changed
		Improved UI	Feature	Setting				
Frequency Generator	+Mode buttons, +LR button		✓		n/a	n/a	n/a	n/a
CPU-Z	ΔTrophy button	✓						
Trigonometry	ΔLayout, +function icons	✓						
Perfect Ear	Δ"Theory" icon	✓						
Midifun Karaoke	+Buttons at the top, Δrecord button	✓	✓					
Maths Formulas Free	ΔButtons at the top, +formulas	✓						
Physics Formulas Free	ΔButtons at the top		✓					
Calculator	+Buttons at the top, Δbutton style	✓	✓					
Mi Calculator	+"Investment" button, Δ"Loan" button		✓					
Free scientific calculator	ΔButtons	✓						
MyObservatory	+Buttons at the bottom		✓					
Chinese Handwriting Recog	ΔBackground image, Δcolor style	✓						
Fractions	ΔOperator buttons	✓						
Thermonator	+Icon, +settings, +drop-down button		✓	✓				
Math Tests	+"Theory" button, +"More subjects" button		✓					
Master for Mi Band	Δversion number							
Remote for Samsung TV Smart	+Settings			✓				
Voice Recorder	ΔLayout, Δbuttons	✓						
Simple Alarm Clock	ΔLayout	✓						
Compass	+Settings, +prompt text			✓				
Visual Studio Code	+"Create an Azure Account" item, Δlayout, Δicons	✓	✓		[5]	fixed	[21]	yes
	Δ"Azure App Service" install layout, Δicons	✓			[5]	fixed	[21]	yes
	Δ"Remote Breakpoint" layout, Δicons	✓			[5]	fixed	[21]	yes
	Δ"Start Remote Debugging" layout, Δitems, Δicons	✓	✓		[5]	fixed	[21]	yes
	Δ"Notebook" layout, Δicons	✓			[6]	fixed	[24]	yes
	Δ"Notebook" layout, Δicons	✓			[6]	fixed	[24]	yes
	Δ"Notebook" layout, Δicons	✓			[6]	fixed	[24]	yes
	Δ"Notebook" layout, Δicons	✓			[6]	fixed	[24]	yes
	+"High Contrast" theme option, Δlayout, Δicons	✓	✓		[7]	fixed	[13]	no
	Δ"Extensions View Filter Menu" layout, Δitems	✓	✓		[8]	fixed	[23]	yes
	Δ"Ignore Recommendation" button, Δlayout	✓	✓		[8]	fixed	[23]	yes
	+info of "More" Button		✓		[8]	fixed	[23]	yes
	Δ"Dropdown" items		✓		[8]	fixed	[23]	yes
	Δ"Recommendations" layout, Δicons	✓	✓		[8]	fixed	[23]	yes
	Δ setting ID, Δ"Backup and Sync Settings" button	✓	✓		[16]	fixed	[27]	yes
	Δ"Backup and Sync Settings" button	✓			[16]	fixed	[27]	yes
	Δ"MongoDB" configuration layout, Δ buttons	✓	✓		[15]	fixed	[22]	yes

+: additions; Δ: modifications.

Google Play. VS Code documentation contains around 1,500 screenshots. For Android applications on Google Play, we select our subjects based on three criteria: (1) the website must contain at least one screenshot without decorations, and (2) the subjects must represent different app categories. In total, we randomly select the websites of 50 apps. To examine the generality of DOSUD, we choose different apps from those in Section 6. The websites should provide screenshots of the latest versions. If a screenshot does not appear in the latest version but in a previous version, we determine it as outdated.

We collect major versions for each application and train our model on them. Then, DOSUD takes the website address as its input and checks whether the website contains outdated screenshots. We manually verify the obsolescence of each identified screenshot by the following steps. First, we interact with the application by pressing buttons and switching between graphical user interfaces (GUIs) of all versions to understand its functionalities and workflows. Next, we inspect the outdated screenshot and learn when and where it can appear. Finally, we navigate to the corresponding GUI, compare the outdated screenshot with the latest GUI, and identify any major changes, such as a new button. If there are major changes, we further compare the screenshots with those from older versions. If a screenshot from an older version is identical to the detected screenshot, we confirm that the screenshot is outdated. Our identified screenshots are truly outdated because they do not require much programming expertise to verify whether they are true positive. We list all our detected bugs on our project website for other researchers to validate them.

Due to the lack of available contact information from Android app authors for their issue trackers, or instances where responses are not received, we were unable to report the Android bugs we detected. Although a reviewer suggested reporting them through their Android Marketplaces,



(a) Better UI. **(b)** Changed feature. **(c)** More settings.

Fig. 6. The categories of our detected bugs.

our reports to their websites were overwhelmed by comments from other users. Although we cannot obtain feedback from application programmers, VS code provides a channel to report our found bugs. Through this channel, we reported our detected VS Code bugs. In our bug reports, we provided the latest screenshots to allow developers to check and fix bugs.

5.2 Result

Among the 50 sampled Android apps, DOSUD detected 20 outdated screenshots from 20 apps. In addition, it detected 17 instances from the VS Code documentation. Across our reported 17 outdated screenshots, the programmers of the VS code documentation identified no false positives, and the precision could be considered as 100%. Table 1 shows our results. Column "App name" indicates the name of the app. Column "ΔScreenshot" illustrates the difference between the outdated screenshot and the corresponding latest screenshot. The "+" symbol means that the item is added to the latest version. The "Δ" symbol means that the item is altered. Column "Category" indicates the category of the bug. Based on the symptoms, we classify bugs into three categories:

1. 70% of our found bugs miss improved UI of apps. For example, as explained in this document [10], if users want to install the Azure App Service extension, they can search for "azure app service" to filter the results. Upon locating the desired extension, users can proceed to install it. The document has an embedded screenshot as shown in Figure 6a, but this screenshot is outdated. Its keyword is "app service". As shown in Figure 6a, in the latest version, three icons (highlighted in red) in VS Code have been replaced by their updated counterparts (indicated in green). Furthermore, the latest screenshot presents the UI for extension search within Visual Studio Code. The latest screenshots have more extension details (*e.g.*, new icons, stars, and download counts).

2. 49% of our found bugs miss important changed features. For example, as explained in this document [9], if users want to ignore the recommendation of an extension, they can click the Manage gear button to display the context menu, and select the Ignore Recommendation menu item. To help programmers understand the steps, this document provides the screenshot as shown in the upper half of Figure 6b. However, instead of explaining the steps, this screenshot causes confusion. In specific, this screenshot does not have the Manage gear button, and programmers cannot display the said context menu. DOSUD detects that this screenshot is outdated. The lower half of Figure 6b shows the latest one. We highlight the mentioned Manage gear button and context menu in green boxes. Furthermore, the yellow arrow presents the evolution.

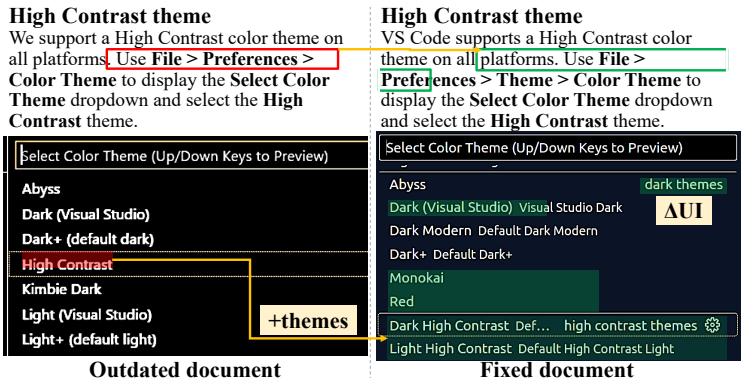


Fig. 7. An outdated path of the theme setting.

3. 8% of our found bugs miss changes on settings. For example, Samsung TV Smart [4] is an app that enables users to customize the Samsung TV. Figure 6c shows the latest version of the app, but the screenshot on its website is outdated. The difference between the two screenshots reveals that the latest version has more settings (*e.g.*, disabling rotation).

These examples highlight the usefulness and relevance of DOSUD for enhancing app documentation quality.

> **Finding 4:** The negative impacts of outdated screenshots include failing to show improved UIs (70%), new features (49%), and changed settings (8%).

We report our found bugs to VS Code. Table 1 shows the result. Column "Issue URL" lists our bug reports. We report bugs at the granularity of files, and each file may encompass multiple outdated screenshots. Columns "Status" and "PR URL" list the feedback from developers. Column "Text changed" lists the cases when both descriptions and screenshots are modified. Only one pull request modifies screenshots without changing the accompanying text. Although this pull request also affects reader comprehension, our other bug reports can have a deeper impact since developers need to modify both screenshots and textual descriptions to fix them. All the pull requests are submitted by their developers, highlighting the significance of outdated screenshots.

As DOSUD does not report the versions of screenshots, we do not list the versions of screenshots in Table 1. When the developers of the VS Code documentation determine whether a bug is true, they do not need the versions of screenshots. They can compare our reported screenshots with those of the latest version to make the decision. All our reported bugs have been confirmed by the developers. Section 2 already introduced an example of a fixed bug. We next introduce two additional bug reports:

1. An outdated path of the high-contrast themes. As shown in the left side of Figure 7, if programmers need to set the high contrast theme, they will read an outdated document, since this document provides an outdated path, **File** > **Preferences** > **Color Theme**. To understand this path, this document provides a screenshot. However, if programmers use the latest version of VS Code, they will be confused by the document, since there is no longer a **Color Theme** under **Preferences**. In the latest version, **Preferences** has another option called **Theme**, and **Color Theme** is now located under **Theme**. Specifically, programmers should navigate to the path **File** > **Preferences** > **Theme** > **Color Theme** and select the desired theme. Developers fixed this issue by updating the description but failed to update the accompanying screenshot. After we reported

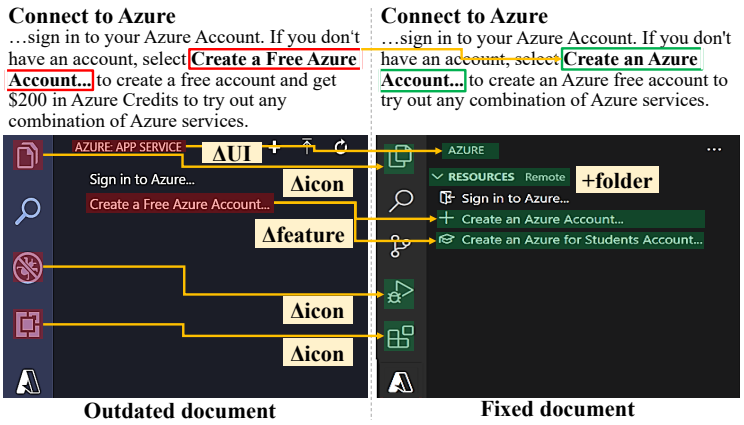


Fig. 8. Outdated Azure account actions.

this bug, the developers of VS Code fixed the screenshot. The right side of Figure 7 shows the fixed document. The fixed screenshot shows the latest interface. In specific, The dropdown now includes more theme choices, categorized into dark themes, high-contrast themes, and others (highlighted in green boxes). Moreover, the "High Contrast" theme has been split into two categories: dark and light (highlighted by the yellow arrow). Our bug report [7] was confirmed in one day. Three days later, a developer fixed this bug and commented that "*Excellent, Thanks!*"

2. Outdated Azure account actions. As shown in the left side of Figure 8, if programmers need to connect to Azure but have no account, they will read an outdated document. This document mentions a button called "Create a Free Azure Account" and uses a screenshot to explain its location. However, if programmers use the latest VS Code, they will be confused, since they cannot find the button as illustrated by the left side of Figure 8. The current version of VS Code splits the "Create a Free Azure Account" button into two new buttons: "Create an Azure Account" and "Create an Azure Account for Student Account". After we reported this bug, VS Code developers fixed this bug. The right side of Figure 8 shows the fixed document. Besides the above modifications, the outdated screenshots lose other modifications. For example, the `AZURE:APP_SERVICE` tag is renamed to `AZURE`. In Figure 8, we highlight the modifications with the yellow arrows. As this bug is critical, a developer of VS Code added our bug report [5] to the project milestones [18], signifying its substantial impact on the entire documentation.

The feedback from developers leads to a finding:

Finding 5: In total, we report 17 outdated screenshots, and 17 of them are already fixed.

In summary, our detected outdated screenshots fall into missing improved UIs, new features, and changed settings. From the latest version of the most popular IDE, we detected 17 outdated screenshots, and 17 of them are already fixed.

6 EVALUATION ON BENCHMARK

Our evaluations on benchmark explore the following research questions:

(RQ3) How effective is DOSUD (Section 6.2)?
(RQ4) What are the impacts of classifiers (Section 6.3)?
(RQ5) What are the impacts of filters (Section 6.4)?
(RQ6) What are the impacts of resolutions (Section 6.5)?

Table 2. Dataset.

App name	Category	Description	Versions	Screenshots	Reviews
Frequency Generator	Tool	It generates sounds with specified frequencies.	v2.0 v3.0 v4.0	723	19,090
CPU-Z	Tool	It reports information about the device.	v1.0 v1.20 v1.28 v1.33 v1.38	748	358,565
920	Productivity	A text editor.	v2.16.7.15 v2.16.9.7 v2.17.8.30	594	887
QR	Productivity	A QR code scanner.	v1.4.2 v2.0.2 v2.3.3 v2.6.7	617	851,148
Trigonometry	Education	It teaches trigonometry visually.	v3.1 v3.23	218	4,187
EveryCircuit	Education	A circuit simulator.	v2.14 v2.20 v2.25	454	50,959
2048	Game	A casual game.	v1.0 v2.0 v2.8	76	279,160
BBTAN	Game	A casual game.	v2.3 v3.0 v3.27	108	477,242
Cube Solver	Game	A puzzle game.	v3.3.1 v4.1.4 v4.3.0 v4.4.1	140	1,107,017
VS Code	IDE	A code editor	v17.0 v18.4	1,889	-

In RQ3, DOSUD achieved 0.87 to 1 F1-score values on a benchmark of 10 applications. Here, as there are no prior approaches, we cannot compare DOSUD with baselines. To shed light on future research, in RQ4, RQ5, and RQ6, we conduct ablation studies to explore the impact of our internal techniques. Our results show that internal classifiers have a more visible impact than resampling filters and target resolutions.

6.1 Benchmark

Table 2 shows our benchmark. It consists of 9 Android apps and a desktop app, VS Code. As the first exploration on this research topic, we select these applications since most GUI elements of them are standard and easy to enumerate. We focused on four categories from the Google Play Store, selecting both popular and lesser-known apps. We omitted Android applications if they do not provide their previous versions. Column "Versions" shows the versions of the app that we use. To reduce the training effort of our model, we only choose major versions. We select these versions within approximately equal time intervals of no more than one month. Column "Screenshots" shows the total number of unique screenshots that we capture from these apps using DOSUD. We set the time interval between any two actions as 300 ms and the time limit for the capturing process of each APK as 5 minutes. We empirically determine these limits based on their sufficiency to obtain enough screenshots of our subject apps. For Visual Studio Code (VS Code), OBS captures a video of approximately 8 to 10 minutes for testing scripts of each version. To capture all the scenarios, we process the video at a rate of 60 frames per second. Although we capture many duplicated screenshots, we remove duplicated screenshots to mitigate the potential bias by matching identical ones. In our datasets, all screenshots are unique. Column "Reviews" shows the number of reviews of each app on the Google Play page. We include both popular (*e.g.*, Cube Solver) and less popular ones (*e.g.*, 920). Here, 2048, BBTAN, and Cube Solver are games, and their workflows are more complicated than the other applications.

We do not need manual effort to build labels. DOSUD assigns labels (*i.e.*, `true` for outdated screenshots and `false` for the latest screenshots) to captured screenshots according to whether they are extracted from the latest version or an outdated version. Here, if a screenshot appears in both versions, it is labeled as `false`, since it indicates the absence of any modifications between the two iterations. Thus, it reflects the layout of the latest version.

As the GUIs of an application typically follow the same style, as expected, we found that some captured screenshots are similar. The observation highlights the challenges of our target research problem, since the screenshots of the latest versions can be similar to obsolete ones. The changes between versions may be subtle and not easily noticeable. Therefore, we propose to train a classifier that can detect outdated screenshots, rather than a simple "distance" metric that measures the similarity between screenshots. Meanwhile, from the applications like 2048, we captured limited screenshots. Although our underlying testing tool can fail to enumerate all possible GUIs, this

Table 3. Overall effectiveness.

App name	Accuracy	Precision	Recall	F1-score
Frequency Generator	1.00	1.00	1.00	1.00
CPU-Z	0.95	0.95	0.98	0.97
920	0.99	0.99	1.00	1.00
QR	0.93	0.94	0.95	0.95
Trigonometry	0.99	0.99	0.99	0.99
EveryCircuit	1.00	1.00	1.00	1.00
2048	0.91	0.90	0.88	0.89
BBTAN	0.98	0.98	0.98	0.98
Cube Solver	0.85	0.81	0.92	0.87
VS Code	0.98	0.97	0.98	0.98

limitation may not introduce a significant negative impact, since GUI documents often provide typical screenshots and our approach works for screenshots with subtle differences.

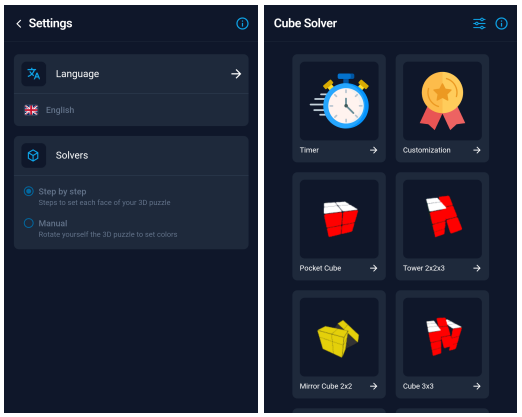
6.2 RQ3. Overall Effectiveness

6.2.1 Setup. For each application in Table 2, we train a model to predict its outdated screenshots. In particular, we apply a 10-fold cross-validation method [1] to our extracted data. The screenshot-label pairs are shuffled and equally divided into 10 subsets. Each subset serves as the test set once while the remaining 9 subsets constitute the training set. In each iteration, we use one of the 10 subsets as the testing data and the rest as the training data. We compare the predictions with the ground truth and categorize them into four groups: The true positive (TP) is the number of outdated screenshots that are correctly predicted as outdated. The false positive (FP) is the number of latest screenshots that are incorrectly predicted as outdated. The true negative (TN) is the number of latest screenshots that are correctly predicted as the latest. The false negative (FN) is the number of outdated screenshots that are incorrectly predicted as the latest. We use accuracy, precision, recall, and F1-score as our metrics. For all the metrics, a value that is closer to 1 indicates a better result.

6.2.2 Result. Training a model requires less than ten minutes. We demonstrate the effectiveness of our trained models in Table 3. Except for 2048 and Cube Solver, most applications have F1-score values that are close to 1. Nevertheless, these results do not imply that our target research problem has been solved completely. The majority of our selected apps have GUIs that are easy to enumerate, and such GUIs usually consist of standard or slightly modified menus, buttons, and other simple GUI elements so that it takes the GUI testing tool a short time to enumerate most of the GUIs. Based on our observations, we derive a finding:

Finding 6: The F1-scores of DOSUD are close to 1 when the apps have classical GUIs.

However, the effectiveness of DOSUD is reduced if applications have more complicated GUIs. For instance, our F1-score values of 2048 and Cube Solver drop to 0.89 and 0.87, respectively. Both applications are dynamic animation-based games, and their screenshots are complicated. Figure 9 shows a false positive and a true negative when DOSUD predicts screenshots for Cube Solver. Both screenshots in Figure 9 are captured from the latest version. DOSUD wrongly identifies that the screenshot in Figure 9a is outdated but correctly identifies that the screenshot in Figure 9b is not outdated. The number of similar screenshots can affect the results. The screenshot in Figure 9a has only 3 similar screenshots, but the screenshot in Figure 9b has 140 similar screenshots. As a result, DOSUD correctly identifies the latter screenshot.



(a) false positive **(b)** true negative

Fig. 9. Two cases of Cube Solver .

More advanced testing and static tools can capture more screenshots. With more screenshots, we can train more reliable models. We further discuss this issue in Section 8.

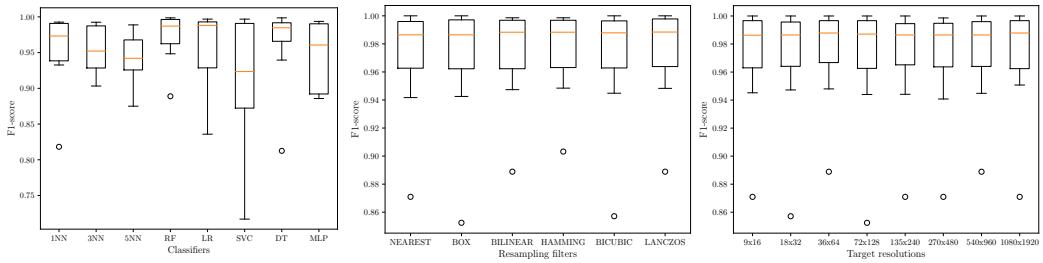
6.3 RQ4. Impact of Classifiers

6.3.1 Setting. In this research question, we explore the impact of our underlying classifier. We compare the underlying classifier with the nearest neighbor classifier (NN), the logistic regression classifier (LR), the Support Vector Classifier (SVC), the decision tree classifier (DT), and the multi-layer perceptron classifier (MLP) [70]. For each classifier, we conduct a 10-fold cross-validation on all the apps in our dataset. We use the default parameters for all the classifiers except KNN and MLP. KNN requires specifying the number of neighbors. For the k value NN, we selected 1, 3, and 5. MLP requires specifying the size of each hidden layer and learning rate. We set the `hidden_layer_sizes` as (128, 64) and `learning_rate` as 0.001. We compare F1-score values to determine the best one. Additionally, we conduct t-tests to evaluate the statistical significance of differences between the F1-score values of each two classifiers.

6.3.2 Result. Figure 10a presents the box plots of F1-score values. The horizontal axis represents the classifiers, where "1NN", "3NN", and "5NN" indicate the outcomes of the KNN classifier when k is 1, 3, and 5 respectively. The vertical axis shows F1-score values. According to our analysis of the KNN classifier, the optimal result is obtained when k equals one. The observations lead to a finding:

> **Finding 7:** Among our evaluated classifiers, RF is the best one for detecting outdated screenshots, but DT can achieve similar results.

Among all the combinations of classifiers, the minimum p-value is 0.033. We obtain this value when comparing SVC with RF. Only the difference of this pair is significantly different. The p-values between RF and other classifiers are all more than 0.26, indicating that the differences are insignificant. Our compared classifiers are classical, but researchers have proposed more advanced classification techniques (*e.g.*, [47]). Notably, we also included a deep learning classifier, MLP, in our comparison. Despite its increased model complexity, MLP does not outperform classical classifiers such as RF and DT in our setting. Finding 1 demonstrates that DOSUD is already a highly accurate classifier when apps use GUIs that are easy to enumerate. For these apps, a better classifier may not introduce significant improvements. Finding 1 also mentions that it is challenging to detect outdated



(a) Internal classifiers. **(b)** Resampling filters. **(c)** Target resolution.

Fig. 10. The result of our ablation studies.

screenshots for more complicated apps (*e.g.*, games). For these applications, GUI elements are much more flexible. For example, many mobile games are built upon various game frameworks (*e.g.*, unity [55]). As their screenshots can be infinite, a more sophisticated classifier alone is insufficient to address the challenges, and we discuss this issue in Section 8.

6.4 RQ5. Impact of Resampling Filters

6.4.1 Setting. In this research question, we compare our resampling filter with the other filters. Given a one-dimensional signal, the filters select the value at a location with 6 following strategies. **1) LANCZOS** is our underlying filter. **2) NEAREST** picks the pixel that is the nearest to the selected location from an image. **3) BOX** generates a pixel that is the mean of input pixels. **4) BILINEAR** generates a pixel with linear interpolation [61] on all input pixels. **5) HAMMING** generates a pixel with the Hamming network [56] on all input pixels. **6) BICUBIC** generates a pixel with cubic interpolation [61] on all input pixels.

For each filter, we conduct a 10-fold cross-validation on all the apps in Table 2 and use F1-score as the metric. Additionally, we conduct t-tests to evaluate the statistical significance of differences between the F1-score values of each two resampling filters.

6.4.2 Result. Figure 10b shows the results. The horizontal axis lists the filters. For all the measures, the filters do not introduce significant differences. The minimum p-value observed between different filters is 0.79. It is obtained when we compare BOX with LANCZOS. It indicates that the performance differences among the filters are not statistically significant. The observation leads to a finding:

> **Finding 8:** Empirically, we find that LANCZOS is the best filter, but the differences are minor.

According to this finding, in other research questions, we set the resampling filter as LANCZOS.

6.5 RQ6. Impact of Target Resolutions

6.5.1 Setting. In this research question, we explore the impacts of our target resolution. We compare our target resolution (36×64 as described in Section 4.2.2) with several alternatives. For each resolution, we conduct a 10-fold cross-validation on all the apps in Table 2, and draw the box plots of their F1-score. Additionally, we conduct t-tests to evaluate the statistical significance of differences between the F1-score values of each two target resolutions.

6.5.2 Result. Figure 10c shows the box plots of F1-score values. The storage space required for a screenshot increases with its resolution. Consequently, higher-resolution screenshots entail longer training and prediction times for a model. However, lower-resolution screenshots may compromise the quality of the information captured. Therefore, all the measures except the time deteriorate

with lower resolutions. Moreover, we observe that beyond 36×64, further increasing the resolution does not improve the performance, but only adds to the time cost. For all measures, the target resolutions do not introduce significant differences. The minimum p-value observed between different resolutions is 0.80 (between 36×64 and 72×128). This observation leads us to a finding:

Finding 9: The optimal target resolution is 36×64, but the differences are minor.

According to this finding, in other research questions, we set the target resolution as 36×64.

7 LIMITATION AND THREAT

In this section, we discuss the limitations and threats of our work.

1. GUI documents can contain rare screenshots. Our approach can fail to identify outdated screenshots if documents contain rare screenshots or if captured screenshots are insufficient. Although DOSUD can fail in such rare cases, its impact can be minor. GUI documents often describe typical scenarios, and they may not use rare screenshots. Furthermore, although 2048 has only 76 screenshots, our model still achieved a relatively high F1-score of 0.89. This result shows the robustness of our approach. We agree that we can train a more reliable model if more screenshots are captured. The current implementation of DOSUD uses Fastbot to capture screenshots and build training data. When we build the training data for VSCode, we use its manually written testing scripts. Besides the two sources, we envisage that more advanced testing and static tools are useful for capturing screenshots. For instance, GUI testing is intensively studied [49, 67]. Furthermore, some applications are open source. Through static analysis, we can extract GUI elements from source code and metadata [62]. It is challenging to capture new screenshots, since most testing scripts are written for regression testing. The above approaches are useful for capturing new screenshots if they contain new features. With a larger dataset, we can train a more reliable model, but more screenshots are insufficient to handle more complicated applications. For instance, if a GUI page has a box to display dynamic text contents, it is infeasible to capture all its screenshots. In this case, our model can determine which pixels are useful for determining outdated screenshots.

2. We must train a model for each application and retrain it when a new version is released. In our evaluations, we train a model for each project. Although training a unified model is an ambitious research goal, it is quite difficult to achieve the goal, since applications can have quite different GUIs. This limitation is shared with all other approaches. For instance, researchers have proposed various approaches to detect bugs, but a survey [50] shows that in most cases, the effectiveness of cross-project learning is still significantly poorer than in-project learning. In addition, a trained model can decay over time. If a new version is released, to achieve the best results, users need to re-train the model. As noted in Section 6.2.2, training takes less than ten minutes. Although the cost of a single update is low, the cumulative effort required for numerous historical versions can be burdensome. However, if two updates occur within a short time interval, the GUI elements may remain largely unchanged, thereby reducing the practical retraining cost. A potential solution is transfer learning, a technique in machine learning that uses domain adaptation to address the problem of concept drift [59, 65, 66]. Although it is infeasible to train a unified model, we can transfer trained models to new projects and versions. In addition, if we analyze the UI changes in revision histories, it could be feasible to train more effective models. Alternatively, we can extract reliable rules from revision histories to prioritize the obsolete elements of screenshots.

3. We did not obtain the feedback from Android programmers. Our selected Android applications provide no channel to report bugs, and their programmers did not reply to our emails. As a result, our found Android bugs in Table 1 have no feedback from Android programmers.

However, unlike other types of bugs, it does not require much programming experience to determine whether a screenshot is outdated. As shown in Figure 2b, most reporters have less than three years of programming experience when they identify outdated screenshots. Detecting outdated screenshots is challenging, since there are many screenshots in documents and applications that have complicated GUIs. Still, after they are detected, programmers are unlikely to introduce human errors when determining whether they are truly outdated. As shown in Table 1, all our reported VS Code bugs are confirmed and fixed. As a result, although we found Android bugs with no feedback, they are unlikely to be wrong. To further reduce the threat, we release our found bugs on our website, and other researchers can check them.

Besides the two limitations, we have other threats that appear in all studies. For instance, one external threat to validity is the limited number of subjects. The effectiveness of DOSUD on more complex applications (e.g., games) may vary. While our analysis includes 50 Android applications and Visual Studio Code in Section 5, alongside 9 Android applications and Visual Studio Code in Section 6, this threat could be mitigated by analyzing a greater number of subjects.

8 FUTURE RESEARCH OPPORTUNITY

Besides tuning our approach [32, 57], some other future research directions are as follows:

Handling more complicated applications. Our results on benchmarks can be significantly reduced, if applications produce more complicated screenshots. There are many research opportunities to handle complicated screenshots. For example, a game can have various characters that can move around scenes. After its screenshots are captured by corresponding testing techniques [34], face recognition techniques [80] can be useful for detecting outdated ones. As another example, some screenshots can be triggered by complicated inputs (*e.g.*, usernames). As it is difficult to extract complete screenshots, the screenshots in the training set can have different configurations and themes from those in the testing set. More advanced testing techniques can be useful to trigger such screenshots. In some documents, screenshots can be decorated or incomplete. Image segmentation [48] is the task of identifying the meaningful parts of an image, and these techniques may help address these problems. Moreover, computer vision algorithms such as SIFT [60] and SURF [35] can extract image features that are invariant to positions, scales, and rotations. These algorithms may be useful for handling decorated or incomplete screenshots. When programmers determine whether a reported outdated screenshot is a true bug, they need to compare the screenshot with only the latest version. Although they do not need the version of the screenshot, it could be helpful if a tool predicts the precise versions of screenshots. Programmers can have more confidence to determine whether a screenshot is outdated.

Handling more complicated documents. Some documents may include both updated screenshots and screenshots of a specific version. For instance, a report of a recurring bug may have screenshots of a previous version. Since such screenshots often contain contextual natural language descriptions, natural language processing techniques [41] may help determine whether the screenshots of a specific version are appropriate for a bug report. It may even be possible to detect some outdated screenshots from the documents alone. For example, if a tutorial mentions a button that does not appear in the screenshots, this may indicate a discrepancy between the descriptions and the screenshots. It can become challenging for programmers to check outdated screenshots manually even if our tool identifies them. In addition to processing screenshots at the pixel level, researchers have explored encoding app screenshots considering GUI widgets [51, 84] and migrating GUI test cases based on widgets [79]. If we extend our approach to work on widgets, programmers can understand our results more easily. For instance, besides reporting the bug in Figure 6b, the envisaged approach can detect that the two GUI elements, such as “License” and

"Repository", are missing. Programmers can determine whether this screenshot is truly outdated by checking the two GUI elements.

9 RELATED WORK

Detecting bugs in software engineering documents. Software engineering documents (*e.g.*, bug reports) differ from natural language texts in that they often contain code elements that need to be consistent and accurate. Therefore, traditional methods for bug detection are not suitable for these documents. Previous studies have addressed this challenge by focusing on specific types of bugs, such as outdated or broken code names [71, 81], uncertainty cues [72], outdated API names [58], wrong directive defects [82], missing descriptions [40], and errors in Solidity smart contract API documentation [83]. These studies rely on identifying code samples and matching them with natural language texts or code changes. However, they do not consider bugs that involve images in documents, which can also convey important information. We are the first to detect outdated screenshots in GUI documents by leveraging computer vision and machine learning techniques, complementing the prior approaches.

Screenshots in software engineering. Screenshots are visual representations of software applications that can be used for various purposes in software engineering research. For example, Yeh *et al.* [77] employ screenshots as queries to search documents. Deka *et al.* [43] construct a large dataset of app screenshots to support the design of new apps. Souza *et al.* [73] propose an approach to generate user guides with captured screenshots. Martens *et al.* [63] use screenshots to collect fake reviews in app stores. Yu *et al.* [78] use screenshots to prioritize crowdsourced test reports. Wang *et al.* [75] utilized both textual data and screenshots to identify duplicate bug reports within bug management systems. Cooper *et al.* [42] and Yan *et al.* [76] extract visual features from screenshots, which are sampled from videos, to detect duplicate video-based bug reports. Furthermore, Yan *et al.* [76] employ vision transformers to discern subtle visual and textual patterns, thereby enhancing the detection of duplicate video-based bug reports. The quality and accuracy of screenshots can affect the effectiveness of these approaches. Therefore, detecting outdated screenshots is an important task that can improve the quality of software engineering documents.

10 CONCLUSION

Screenshots can enhance the understanding of these documents, but they may become outdated when the applications change over time. The problem of outdated screenshots is prevalent in the documents of popular software projects, and it requires considerable effort from developers to update them manually. However, this problem has received little attention from the research community. Detecting outdated screenshots is a challenging task that involves interdisciplinary techniques from various research fields. In this paper, we propose DOSUD, the first approach that automatically detects outdated screenshots from documents. We evaluated our approach on benchmarks and in the wild. The results show that DOSUD is effective on benchmarks and it detects new bugs in the wild.

DATA-AVAILABILITY STATEMENT

More details are presented on our website: https://anonymous.4open.science/r/DOSUD-FDD3.

REFERENCES

[1] K-folds cross-validator. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html, 2020.
[2] Content is inconsistent with images. https://github.com/MicrosoftDocs/azure-docs/issues/73030, 2021.
[3] Microsoft Documentation. https://github.com/MicrosoftDocs/, 2021.

[4] Remote for Samsung TV | Smart & WiFi Direct. https://play.google.com/store/apps/details?id=smart.tv.wifi.remote.control.samcontrol, 2021.
[5] /docs/azure/remote-debugging.md has outdated screenshot. https://github.com/microsoft/vscode-docs/issues/6866, 2023.
[6] /docs/datascience/jupyter-notebooks.md has outdated screenshots. https://github.com/microsoft/vscode-docs/issues/6878, 2023.
[7] /docs/editor/accessibility.md has outdated screenshot. https://github.com/microsoft/vscode-docs/issues/6883, 2023.
[8] /docs/editor/extension-marketplace.md has outdated screenshot. https://github.com/microsoft/vscode-docs/issues/6884, 2023.
[9] Ignoring recommendations. https://github.com/microsoft/vscode-docs/blob/main/docs/editor/extension-marketplace.md#ignoring-recommendations, 2023.
[10] Install the extension. https://github.com/microsoft/vscode-docs/blob/main/docs/azure/remote-debugging.md#install-the-extension, 2023.
[11] Open Broadcaster Software (OBS). https://obsproject.com/, 2023.
[12] Stack Overflow 2023 Developer Survey. https://survey.stackoverflow.co/2023/, 2023.
[13] Update high-contrast theme image. https://github.com/microsoft/vscode-docs/pull/6893, 2023.
[14] Visual studio code documentation. https://github.com/microsoft/vscode-docs, 2023.
[15] /docs/azure/mongodb.md has outdated screenshot. https://github.com/microsoft/vscode-docs/issues/7285, 2024.
[16] /docs/supporting/troubleshoot-terminal-launch.md has outdated screenshot. https://github.com/microsoft/vscode-docs/issues/7204, 2024.
[17] Focus is not visible for the dismiss this update button in the version info bar. https://github.com/microsoft/vscode-docs/issues/7418, 2024.
[18] Milestones: April 2024. https://github.com/microsoft/vscode-docs/milestone/131?closed=1, 2024.
[19] Outdated gif for “add a jar”. https://github.com/microsoft/vscode-docs/issues/4948, 2024.
[20] Outdated screenshot in word count extension tutorial. https://github.com/microsoft/vscode-docs/issues/1405, 2024.
[21] Refresh azure remote debugging content. https://github.com/microsoft/vscode-docs/pull/7222, 2024.
[22] Refresh azure/mongodb screenshots. https://github.com/microsoft/vscode-docs/pull/7318, 2024.
[23] Refresh extensions marketplace screenshots & content. https://github.com/microsoft/vscode-docs/pull/7024, 2024.
[24] Refresh notebooks screenshots. https://github.com/microsoft/vscode-docs/pull/7435, 2024.
[25] scikit-learn: machine learning in python. https://scikit-learn.org/, 2024.
[26] Should you add screenshots to documentation? https://news.ycombinator.com/item?id=38639629, 2024.
[27] Update terminal troubleshooting screenshots. https://github.com/microsoft/vscode-docs/pull/7260, 2024.
[28] Update website main screenshots. https://github.com/microsoft/vscode-docs/issues/1124, 2024.
[29] Vs code 1.69 release notes - emmet - inline completions setting - incorrect setting name. https://github.com/microsoft/vscode-docs/issues/5465, 2024.
[30] Vs code microsoft software license terms. https://code.visualstudio.com/License, 2024.
[31] vscecli doc outdated. https://github.com/microsoft/vscode-docs/issues/550, 2024.
[32] N. A. Al-Thanoon, O. S. Qasim, and Z. Y. Algamal. Tuning parameter estimation in SCAD-support vector machine using firefly algorithm with application in gene selection and cancer classification. *Computers in biology and medicine*, 103:262–268, 2018.
[33] E. Alégroth, R. Feldt, and L. Ryrholm. Visual GUI testing in practice: challenges, problemsand limitations. *Empirical Software Engineering*, 20(3):694–744, 2015.
[34] S. Ariyurek, A. Betin-Can, and E. Surer. Automated video game testing using synthetic and humanlike agents. *IEEE Transactions on Games*, 13(1):50–67, 2019.
[35] H. Bay, T. Tuytelaars, and L. V. Gool. Surf: Speeded up robust features. *Springer-Verlag*, 2006.
[36] L. Breiman. Random Forests. *Machine Learning*, 2001.
[37] T. Cai, Z. Zhang, and P. Yang. Fastbot: A Multi-Agent Model-Based Test Generation System. In *Proc. AST*, pages 93–96, 2020.
[38] Z. Cao, X. Wang, S. Yu, Y. Yun, and C. Fang. STIFA: Crowdsourced Mobile Testing Report Selection Based on Text and Image Fusion Analysis. In *Proc. ASE*, pages 1331–1335, 2020.
[39] A. Carpentier, A. Lazaric, M. Ghavamzadeh, R. Munos, and P. Auer. *Upper-Confidence-Bound Algorithms for Active Learning in Multi-Armed Bandits*. Springer Berlin Heidelberg, 2011.
[40] O. Chaparro, J. Lu, F. Zampetti, L. Moreno, M. Di Penta, A. Marcus, G. Bavota, and V. Ng. Detecting missing information in bug descriptions. In *Proc. ESEC/FSE*, pages 396–407, 2017.
[41] K. Chowdhary and K. Chowdhary. Natural language processing. *Fundamentals of artificial intelligence*, pages 603–649, 2020.

[42] N. Cooper, C. Bernal-Cárdenas, O. Chaparro, K. Moran, and D. Poshyvanyk. It takes two to tango: Combining visual and textual information for detecting duplicate video-based bug reports. In *Proc. ICSE*, pages 957–969. IEEE, 2021.
[43] B. Deka, Z. Huang, C. Franzen, J. Hibschman, and R. Kumar. Rico: A Mobile App Dataset for Building Data-Driven Design Applications. In *Proc. UIST*, pages 845–854, 2017.
[44] C. Duchon. Lanczos Filtering in One and Two Dimensions. *Journal of Applied Meteorology and Climatology*, 18:1016–1022, 08 1979.
[45] B. Efron and R. J. Tibshirani. *An introduction to the bootstrap*. CRC press, 1994.
[46] G. Evangelista. Design of digital systems for arbitrary sampling rate conversion. *Signal processing*, 83(2):377–387, 2003.
[47] H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller. Deep learning for time series classification: a review. *Data Mining and Knowledge Discovery*, 33(4):917–963, 2019.
[48] P. F. Felzenszwalb and D. P. Huttenlocher. Efficient Graph-Based Image Segmentation. *International Journal of Computer Vision*, 59(2):167–181, 2004.
[49] T. Fulcini, R. Coppola, L. Ardito, and M. Torchiano. A review on tools, mechanics, benefits, and challenges of gamified software testing. *ACM Computing Surveys*, 55(14s):1–37, 2023.
[50] S. Hosseini, B. Turhan, and D. Gunarathna. A systematic literature review and meta-analysis on cross project defect prediction. *IEEE Transactions on Software Engineering*, 45(2):111–147, 2017.
[51] Y. Hu, J. Gu, S. Hu, Y. Zhang, W. Tian, S. Guo, C. Chen, and Y. Zhou. Appaction: Automatic GUI Interaction for Mobile Apps via Holistic Widget Perception. In *Proc. ESEC/FSE*, pages 1786–1797, 2023.
[52] T. K. Huang. Investigating user acceptance of a screenshot-based interaction system in the context of advanced computer software learning. In *Proc. HICSS*, pages 4956–4965, 2014.
[53] A. J. Jerri. The Shannon sampling theorem—Its various extensions and applications: A tutorial review. *Proceedings of the IEEE*, 65(11):1565–1596, 1977.
[54] C.-H. Kim, S.-M. Seong, J.-A. Lee, and L.-S. Kim. Winscale: An image-scaling algorithm using an area pixel model. *IEEE Transactions on circuits and systems for video technology*, 13(6):549–553, 2003.
[55] S. L. Kim, H. J. Suk, J. H. Kang, J. M. Jung, T. H. Laine, and J. Westlin. Using Unity 3D to facilitate mobile augmented reality game development. In *Proc. WF-IoT*, pages 21–26, 2014.
[56] K. Koutroumbas and N. Kalouptsidis. Generalized Hamming networks and applications. *Neural Networks*, 18(7):896–913, 2005.
[57] A. Kulkarni, V. K. Jayaraman, and B. D. Kulkarni. Support vector classification with parameter tuning assisted by agent-based technique. *Computers & chemical engineering*, 28(3):311–318, 2004.
[58] S. Lee, R. Wu, S. C. Cheung, and S. Kang. Automatic Detection and Update Suggestion for Outdated API Names in Documentation. *IEEE Transaction on Software Engineering*, pages 1–1, 2019.
[59] S. Liu, G. Lin, L. Qu, J. Zhang, O. De Vel, P. Montague, and Y. Xiang. CD-VulD: Cross-domain vulnerability discovery based on deep domain adaptation. *IEEE Transactions on Dependable and Secure Computing*, 19(1):438–451, 2020.
[60] D. G. Lowe. Object recognition from local scale-invariant features. In *Proc. ICCV*, 1999.
[61] E. Maeland. On the comparison of interpolation methods. *IEEE transactions on medical imaging*, 7(3):213–217, 1988.
[62] J. Mahmud, N. De Silva, S. A. Khan, S. H. Mostafavi, S. H. Mansur, O. Chaparro, A. Marcus, and K. Moran. On Using GUI Interaction Data to Improve Text Retrieval-based Bug Localization. In *Proc. ICSE*, pages 1–13, 2024.
[63] D. Martens and W. Maalej. Towards Understanding and Detecting Fake Reviews in App Stores. *Empirical Software Engineering*, pages 3316–3355, 2019.
[64] S. Moir. Should you add screenshots to documentation? https://thisisimportant.net/posts/screenshots-in-documentation/, 2023.
[65] V. Nguyen, T. Le, O. de Vel, P. Montague, J. Grundy, and D. Phung. Dual-component deep domain adaptation: A new approach for cross project software vulnerability detection. In *Proc. PAKDD*, pages 699–711, 2020.
[66] V. Nguyen, T. Le, T. Le, K. Nguyen, O. DeVel, P. Montague, L. Qu, and D. Phung. Deep domain adaptation for vulnerable code function identification. In *Proc. IJCNN*, pages 1–8, 2019.
[67] L. Nie, K. S. Said, L. Ma, Y. Zheng, and Y. Zhao. A systematic mapping study for graphical user interface testing on mobile apps. *IET Software*, 17(3):249–267, 2023.
[68] P. Parsania and P. Virpari. A review: Image interpolation techniques for image scaling. *International Journal of Innovative Research in Computer and Communication Engineering*, 2(12):7409–7414, 2014.
[69] F. Pendlebury, F. Pierazzi, R. Jordaney, J. Kinder, and L. Cavallaro. {TESSERACT}: Eliminating experimental bias in malware classification across space and time. In *Proc. USENIX Security*, pages 729–746, 2019.
[70] M.-C. Popescu, V. E. Balas, L. Perescu-Popescu, and N. Mastorakis. Multilayer perceptron and neural networks. *WSEAS Transactions on Circuits and Systems*, 8(7):579–588, 2009.
[71] P. C. Rigby and M. P. Robillard. Discovering Essential Code Elements in Informal Documentation. In *Proc. 35th ICSE*, page 11, 2013.

[72] K. Shumaiev and M. Bhat. Automatic Uncertainty Detection in Software Architecture Documentation. In *Proc. ICSAW*, pages 216–219, 2017.
[73] R. Souza and A. Oliveira. GuideAutomator: Continuous Delivery of End User Documentation. In *Proc. ICSE*, 2017.
[74] K. Turkowski. Filters for common resampling tasks. *Graphics Gems I*, pages 147–165, 1990.
[75] J. Wang, M. Li, S. Wang, T. Menzies, and Q. Wang. Images don't lie: Duplicate crowdtesting reports detection with screenshot information. *Information and Software Technology*, 110:139–155, 2019.
[76] Y. Yan, N. Cooper, O. Chaparro, K. Moran, and D. Poshyvanyk. Semantic GUI Scene Learning and Video Alignment for Detecting Duplicate Video-based Bug Reports. In *Proc. ICSE*, pages 1–13, 2024.
[77] T. Yeh, T. H. Chang, and R. C. Miller. Sikuli: Using GUI screenshots for search and automation. In *Proc. UIST*, pages 183–192, 2009.
[78] S. Yu, C. Fang, Z. Cao, X. Wang, T. Li, and Z. Chen. Prioritize crowdsourced test reports via deep screenshot understanding. In *Proc. ICSE*, pages 946–956, 2021.
[79] Y. Zhang, W. Zhang, D. Ran, Q. Zhu, C. Dou, D. Hao, T. Xie, and L. Zhang. Learning-based Widget Matching for Migrating GUI Test Cases. In *Proc. ICSE*, pages 1–13, 2024.
[80] W. Zhao, R. Chellappa, P. J. Phillips, and A. Rosenfeld. Face recognition: A literature survey. *ACM computing surveys*, 35(4):399–458, 2003.
[81] H. Zhong and Z. Su. Detecting API documentation errors. In *Proc. OOPSLA*, pages 803–816, 2013.
[82] Y. Zhou, X. Yan, T. Chen, S. Panichella, and H. Gall. DRONE: A Tool to Detect and Repair Directive Defects in Java APIs Documentation. In *Proc. ICSE*, pages 115–118, 2019.
[83] C. Zhu, Y. Liu, X. Wu, and Y. Li. Identifying Solidity Smart Contract API Documentation Errors. In *Proc. ASE*, pages 1–13, 2022.
[84] P. Zhu, Y. Li, T. Li, W. Yang, and Y. Xu. Gui widget detection and intent generation via image understanding. *IEEE Access*, 9:160697–160707, 2021.