It Takes Two to TANGO: Combining Visual and Textual Information for Detecting Duplicate Video-Based Bug Reports

Nathan Cooper*, Carlos Bernal-Cárdenas*, Oscar Chaparro*, Kevin Moran†, Denys Poshyvanyk*
*College of William & Mary (Williamsburg, VA, USA), †George Mason University (Fairfax, VA, USA)
nacooper01@email.wm.edu, cebernal@cs.wm.edu, oscarch@wm.edu, kpmoran@gmu.edu, denys@cs.wm.edu

***Abstract*—When a bug manifests in a user-facing application, it is likely to be exposed through the graphical user interface (GUI). Given the importance of visual information to the process of identifying and understanding such bugs, users are increasingly making use of screenshots and screen-recordings as a means to report issues to developers. However, when such information is reported *en masse*, such as during crowd-sourced testing, managing these artifacts can be a time-consuming process. As the reporting of screen-recordings in particular becomes more popular, developers are likely to face challenges related to manually identifying videos that depict duplicate bugs. Due to their graphical nature, screen-recordings present challenges for automated analysis that preclude the use of current duplicate bug report detection techniques. To overcome these challenges and aid developers in this task, this paper presents TANGO, a duplicate detection technique that operates purely on video-based bug reports by leveraging *both* visual and textual information. TANGO combines tailored computer vision techniques, optical character recognition, and text retrieval. We evaluated multiple configurations of TANGO in a comprehensive empirical evaluation on 4,860 duplicate detection tasks that involved a total of 180 screen-recordings from six Android apps. Additionally, we conducted a user study investigating the effort required for developers to manually detect duplicate video-based bug reports and compared this to the effort required to use TANGO. The results reveal that TANGO's optimal configuration is highly effective at detecting duplicate video-based bug reports, accurately ranking target duplicate videos in the top-2 returned results in 83% of the tasks. Additionally, our user study shows that, on average, TANGO can reduce developer effort by over 60%, illustrating its practicality.**

***Index Terms*—Bug Reporting, Screen Recordings, Duplicate Detection**

I. INTRODUCTION

Many modern mobile applications (apps) allow users to report bugs in a graphical form, given the GUI-based nature of mobile apps. For instance, Android and iOS apps can include built-in screen-recording capabilities in order to simplify the reporting of bugs by end-users and crowd-testers [10, 12, 18]. The reporting of visual data is also supported by many crowd-testing and bug reporting services for mobile apps [8–13, 15, 17–19], which intend to aid developers in collecting, processing, and understanding the reported bugs [26, 77].

The proliferation of sharing images to convey additional context for understanding bugs, *e.g.,* in Stack Overflow Q&As, has been steadily increasing over the last few years [81]. Given this and the increased integration of screen capture technology into mobile apps, developers are likely to face a growing set of challenges related to processing and managing app screen-recordings in order to triage and resolve bugs — and hence maintain the quality of their apps.

One important challenge that developers will likely face in relation to video-related artifacts is determining whether two videos depict and report the same bug (*i.e.,* detecting duplicate video-based bug reports), as it is currently done for textual bug reports [27, 86, 87]. When video-based bug reports are collected at scale, either via a crowdsourced testing service [8–13, 15, 17–19] or by popular apps, the sizable corpus of collected reports will likely lead to significant developer effort dedicated to determining if a new bug report depicts a previously-reported fault, which is necessary to avoid redundant effort in the bug triaging and resolution process [26, 27, 73, 77]. In a user study which we detail later in this paper (Sec. III-E), we investigated the effort required for experienced programmers to identify duplicate video-based bug reports and found that participants reported a range of difficulties for the task (*e.g.,* a single change of one step can result in two very similar looking videos showing entirely different bugs), and spent about 20 seconds of comprehension effort on average per video viewed. If this effort is extrapolated to the large influx of bug reports that could be collected on a daily basis [27, 35, 86, 87], it illustrates the potential for the excessive effort associated with video-based duplicate bug detection. This is further supported by the plans of a large company that supports crowd-sourced bug reporting (name omitted for anonymity), which we contacted as part of eliciting the design goals for this research, who stated that they anticipate increasing developer effort in managing video-based reports and that they are planning to build a feature in their framework to support this process.

To aid developers in determining whether video-based bug reports depict the same bug, this paper introduces TANGO, a novel approach that analyzes both visual and textual information present in mobile screen-recordings using tailored computer vision (CV) and text retrieval (TR) techniques, with the goal of generating a list of candidate videos (from an issue tracker) similar to a target video-based report. In practice, TANGO is triggered upon the submission of a new video-based report to an issue tracker. A developer would then use TANGO to retrieve the video-based reports that are most similar (*e.g.,* top-5) to the incoming report for inspection. If duplicate videos are found in the ranked results, the new bug report can be marked as a duplicate in the issue tracker. Otherwise, the developer can continue to inspect the ranked results until she has enough confidence that the newly reported bug was not reported before (*i.e.,* it is not a duplicate).

TANGO operates *purely* upon the graphical information in videos in order to offer flexibility and practicality. These videos may show the *unexpected behavior* of a mobile app (*i.e.,* a crash or other misbehavior) and the *steps to reproduce* such behavior. Two videos are considered to be *duplicates* if they show the same unexpected behavior (*aka* a bug) regardless of the steps to reproduce the bug. Given the nature of screen-recordings, video-based bug reports are likely to depict unexpected behavior towards the end of the video. TANGO attempts to account for this by leveraging the temporal nature of video frames and weighting the importance of frames towards the end of videos more heavily than other segments.

We conducted two empirical studies to measure: (i) the *effectiveness* of different configurations of TANGO by examining the benefit of combining visual and textual information from videos, as opposed to using only a single information source; and (ii) TANGO's ability to *save developer effort* in identifying duplicate video-based bug reports. To carry out these studies, we collected a set of 180 video-bug reports from six popular apps used in prior research [25, 34, 78, 79], and defined 4,860 duplicate detection tasks that resemble those that developers currently face for textual bug reports – wherein a corpus of potential duplicates must be ranked according to their similarity to an incoming bug report.

The results of these studies illustrate that TANGO's most effective configuration, which selectively combines visual and textual information, achieves 79.8% mRR and 73.2% mAP, an average rank of 1.7, a HIT@1 of 67.7%, and a HIT@2 of 83%. This means that TANGO is able to suggest correct duplicate reports in the top-2 of the ranked candidates for 83% of duplicate detection tasks. The results of the user study we conducted with experienced programmers demonstrate that on a subset of the tasks, TANGO can reduce the time they spend in finding duplicate video-based bug reports by $\approx 65\%$.

In summary, the main contributions of this paper are:

1) TANGO, a duplicate detection approach for video-based bug reports of mobile apps which is able to accurately suggest duplicate reports;
2) The results of a comprehensive empirical evaluation that measures the *effectiveness* of TANGO in terms of suggesting candidate duplicate reports;
3) The results of a user study with experienced programmers that illustrates TANGO's practical applicability by measuring its potential for *saving developer effort*; and
4) A benchmark (included in our online appendix [42]) that enables (i) future research on video-based duplicate detection, bug replication, and mobile app testing, and (ii) the replicability of this work. The benchmark contains 180 video-based bug reports with duplicates, source code, trained models, duplicate detection tasks, TANGO's output, and detailed evaluation results.

II. THE TANGO APPROACH

TANGO (deTecting duplicAte screeN recordinGs of sOftware bugs) is an automated approach based on CV and TR techniques, which leverages visual and textual information to detect duplicate video-based bug reports.

A. TANGO *Overview*

TANGO models duplicate bug report detection as an information retrieval problem. Given a new video-based bug report, TANGO computes a similarity score between the new video and videos previously submitted by app users in a bug tracking system. The new video represents the query and the set of existing videos represent the corpus. TANGO sorts the corpus of videos in decreasing order by similarity score and returns a ranked list of candidate videos. In the list, those videos which are more likely to show the same bug as the new video are ranked higher than those that show a different bug.

TANGO has two major components, which we refer to as TANGO$_{vis}$ and TANGO$_{txt}$ (Fig. 1), that compute video similarity scores independently. TANGO$_{vis}$ computes the *visual similarity* and TANGO$_{txt}$ computes the *textual similarity* between videos. The resulting similarity scores are linearly combined to obtain a final score that indicates the likelihood of two videos being duplicates. In designing TANGO$_{vis}$, we incorporated support for two methods of computing visual similarity — one of which is sensitive to the *sequential order* of visual data, and the other one that is not — and we evaluate the effectiveness of these two techniques in experiments described in Sec. III-IV.

The first step in TANGO's processing pipeline (Fig. 1) is to decompose the query video, and videos from the existing corpus, into their constituent frames using a given sampling rate (*i.e.,* 1 and 5 frames per second - fps). Then, the TANGO$_{vis}$ and TANGO$_{txt}$ components of the approach are executed in parallel. The *un-ordered* TANGO$_{vis}$ pipeline is shown at the top of Fig. 1, comprising steps V_1-V_3; the *ordered* TANGO$_{vis}$ pipeline is illustrated in the middle of Fig. 1, comprising steps V_1, V_4, and V_5; and finally, the TANGO$_{txt}$ pipeline is illustrated at the bottom of Fig. 1 through steps T_1-T_3. Any of these three pipelines can be used to compute the video ranking independently or in combination (*i.e.,* combining the two TANGO$_{vis}$ together, one TANGO$_{vis}$ pipeline with TANGO$_{txt}$, which we call TANGO$_{comb}$, or all three – see Sec. III-C). Next, we discuss these three pipelines in detail.

B. *TANGO$_{vis}$: Measuring Unordered Visual Video Similarity*

The *unordered* version of TANGO$_{vis}$ computes the visual similarity (S_{vis}) of video-based bug reports by extracting visual features from video frames and converting these features into a vector-based representation for a video using a Bag-of-Visual-Words (BoVW) approach [58, 92]. This process is depicted in the top of Fig. 1. The visual features are extracted by the *visual feature extractor* model (V_1 in Fig. 1). Then, the *visual indexer* V_2 assigns to each frame feature vector a

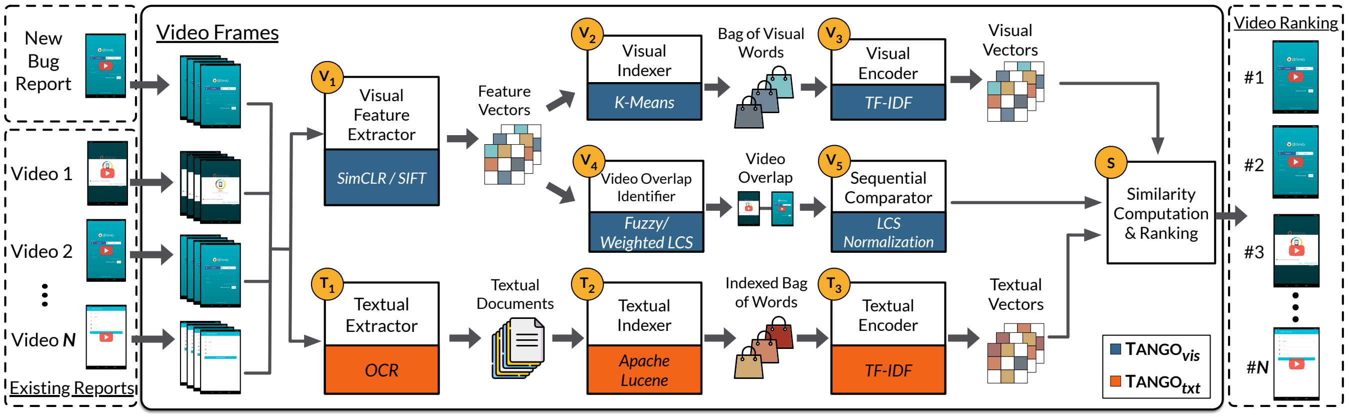

Fig. 1. The TANGO approach for detecting duplicate video-based bug reports.

visual word from a visual codebook and produces a BoVW for a video. The *visual encoder* $\textcircled{V_5}$, based on the video BoVW, encodes the videos using a TF-IDF representation that can be used for similarity computation.

1) Visual Feature Extraction: The *visual feature extractor* $\textcircled{V_1}$ can either use the SIFT [75] algorithm to extract features, or SimCLR [40], a recently proposed Deep Learning (DL) model capable of learning visual representations in an unsupervised, contrastive manner. TANGO's implementation of SimCLR is adapted to extract visual features from app videos.

The first method by which TANGO can extract visual features is using the Scale-Invariant Feature Transform (SIFT) [75] algorithm. SIFT is a state-of-the-art model for extracting local features from images that are invariant to scale and orientation. These features can be matched across images for detecting similar objects. This matching ability makes SIFT promising for generating features that can help locate duplicate images (in our case, duplicate video frames) by aggregating the extracted features. TANGO's implementation of SIFT does not resize images and uses the top-10 features that are the most invariant to changes and are based on the local contrast of neighboring pixels, with higher contrast usually meaning more invariant. This is done to reduce the number of SIFT features, which could reach at least three orders of magnitude for a single frame, and make the *visual indexing* $\textcircled{V_2}$ (through k-Means – see Sec. II-B2) computationally feasible.

The other technique that TANGO can use to extract features is SimCLR. In essence, the goal of this technique is to generate robust visual features that cluster similar images together while maximizing the distance between dissimilar images in an abstract feature space. This is accomplished by (i) generating sets of image pairs (containing one original image and one augmented image) and applying a variety of random augmentations (*i.e.,* image cropping, horizontal flipping, color jittering, and gray-scaling); (ii) encoding this set of image pairs using a base encoder, typically a variation of a convolutional neural network (CNN); and (iii) training a multi-layer-perceptron (MLP) to produce feature vectors that increase the cosine similarity between each pair of image variants and decrease the cosine similarity between negative examples, where negative examples for a given image pair are represented as all other images not in that pair, for a given training batch. TANGO's implementation of SimCLR employs the ResNet50 [53] CNN architecture as the base encoder, where this architecture has been shown to be effective [40].

To ensure that TANGO's *visual feature extractor* is tailored to the domain of mobile app screenshots, we trained this component on the entire RICO dataset [43], which contains over 66k Android screenshots from over 9k of the most popular apps on Google Play. Our implementation of SimCLR was trained using a batch size of $1,792$ and 100 epochs, the same hyperparameters (*e.g.,* learning rate, weight decay, *etc.*) recommended by Chen *et al.* [40] in the original SimCLR paper, and resized images to 224×224 to ensure consistency with our base ResNet50 architecture. The training process was carried out on an Ubuntu 20.04 server with three NVIDIA T4 Tesla 16GB GPUs. The output of the *feature extractor* for SimCLR is a feature vector (of size 64) for each frame of a given video.

2) Visual Indexing: While the SimCLR or SIFT feature vectors generated by TANGO's *visual feature extractor* $\textcircled{V_1}$ could be used to directly compute the similarity between video frames, recent work has suggested that a BoVW approach combined with a TF-IDF similarity measure is more adept to the task of video retrieval [66]. Therefore, to transform the SimCLR or SIFT feature vectors into a BoVW representation, TANGO uses a *visual indexing process* $\textcircled{V_2}$. This process produces an artifact known as a Codebook that maps SimCLR or SIFT feature vectors to "visual words" — which are discrete representations of a given image, and have been shown to be suitable for image and video recognition tasks [66]. The Codebook derives these visual words by clustering feature vectors and computing the centroids of these clusters, wherein each centroid corresponds to a different visual word.

The Codebook makes use of the k-Means clustering algorithm, where the k represents the diversity of the visual words, and thus can affect the representative power of this indexing process. TANGO's implementation of this process is configurable to 1k, 5k, or 10k for the k number of clusters (*i.e.,* the number of visual words - VW). 1k VW and 10k VW were selected as recommended by Kordopatis-Zilos *et al.* [66] and we included 5k VW as a "middle ground" to better understand how the number of visual words impacts TANGO's performance. A Codebook is generated only once for a given k, however, it must be trained before it can be applied to convert an input feature vector to its corresponding visual word(s). Once trained, a Codebook can then be used to map visual words from frame feature vectors without any further modification. Thus, we trained TANGO's six Codebooks, three for SIFT and three for SimCLR, using features extracted from $15,000$ randomly selected images from the RICO dataset [43]. We did not use the entire RICO dataset due to computational constraints of the k-means algorithm.

After the feature vector for a video frame is passed through the *visual indexing* process, it is mapped to its BoVW representation by a trained Codebook. To do this, the Codebook selects the closest centroid to each visual feature vector, based on Euclidean distance. For SIFT, this process may generate more than one feature vector for a single frame, due to the presence of multiple SIFT feature descriptors. In this case, TANGO assigns multiple visual words to each frame. For SimCLR, TANGO assigns one visual word to each video frame, as SimCLR generates only one vector per frame.

3) Visual Encoding: After the video is represented as a BoVW, the *visual encoder* $\textcircled{V_3}$ computes the final vector representation of the video through a TF-IDF-based approach [91]. The term frequency (TF) is computed as the number of visual words occurrences in the BoVW representation of the video, and the inverse document frequency (IDF) is computed as the number of occurrences of a given visual word in the BoVW representations built from the RICO dataset. Since RICO does not provide videos but individual app screenshots, we consider each RICO image as one document. We opted to use RICO to compute our IDF representation for two reasons: (i) to combat the potentially small number of frames present in a given video recording, and (ii) to bolster the generalizability of our similarity measure across a diverse set of apps.

4) Similarity Computation: Given two videos, TANGO$_{vis}$ encodes them into their BoVW representations, and each video is represented as one visual TF-IDF vector. These vectors are compared using cosine similarity, which is taken as the *visual similarity* $\textcircled{S}$ of the videos ($S_{vis} = S_{BoVW}$).

C. TANGO$_{vis}$: Measuring Ordered Visual Video Similarity

The *ordered* version of TANGO$_{vis}$ considers the sequence of video frames when comparing two videos and is capable of giving more weight to common frames nearer the end of the videos, as this is likely where buggy behavior manifests. To accomplish this, the *feature vector extractor* $\textcircled{V_1}$ is used to derive descriptive vectors from each video frame using either SimCLR or SIFT. TANGO determines how much the two videos overlap using an adapted longest common substring (LCS) algorithm $\textcircled{V_4}$. Finally, during the *sequential comparison process* $\textcircled{V_5}$, TANGO calculates the similarity score by normalizing the computed LCS score.

1) Video Overlap Identification: In order to account for the sequential ordering of videos, TANGO employs two different versions of the longest common substring (LCS) algorithm. The first version, which we call fuzzy-LCS (f-LCS), modifies the comparison operator of the LCS algorithm to perform fuzzy matching instead of exact matching between frames in two videos. This fuzzy matching is done differently for SimCLR and SIFT-derived features. For SimCLR, given that each frame is associated with only a single visual word, the standard BoVW vector would be too sparse for a meaningful comparison. Therefore, we compare the feature vectors that SimCLR extracts from the two frames *directly* using cosine similarity. For SIFT, we utilize the BoVW vectors derived by the *visual encoder* $\textcircled{V_3}$, but at a per-frame level.

The second LCS version, which we call weighted-LCS (w-LCS), uses the same fuzzy matching that f-LCS performs. However, the similarity produced in this matching is then weighted depending on where the two frames from each video appeared. Frames that appear later in the video are weighted more heavily, since that is where the buggy behavior is typically occurring in a video-based bug report, and thus should be given more weight for duplicate detection. The exact weighting scheme used is $\frac{i}{m} \times \frac{j}{m}$, where i is the ith frame of video A, m is the # of frames in video A, j is the jth frame of video B, and n is the # of frames in video B.

2) Sequential Comparison: In order to incorporate the LCS overlap measurements into TANGO's overall video similarity calculation, the overlap scores must be normalized between zero and one ($[0, 1]$). To accomplish this, we consider the case where two videos overlap perfectly to be the upper bound of the possible LCS score between two videos, and use this to perform the normalization. For f-LCS, this is done by simply dividing by the # of frames in the smaller video since the max possible overlap that could occur is when the smaller video is a subsection in the bigger video, calculated as $overlap/min$ where $overlap$ denotes the amount the two videos share in terms of their frames and min denotes the # of frames in the smaller of the two videos. For w-LCS, if the videos are different lengths, we align the end of the shorter video to the *end* of the longer video and consider this the upper bound on the LCS score, which is normalized as follows:

$$S_{w-LCS} = \frac{overlap}{\sum_{i=min}^{1} \frac{i}{min} \times \frac{max-i}{max}} \tag{1}$$

where S_{w-LCS} is the normalized similarity value produced by w-LCS, $overlap$ and min are similar to the f-LCS calculation and max denotes the # of frames in the longer of the two videos. The denominator in Eq. 1 calculates the maximum possible overlap that can occur if the videos were exact matches, summing across the similarity score of each frame pair. Our online appendix contains the detailed f/w-LCS algorithms with examples [42].

3) Similarity Computation: f-LCS and w-LCS output the *visual similarity* $\textcircled{S}$ score S_{f-LCS} and S_{w-LCS}, respectively. This can be combined with S_{BoVW} to obtain an aggregate visual similarity score: $S_{vis} = (S_{BoVW} + S_{f-LCS})/2$ or $S_{vis} = (S_{BoVW} + S_{w-LCS})/2$. We denote these TANGO$_{vis}$ variations as B+f-LCS and B+w-LCS, respectively.

D. Determining the Textual Similarity between Videos

In order to determine the textual similarity between video-based bug reports, Tango leverages the textual information from labels, titles, messages, *etc.* found in the app GUI components and screens depicted in the videos.

Tango$_{txt}$ adopts a standard text retrieval approach based on Lucene [51] and Optical Character Recognition (OCR) [1, 16] to compute the textual similarity (S_{txt}) between video-based bug reports. First, a textual document is built from each video in the issue tracker ($\textcircled{T_1}$ in Fig. 1) using OCR to extract text from the video frames. The textual documents are pre-processed using standard techniques to improve similarity computation, namely tokenization, lemmatization, and removal of punctuation, numbers, special characters, and one- and two-character words. The pre-processed documents are indexed for fast similarity computation $\textcircled{T_2}$. Each document is then represented as a vector using TF-IDF and the index [91] $\textcircled{T_3}$.

In order to build the textual documents from the videos, Tango$_{txt}$ applies OCR on the video frames through the Tesseract engine [1, 16] in the *textual extractor* $\textcircled{T_1}$. We experiment with three strategies to compose the textual documents using the extracted frame text. The first strategy (all-text) concatenates all the text extracted from the frames. The second strategy (unique-frames) concatenates all the text extracted from unique video frames, determined by applying exact text matching (before text pre-processing). The third strategy (unique-words) concatenates the unique words in the frames (after pre-processing).

1) Similarity Computation: Tango computes the *textual similarity* (S_{txt}) in $\textcircled{S}$ using Lucene's scoring function [14] based on cosine similarity and document length normalization.

E. Combining Visual and Textual Similarities

Tango combines both the visual (S_{vis}) and textual (S_{txt}) similarity scores produced by Tango$_{vis}$ and Tango$_{txt}$, respectively ($\textcircled{S}$ in Fig. 1). Tango uses a linear combination approach to produce an aggregate similarity value:

$$S_{comb} = (1 - w) \times S_{vis} + w \times S_{txt} \tag{2}$$

where w is a weight for S_{vis} and S_{txt}, and takes a value between zero (0) and one (1). Smaller w values weight S_{vis} more heavily, and larger values weight S_{txt} more heavily. We denote this approach as Tango$_{comb}$.

Based on the combined similarity, Tango generates a ranked list of the video-based bug reports found in the issue tracker. This list is then inspected by the developer to determine if a new video reports a previously reported bug.

III. Tango's Empirical Evaluation Design

We empirically evaluated Tango with two goals in mind: (i) determining how effective Tango is at detecting duplicate video-based bug reports, when considering different configurations of components and parameters, and (ii) estimating the effort that Tango can save developers during duplicate video bug detection. Based on these goals, we defined the following research questions (RQs):

RQ$_1$: *How effective is* Tango *when using either visual or textual information alone to retrieve duplicate videos?*

RQ$_2$: *What is the impact of combining frame sequence and visual information on* Tango*'s detection performance?*

RQ$_3$: *How effective is* Tango *when combining both visual and textual information for detecting duplicate videos?*

RQ$_4$: *How much effort can* Tango *save developers in finding duplicate video-based bug reports?*

To answer our **RQs**, we first collected video-based bug reports for six Android apps (Sec. III-A), and based on them, defined a set of duplicate detection tasks (Sec. III-B). We instantiated different configurations of Tango by combining its components and parameters (Sec. III-C), and executed these configurations on the defined tasks (Sec. III-D). Based on standard metrics, applied on the video rankings that Tango produces, we measured Tango's effectiveness (Sec. III-D). We answer **RQ**$_1$, **RQ**$_2$, and **RQ**$_3$ based on the collected measurements. To answer **RQ**$_4$ (Sec. III-E), we conducted a user study where we measured the time humans take to find duplicates for a subset of the defined tasks, and estimated the time Tango can save for developers. We present and discuss the evaluation results in Sec. IV.

A. Data Collection

We collected video-based bug reports for six open-source Android apps, namely AntennaPod (APOD) [2], Time Tracker (TIME) [6], Token (TOK) [7], GNUCash (GNU) [4], Grow-Tracker (GROW) [5], and Droid Weight (DROID) [3]. We selected these apps because they have been used in previous studies [25, 34, 78, 79], support different app categories (finance, productivity, *etc.*), and provide features that involve a variety of GUI interactions (taps, long taps, swipes, *etc.*). Additionally, none of these apps are included as part of the RICO dataset used to train Tango's SimCLR model and Codebooks, preventing the possibility of data snooping. Since video-based bug reports are not readily available in these apps' issue trackers, we designed and carried out a systematic procedure for collecting them.

In total, we collected 180 videos that display 60 distinct bugs – 10 bugs for each app and three videos per bug (*i.e.,* three duplicate videos per bug). From the 60 bugs, five bugs (one bug per app except for DROID) are reported in the apps' issue trackers. These five bugs were selected because they were the only ones in the issue trackers that we were able to reproduce based on the provided bug descriptions. During the reproduction process, we discovered five *additional* new bugs in the apps not reported in the issue trackers (one bug each for APOD, GNU, and TOK, and two bugs for TIME) for a total of 10 confirmed real bugs.

The remaining 50 bugs were introduced in the apps through mutation by executing MutAPK [46], a mutation testing tool that injects bugs (*i.e.,* mutants) into Android APK binaries via a set of 35 mutation operators that were derived from a large-scale empirical study on *real* Android application faults. Given the empirically-derived nature of these operators, they were shown to accurately simulate real-world Android faults [46]. We applied MutAPK to the APKs of all six apps. Then, from the mutant list produced by the tool, we randomly selected 7 to 10 bugs for each app, and ensured that they could be reproduced and manifested in the GUI. To diversify the bug pool, we selected the bugs from multiple mutant operators and ensured that they affected multiple app features/screens.

When selecting the 60 bugs, we ensured they manifest graphically and were reproducible by manually replicating them on a specific Android emulator configuration (virtual Nexus 5X with Android 7.0 configured via Android Studio). For all the bugs, we screen-recorded the bug and the reproduction scenario. We also generated a textual bug report (for bugs that did not have one) containing the description of the unexpected and expected app behavior and the steps to reproduce the bug.

To generate the remaining 120 video-based bug reports, we asked two professional software engineers and eight computer science (CS) Ph.D. students to replicate and record the bugs (using the same Android emulator), based only on the textual description of the unexpected and expected app behavior. The participants have between 2 and 10 years of programming experience (median of 6 years).

All the textual bug reports given to the study participants contained *only* a brief description of the observed and expected app behavior, with *no specific reproduction steps*. We opted to perform the collection in this manner to ensure the robustness of our evaluation dataset by maximizing the diversity of video-based reproduction steps, and simulating a real-world scenario where users are aware of the observed (incorrect) and expected app behavior, and must produce the reproduction steps themselves.

We randomly assigned the bugs to the participants in such a way that each bug was reproduced and recorded by two participants, and no participant recorded the same bug twice. Before reproducing the bugs, the participants spent five minutes exploring the apps to become familiar with their functionality. Since some of the participants could not reproduce the bugs after multiple attempts (mainly due to bug misunderstandings) and some of the videos were incorrectly recorded (due to mistakes), we reassigned these bugs among the other participants, who were able to reproduce and record them successfully.

Our bug dataset consists of 35 crashes and 25 non-crashes, and include a total of 470 steps (397 taps, 12 long taps, 14 swipes, among other types), with an average of 7.8 steps per video. The average video length is ≈ 28 seconds.

B. *Duplicate Detection Tasks*

For each app, we defined a set of tasks that mimic a realistic scenario for duplicate detection. Each duplicate detection task is composed of a new video (*i.e.,* the new bug report, *aka* the query) and a set of existing videos (*i.e.,* existing bug reports in the issue tracker, *aka* the corpus). In practice, a developer would determine if the new video is a duplicate by inspecting the corpus of videos in the order given by TANGO (or any other approach). For our task setup, the corpus contains both duplicate and non-duplicate videos. There are two different types of duplicate videos that exist in the corpus: (i) those videos that are a duplicate of the query (the *Same Bug* group), and (ii) those videos which are duplicates of each other, but are not a duplicate of the query (the *Different Bug* group). This second type of duplicate video is represented by bug reports marked as duplicates in the issue tracker and their corresponding master reports [35, 86, 93]. Each non-duplicate video reports a distinct bug.

We constructed the duplicate detection tasks on a *per app* basis, using the 30 video reports collected for each app (*i.e.,* three video reports for each of the 10 bugs, for a total of 30 video reports per app). We first divided all the 30 videos for an app into three groups, each group containing 10 videos (one for each bug) created by one or more participants. Then, we randomly selected a video from one bug as the query and took the other two videos that reproduce the same bug as the *Same Bug* duplicate group (*i.e.,* the ground truth). Then, we selected one of the remaining nine bugs and added its three videos to the *Different Bug* duplicate group. Finally, we selected one video from the remaining eight bugs, and used these as the corpus' *Non-Duplicate* group. This resulted in a total of 14 distinct bug reports per task (two in the *Same Bug* group, three in the *Different Bug* group, eight in the *Non-Duplicate* group, and the query video). After creating tasks based on all the combinations of query and corpus, we generated a total of 810 duplicate detection tasks per app or $4,860$ aggregating across all apps.

We designed the duplicate detection setting described above to mimic a scenario likely to be encountered in crowd-sourced app testing, where duplicates of the query, other duplicates not associated with the query, and other videos reporting unique bugs, exist in a shared corpus for a given app. While there are different potential task settings, we opted not to vary this experimental variable in order to make for a feasible analysis that allowed us to explore more thoroughly the different TANGO configurations.

C. TANGO *Configurations*

We designed TANGO$_{vis}$ and TANGO$_{txt}$ to have different configurations. TANGO$_{vis}$'s configurations are based on different visual feature extractors (SIFT or SimCLR), video sampling rates (1 and 5 fps), # of visual words (1k, 5k, and 10k VW), and approaches to compute video similarity (BoVW, f-LCS, w-LCS, B+f-LCS, and B+w-LCS). TANGO$_{txt}$'s configurations are based on the same sampling rates (1 and 5 fps) and the approaches to extract the text from the videos (all-text, unique-frames, and unique-words). TANGO$_{comb}$ combines TANGO$_{vis}$ and TANGO$_{txt}$ as described in Sec. II-E.

D. TANGO*'s Execution and Effectiveness Measurement*

We executed each TANGO configuration on the $4,860$ duplicate detection tasks and measured its effectiveness using standard metrics used in prior text-based duplicate bug detection research [35, 86, 93]. For each task, we compare the ranked list of videos produced by TANGO and the expected duplicate videos from the ground truth.

We measured the *rank* of the first duplicate video found in the ranked list, which serves as a proxy for how many videos the developer has to watch in order to find a duplicate video. A smaller *rank* means higher duplicate detection effectiveness. Based on the *rank*, we computed the *reciprocal rank* metric: $1/rank$. We also computed the *average precision* of TANGO, which is the average of the precision values achieved at all the cutting points k of the ranked list (*i.e.,* precision@k). Precision@k is the proportion of the top-k returned videos that are duplicates according to the ground truth. We also computed *HIT@k* (*aka* Recall Rate@k [35, 86, 93]), which is the proportion of tasks that are successful for the cut point k of the ranked list. A task is successful if at least one duplicate video is found in the top-k results returned by TANGO. We report HIT@k for cut points k = 1-2 in this paper, and 1-10 in our online appendix [42].

Additionally, we computed the average of these metrics over sets of duplicate detection tasks: mean reciprocal rank (mRR), mean average precision (mAP), and mean rank (μ rank or μRk) per app and across all apps. Higher mRR, mAP, and HIT@k values indicate higher duplicate detection effectiveness. These metrics measure the overall performance of a duplicate detector.

We focused on comparing mRR values to decide if one TANGO configuration is more effective than another, as we consider that it better reflects the usage scenario of TANGO. In practice, the developer would likely stop inspecting the suggested duplicates (given by TANGO) when she finds the first correct duplicate. This scenario is captured by mRR, through the *rank* metric, which considers only the first correct duplicate video as opposed to the entire set of duplicate videos to the query (as mAP does).

E. Investigating TANGO*'s Effort Saving Capabilities*

We conducted a user study in order to estimate the effort that developers would spend while manually finding video-based duplicates. This effort is then compared to the effort measurements of the best TANGO configuration, based on μ *rank* and *HIT@k*. This study and the data collection procedure were conducted remotely due to COVID-19 constraints.

1) Participants and Tasks: One professional software engineer and four CS Ph.D. students from the data collection procedure described in Sec. III-A participated in this study. The study focused on APOD, the app that all the participants had in common from the data collection. We randomly selected 20 duplicate detection tasks, covering all 10 APOD bugs.

2) Methodology: Each of the 20 tasks was completed by two participants. Each participant completed four tasks, each task's query video reporting a unique bug. The assignment of the tasks to the participants was done randomly. For each task, the participants had to watch the new video (the query) and then find the videos in the corpus that showed the same bug of the new video (*i.e.,* find the duplicate videos). All the videos were anonymized so that the duplicate videos were unknown to the participants. To do this, we named each video with a number that represents the video order and the suffix "vid" (*e.g.,* "2_vid.mp4").

The corpus videos were given in random order and the participants could watch them in any order. To make the bug of the new video clearer to the participants, we provided them with the description of the unexpected and expected app behavior, taken from the textual bug reports that we generated for the bugs. We consider the randomization of the videos as a reasonable baseline given that other baselines (*e.g.,* video-based duplicate detectors) do not currently exist and the video-based bug reports in our dataset do not have timestamps (which can be used to give a different order to the videos). This is a threat to validity that we discuss in Sec. V.

3) Collected Measurements: Through a survey, we asked each participant to provide the following information for each task: (i) the name of the first video they deemed a duplicate of the query, (ii) the time they spent to find this video, (iii) the number of videos they had to watch until finding the first duplicate (including the duplicate), (iv) the names of other videos they deemed duplicates, and (v) the time they spent to find these additional duplicates. We instructed the participants to perform the tasks without any interruptions in order to minimize inaccuracies in the time measurements.

4) Comparing TANGO *and Manual Duplicate Detection:* The collected measurements from the participants were compared against the effectiveness obtained by executing the best TANGO configuration on the 20 tasks, in terms of μ *rank* and *HIT@k*. We compared the avg. number of videos the participants watched to find one duplicate against the avg. number of videos they would have watched had they used TANGO.

IV. TANGO'S EVALUATION RESULTS

A. RQ1: Using Only Visual or Textual Information

We analyzed the performance of TANGO when using only visual or textual information exclusively. In this section, we present the results for TANGO's best performing configurations. However, complete results can be found in our online appendix [42]. Table I shows the results for TANGO$_{vis}$ and TANGO$_{txt}$ when using SimCLR, SIFT, as the *visual feature extractor*, and OCR as the *textual extractor*. For simplicity, we use SimCLR, SIFT, and OCR&IR to refer to SimCLR-based TANGO$_{vis}$, SIFT-based TANGO$_{vis}$, and TANGO$_{txt}$, respectively. The best results for each metric are illustrated in bold on a per app basis. The results provided in Table I are those for the best parameters of the SimCLR, SIFT, and OCR&IR feature extractors, which are (BoVW, 5 fps, 1k VW), (w-LCS, 1 fps, 10k VW), and (all-text, 5 fps), respectively.

Table I shows that TANGO$_{vis}$ is more effective when using SimCLR rather than SIFT across all the apps, achieving an overall mRR, mAP, avg. rank, HIT@1, and HIT@2 of 75.3%, 67.8%, 1.9, 61.6%, and 78%, respectively. SimCLR is also superior to OCR&IR overall, whereas SIFT performs least effectively of the three approaches. When analyzing the results per app, we observe that SimCLR is outperformed by OCR&IR (by 0.7% - 4% difference in mRR) for APOD,

TABLE I
EFFECTIVENESS FOR THE BEST TANGO CONFIGURATIONS THAT USE EITHER VISUAL (SIMCLR/SIFT) OR TEXTUAL (OCR&IR) INFORMATION.

App	Config.	mRR	mAP	μRk	HIT@1	HIT@2
APOD	SIFT	64.6%	51.1%	3.0	47.7%	71.7%
	SimCLR	80.0%	66.8%	1.7	**68.1%**	82.6%
	OCR&IR	**80.8%**	**75.3%**	**1.5**	65.7%	**88.6%**
DROID	SIFT	66.3%	55.0%	2.5	49.1%	69.5%
	SimCLR	64.6%	59.2%	2.6	49.5%	61.7%
	OCR&IR	**67.9%**	**64.7%**	**2.3**	**52.0%**	**69.8%**
GNU	SIFT	66.1%	57.2%	2.2	47.4%	68.4%
	SimCLR	81.8%	75.1%	1.6	70.1%	85.3%
	OCR&IR	**84.5%**	**82.3%**	**1.4**	**72.2%**	**92.0%**
GROW	SIFT	56.0%	49.9%	3.0	36.5%	54.3%
	SimCLR	72.7%	68.8%	2.0	57.4%	75.6%
	OCR&IR	**76.8%**	**69.0%**	**1.9**	**63.6%**	**80.1%**
TIME	SIFT	49.2%	40.7%	3.3	26.7%	46.4%
	SimCLR	**74.8%**	**67.6%**	**2.3**	**63.7%**	**75.9%**
	OCR&IR	47.4%	37.7%	4.0	28.3%	44.4%
TOK	SIFT	39.0%	32.1%	4.4	17.0%	33.7%
	SimCLR	**77.7%**	**69.3%**	**1.6**	**60.6%**	**86.7%**
	OCR&IR	61.3%	53.3%	2.6	42.6%	60.7%
Overall	SIFT	56.9%	47.7%	3.1	37.4%	57.3%
	SimCLR	**75.3%**	**67.8%**	**1.9**	**61.6%**	**78.0%**
	OCR&IR	69.8%	63.7%	2.3	54.1%	72.6%

DROID, GNU and GROW; with OCR&IR being the most effective for these apps. SimCLR outperforms the other two approaches for TIME and TOK by more than 16% difference in mRR. The differences explain the overall performance of SimCLR and OCR&IR. SimCLR is more consistent in its performance compared to OCR&IR and SIFT. Across apps, the mRR standard deviation of SimCLR is 6.2%, which is lower than that for SIFT and OCR&IR: 11.1% and 13.9%, respectively. The trend is similar for mAP and avg. rank.

Since the least consistent approach across apps is TANGO$_{txt}$ in terms of effectiveness, we investigated the root causes for its lower performance on TIME and TOK. After manually watching a subset of the videos for these apps, we found that their textual content was quite similar across bugs. Based on this, we hypothesized that the amount of vocabulary shared between duplicate videos (from the same bugs) and non-duplicate videos (across different bugs) affected the discriminatory power of Lucene-based TANGO$_{txt}$ (see Sec. II-D).

To verify this hypothesis, we measured the shared vocabulary of duplicate and non-duplicate video pairs, similarly to Chaparro *et al.*'s analysis of textual bug reports [35]. We formed unique pairs of duplicate and non-duplicate videos from all the videos collected for all six apps. For each app, we formed 30 duplicate and 405 non-duplicate pairs, and we measured the avg. amount of shared vocabulary of all pairs, using the vocabulary agreement metric used by Chaparro *et al.* [35]. Table II shows the vocabulary agreement of duplicate (V_d) and non-duplicate pairs (V_{nd}) as well as the mRR and mAP values of TANGO$_{txt}$ for each app. The table reveals that the vocabulary agreement of duplicates and non-duplicates is very similar for TIME and TOK, and dissimilar for the other apps. The absolute difference between these measurements

TABLE II
VOCABULARY AGREEMENT & EFFECTIVENESS FOR THE BEST TANGO$_{txt}$.

App	Vocabulary agreement			mRR	mAP
	V_d	V_{nd}	$\lvert V_d - V_{nd}\rvert$		
APOD	70.8%	37.9%	32.9%	80.8%	75.3%
DROID	73.9%	57.0%	16.9%	67.9%	64.7%
GNU	82.2%	58.6%	23.6%	84.5%	82.3%
GROW	67.0%	41.7%	25.4%	76.8%	69.0%
TIME	86.0%	86.3%	0.3%	47.4%	37.7%
TOK	69.6%	61.0%	8.6%	61.3%	53.3%
Overall	74.2%	56.7%	17.5%	69.8%	63.7%

(*i.e.,* $|V_d - V_{nd}|$) for TIME and TOK is 0.3% and 8.6%, while for the other apps it is above 16%. We found 0.94 / 0.91 Pearson correlation [47] between these differences and the mRR/mAP values.

The results indicate that, for TIME and TOK, the similar vocabulary between duplicate and non-duplicate videos negatively affects the discriminatory power of TANGO$_{txt}$, which suggests that for some apps, using only textual information may be sub-optimal for duplicate detection.

Answer for RQ$_1$: SimCLR performs the best overall with an mRR and HIT@1 of 75.3% and 61.6%, respectively. For 4 of 6 apps, OCR&IR outperforms SimCLR by a significant margin. However, due to issues with vocabulary overlap, it performs worse overall. SIFT is the worst-performing technique across all the apps.

B. RQ2: Combining Visual and Frame Sequence Information

To answer **RQ**$_2$, we compared the effectiveness of the best configuration of TANGO when using visual information alone (SimCLR, BoVW, 5fps, 1k VW) and when combining visual & frame sequence information (*i.e.,* B+f-LCS and B+w-LCS).

The results are shown in Table III. Overall, using TANGO with BoVW alone is more effective than combining the approaches; TANGO based on BoVW achieves 75.3%, 67.8%, 1.9, 61.6%, and 78% mRR, mAP, avg. rank, HIT@1, and HIT@2, respectively. Using BoVW and w-LCS combined is the least effective approach. BoVW alone and B+f-LCS are comparable in performance. However, BoVW is more consistent in its performance across apps: 6.2% mRR std. deviation vs. 6.6% and 9.2% for B+f-LCS and B+w-LCS.

The per-app results reveal that B+w-LCS consistently is the least effective approach for all apps except for GROW, for which B+w-LCS performs best. After watching the videos for GROW, we found unnecessary steps in the beginning/middle of the duplicate videos, which led to their endings being weighted more heavily by w-LCS, where steps were similar. In contrast, BoVW and B+f-LCS give a lower weight to these cases thus reducing the overall video similarity.

The lower performance of B+f-LCS and B+w-LCS, compared to BoVW, is partially explained by the fact that f-LCS and w-LCS are more restrictive by definition. Since they find the longest common sub-strings of frames between videos,

TABLE III
EFFECTIVENESS FOR THE BEST TANGO$_{vis}$ CONFIGURATION USING EITHER VISUAL INFORMATION (BOVW) OR A COMBINATION OF VISUAL AND FRAME SEQUENCE INFORMATION (B+F-LCS AND B+W-LCS).

App	Config.	mRR	mAP	μRk	HIT@1	HIT@2
APOD	B+f-LCS	79.3%	**67.8%**	**1.7**	66.2%	82.3%
	B+w-LCS	77.2%	65.5%	1.9	64.2%	80.1%
	BoVW	**80.0%**	66.8%	**1.7**	**68.1%**	**82.6%**
DROID	B+f-LCS	**64.8%**	**60.7%**	**2.6**	**50.2%**	61.6%
	B+w-LCS	63.7%	54.8%	2.7	48.9%	62.3%
	BoVW	64.6%	59.2%	**2.6**	49.5%	**61.7%**
GNU	B+f-LCS	**83.3%**	**75.6%**	**1.6**	**73.2%**	**85.6%**
	B+w-LCS	77.3%	65.7%	1.8	62.3%	83.6%
	BoVW	81.8%	75.1%	**1.6**	70.1%	85.3%
GROW	B+f-LCS	76.0%	70.2%	2.0	64.2%	75.2%
	B+w-LCS	**81.3%**	**75.0%**	**1.7**	**70.9%**	**82.8%**
	BoVW	72.7%	68.8%	2.0	57.4%	75.6%
TIME	B+f-LCS	70.4%	63.4%	**2.3**	54.4%	74.3%
	B+w-LCS	63.8%	58.5%	2.8	48.0%	64.9%
	BoVW	**74.8%**	**67.6%**	**2.3**	**63.7%**	**75.9%**
TOK	B+f-LCS	73.4%	65.6%	1.7	54.0%	82.5%
	B+w-LCS	59.2%	53.7%	2.6	37.9%	60.0%
	BoVW	**77.7%**	**69.3%**	**1.6**	**60.6%**	**86.7%**
Overall	B+f-LCS	74.5%	67.2%	2.0	60.4%	76.9%
	B+w-LCS	70.4%	62.2%	2.2	55.4%	72.3%
	BoVW	**75.3%**	**67.8%**	**1.9**	**61.6%**	**78.0%**

TABLE IV
EFFECTIVENESS OF THE BEST TANGO$_{comb}$, TANGO$_{vis}$, AND TANGO$_{txt}$.

App	Config.	mRR	mAP	μRk	HIT@1	HIT@2
APOD	TANGO$_{comb}$	**84.4%**	**75.8%**	**1.4**	**73.1%**	87.9%
	TANGO$_{vis}$	80.0%	66.8%	1.7	68.1%	82.6%
	TANGO$_{txt}$	80.8%	75.3%	1.5	65.7%	**88.6%**
DROID	TANGO$_{comb}$	**70.6%**	**66.7%**	**2.2**	**55.9%**	**71.0%**
	TANGO$_{vis}$	64.6%	59.2%	2.6	49.5%	61.7%
	TANGO$_{txt}$	67.9%	64.7%	2.3	52.0%	69.8%
GNU	TANGO$_{comb}$	**89.5%**	**84.7%**	**1.3**	**81.6%**	**94.2%**
	TANGO$_{vis}$	81.8%	75.1%	1.6	70.1%	85.3%
	TANGO$_{txt}$	84.5%	82.3%	1.4	72.2%	92.0%
GROW	TANGO$_{comb}$	**81.7%**	**75.4%**	**1.7**	**71.4%**	**82.5%**
	TANGO$_{vis}$	72.7%	68.8%	2.0	57.4%	75.6%
	TANGO$_{txt}$	76.8%	69.0%	1.9	63.6%	80.1%
TIME	TANGO$_{comb}$	59.6%	51.7%	2.8	40.2%	58.8%
	TANGO$_{vis}$	**74.8%**	**67.6%**	**2.3**	**63.7%**	**75.9%**
	TANGO$_{txt}$	47.4%	37.7%	4.0	28.3%	44.4%
TOK	TANGO$_{comb}$	69.8%	60.8%	2.0	50.9%	76.9%
	TANGO$_{vis}$	**77.7%**	**69.3%**	**1.6**	**60.6%**	**86.7%**
	TANGO$_{txt}$	61.3%	53.3%	2.6	42.6%	60.7%
Overall	TANGO$_{comb}$	**75.9%**	**69.2%**	**1.9**	**62.2%**	**78.5%**
	TANGO$_{vis}$	75.3%	67.8%	**1.9**	61.6%	78.0%
	TANGO$_{txt}$	69.8%	63.7%	2.3	54.1%	72.6%

small variations (*e.g.,* extra steps) in the reproduction steps of the bugs may lead to drastic changes in similarity measurement for these approaches. Also, these approaches only find one common substring (*i.e.,* the longest one), which may not be highly discriminative for duplicate detection. In the future, we plan to explore additional approaches for aligning the frames, for example, by using an approach based on longest common sub-sequence algorithms [49] that can help align multiple portions between videos. Another potential reason for these results may lie in the manner that TANGO combines visual and sequential similarity scores – weighting both equally. In future work, we plan to explore additional combination techniques.

Answer for RQ$_2$: Combining ordered visual information (via f-LCS and w-LCS) with the orderless BoVW improves the results for four of the six apps. However, across all apps, BoVW performs more consistently.

C. RQ3: Combining Visual and Textual Information

We investigated TANGO's effectiveness when combining visual and textual information. We selected the best configurations of TANGO$_{vis}$ (SimCLR, BoVW, 5 fps, 1k VW) and TANGO$_{txt}$ (all-text, 5 fps) from **RQ$_1$** based on their mRR score and measured its performance overall and per app. We provide the results for the best weight we obtained for TANGO's *similarity computation and ranking* which was $w = 0.2$, *i.e.,* a weight of 0.8 and 0.2 on TANGO$_{vis}$ and TANGO$_{txt}$, respectively. These weights were found by evaluating different w values from zero (0) to one (1) in increments of 0.1 and selecting the one leading to the highest overall mRR score. Complete results can be found in our online appendix [42].

Table IV shows that the overall effectiveness achieved by TANGO$_{comb}$ is higher than that achieved by TANGO$_{txt}$ and TANGO$_{vis}$. TANGO$_{comb}$ achieves 75.9%, 69.2%, 1.9, 62.2%, and 78.5% mRR, mAP, avg. rank, HIT@1, and HIT@2, on average. The avg. improvement margin of TANGO$_{comb}$ is substantially higher for TANGO$_{txt}$ (6.2%/5.5% mRR/mAP) than for TANGO$_{vis}$ (0.7%/1.4% mRR/mAP).

Our analysis of the per-app results explains these differences. Table IV reveals that combining visual and textual information substantially increases the performance over just using one of the information types alone, except for the TIME and TOK apps. This is because TANGO$_{txt}$'s effectiveness is substantially lower for these apps, compared to the visual version (see Table I), due to the aforementioned vocabulary agreement. Thus, incorporating the textual information significantly harms the performance of TANGO$_{comb}$.

1) A Better Combination of Visual and Textual Information: The results indicate that combining visual and textual information is beneficial for most of our studied apps but harmful for a subset (TIME and TOK). This is because the textual information used alone, for TIME and TOK, leads to low performance. The analysis we made for TANGO$_{txt}$ in **RQ$_1$**, revealed that the reason for the low performance of TANGO$_{txt}$ lies in the similar amount of vocabulary overlap between duplicate and non-duplicate videos. Fortunately, based on this amount of vocabulary, we can predict the performance of TANGO$_{txt}$ for new video-based bug reports as follows [35]. In practice, the issue tracker will contain reports marked as duplicates (reporting the same bugs) from previous submissions of bug reports as well as non-duplicates (reporting unique bugs). This information can be used to compute the vocabulary agreement between duplicates and non-duplicates, which can be used to predict how well TANGO$_{txt}$ would perform for new reports.

Based on this, we defined a new approach for TANGO, which is based on the vocabulary agreement metric from [35] applied on existing duplicate and non-duplicate reports. This approach dictates that if the difference of vocabulary agreement between existing duplicates and non-duplicates is greater than a certain threshold, then TANGO should combine visual and textual information. Otherwise, TANGO should only use the visual information because it is likely that the combination would not be better than using the visual information alone.

From the vocabulary agreement measurements shown in Table II, we infer a proper threshold from the new TANGO approach. This threshold may be taken as one value from the interval 8.6% - 16.9% (exclusive) because those are the limits that separate the apps for which TANGO$_{txt}$ obtains low (TIME and TOK) and high performance (APOD, DROID, GNU, and GROW). For practical reasons, we select the threshold to be the middle value: $8.6 + (16.9 - 8.6)/2 = 12.8\%$. In future work, we plan to further evaluate this threshold on other apps.

We implemented this approach for TANGO, using 0.2 as weight, and measured its effectiveness. This approach resulted in a mRR, mAP, avg. rank, HIT@1 and HIT@2 of 79.8%, 73.2%, 1.7, 67.7%, and 83%, respectively. The approach leads to a substantial improvement (*i.e.,* 3.9% / 4.1% higher mRR / mAP) over TANGO$_{comb}$ shown in Table IV.

The results mean that the best version of TANGO is able to suggest correct duplicate video-based bug reports in the first or second position of the returned candidate list for 83% of the duplicate detection tasks.

Answer for RQ$_3$: Combining visual and textual information significantly improves results for 4 of 6 apps. However, due to the vocabulary agreement issue, across all apps, this approach is similar in effectiveness to using visual information alone. Accounting for this vocabulary overlap issue through a selective combination of visual and textual information via a threshold, TANGO achieves the highest effectiveness: an mRR, mAP, avg. rank, HIT@1, and HIT@2 of 79.8%, 73.2%, 1.7, 67.7%, and 83%, respectively.

D. RQ4: Time Saved Discovering Duplicates

As expected, the participants were successful in finding the duplicate videos for all 20 tasks. In only one task, one participant incorrectly flagged a video as duplicate because it was highly similar to the query. Participants found the first duplicate video in 96.4 seconds and watched 4.3 videos on avg. across all tasks to find it. Participants also found all the duplicates in 263.8 seconds on avg. by watching the entire corpus of videos. This means they spent 20.3 seconds in watching one video on average.

We compared these results with the measurements taken from TANGO's best version (*i.e.,* selective TANGO) on the tasks the participants completed. TANGO achieved a 1.5 avg. rank, which means that, by using TANGO, they would only have to watch one or two videos on avg. to find the first duplicate. This would have resulted in $(4.3 - 1.5)/4.3 = 65.1\%$ of the time saved. In other words, instead of spending $20.3 \times 4 = 81.2$ seconds (on avg.) finding a duplicate for a given task, the participants could have spent $20.3 \times 1.5 = 30.5$ seconds. These results indicate the potential of TANGO to help developers save time when finding duplicates.

Answer for RQ$_4$: On average, TANGO's best-performing configuration can save 65.1% of the time participants spend finding duplicate videos.

V. TANGO LIMITATIONS & THREATS TO VALIDITY

Limitations. TANGO has three main limitations that motivate future work.

The first one stems from the finding that textual information may not be beneficial for some apps. The best TANGO version implements an approach for detecting this situation, based on a threshold for the difference in vocabulary overlap between duplicate and non-duplicate videos, which is used for selectively combining visual or textual information. This threshold is based on the collected data and may not generalize to other apps. Second, the visual TF-IDF representation for the videos is based on the mobile app images from the RICO dataset, rather than on the videos found in the tasks' corpus, as it is typically done in text retrieval. Additionally, we considered single images as documents rather than groups of frames that make up a video. These decisions were made to improve the generalization of TANGO's visual features and to support projects that have limited training data. Third, differences in themes and languages across video-based bug reports for an application could have an impact in the performance of TANGO. We believe that different themes (*i.e.,* dark vs. light modes) will not significantly impact TANGO since the SimCLR model is trained to account for such color differences by performing color jittering and gray-scaling augmentations. However, additional experiments are needed to validate this conjecture. For different languages, TANGO currently assumes the text in an application to be English when performing OCR and textual similarity. Therefore, its detection effectiveness where the bug reports display different languages (*e.g.,* English vs. French) could be negatively impacted. We will investigate this aspect in our future work.

Internal & Construct Validity. Most of the mobile app bugs in our dataset were introduced by MutAPK [46], and hence potentially may not resemble real bugs. However, MutAPK's mutation operators were empirically derived from a large-scale study of real Android faults, and prior research lends credence of the ability of mutants to resemble real faults [21]. We intentionally selected generated mutants from a range of operators to increase the diversity of our set of bugs and mitigate potential biases. Another potential threat is related to using real bugs from issue trackers that cannot be reproduced or that do not manifest graphically. We mitigated this threat by using a small, carefully vetted subset of real bugs that were analyzed by multiple authors before being used in our dataset. We did not observe major differences in the results between mutants and real bugs.

Another threat to validity is that our approach to construct the duplicate detection tasks does not take into account bug report timestamps, which would be typical in a realistic scenario [86], and timestamps could be used as a baseline ordering of videos for comparing against the ranking given by TANGO. The lack of timestamps stems from the fact that we were not able to collect the video-based bug reports from existing mobile projects. We mitigated this threat in our user study by randomizing the ordering of the corpus videos given to the participants. We consider this as a reasonable baseline for evaluating our approach considering that, to the best of our knowledge, (1) no existing datasets, with timestamps, are available for conducting research on video-based duplicate detection, and (2) no existing duplicate detectors work exclusively on video-based bug reports, as TANGO does.

External Validity. We selected a diverse set of apps that have different functionality, screens, and visual designs, in an attempt to mitigate threats to external validity. Additionally, our selection of bugs also attempted to select diverse bug types (crashes and non-crashes), and the duplicate videos were recorded by different participants. As previously discussed, there is the potential that TANGO's different parameters & thresholds may not generalize to video data from other apps.

VI. RELATED WORK

Our research is related to work in near duplicate video retrieval, analysis of graphical software artifacts, and duplicate detection of textual bug reports.

Near Duplicate Video Retrieval. Extensive research has been done outside SE in near-duplicate video retrieval, which is the identification of similar videos to a query video (*e.g.,* exact copies [45, 50, 57, 68] or similar events [41, 58, 66, 88]).

The closest work to ours is by Kordopatis-Zilos *et al.* [66], who addressed the problem of retrieving videos of incidents (*e.g.,* accidents). In their work, they explored using handcrafted-based [23, 59, 60, 75, 102, 106] (*e.g.,* SURF or HSV histograms) and DL-based [32, 37, 65, 67, 71, 98] (*e.g.,* CNNs) visual feature extraction techniques and ways of combining the extracted visual features [31, 58, 62, 65, 88, 92] (*e.g.,* VLAD). While we do make use of the best performing model (CNN+BoVW) from this work [66], we did not use the proposed handcrafted approaches, as these were designed for scenes about real-world incidents, rather than for mobile bug reporting. We also further modified and extended this approach given our different domain, through the combination of visual and textual information modalities, and adjustments to the CCN+BoVW model, including the layer configuration and training objective.

Analysis of Graphical Software Artifacts. The analysis of graphical software artifacts to support software engineering tasks has been common in recent years. Such tasks include mobile app testing [25, 56, 61, 79], developer/user behavior modelling [22, 30, 48], GUI reverse engineering and code generation [24, 38, 39, 44, 80, 83], analysis of programming videos [20, 69, 76, 85, 103, 104], and GUI understanding and verification [33, 105]. None of these works deal with finding duplicate video-based bug reports, which is our focus.

Detection of Duplicate Textual Bug Reports. Many research projects have focused on detecting duplicate textual bug reports [28, 29, 35, 36, 52, 54, 55, 64, 70, 72, 74, 82, 84, 86, 87, 89, 90, 93–97, 99–101, 107]. Similar to TANGO, most of the proposed techniques return a ranked list of duplicate candidates [35, 63]. The work most closely related to TANGO is by Wang *et al.* [99], who leveraged attached mobile app images to better detect duplicate textual reports. Visual features are extracted from the images (*e.g.,* representative colors), using computer vision, which are combined with textual features extracted from the text to obtain an aggregate similarity score. While this is similar to our work, TANGO is intended to be applied to videos rather than single images and focuses on video-based bug reports alone, without any extra information such as bug descriptions.

VII. CONCLUSION AND FUTURE WORK

This paper presented TANGO, an approach that combines visual and textual information to help developers find duplicate video-based bug reports. Our empirical evaluation, conducted on $4,680$ duplicate detection tasks created from 180 video-based bug reports from six mobile apps, illustrates that TANGO is able to effectively identify duplicate reports and save developer effort. Specifically, TANGO correctly suggests duplicate video-based bug reports within the top-2 candidate videos for 83% of the tasks, and saves 65.1% of the time that humans spend finding duplicate videos.

Our future work will focus on addressing TANGO's limitations and extending TANGO's evaluation. Specifically, we plan to (1) explore additional ways to address the vocabulary overlap problem, (2) investigate the resilience of TANGO to different app characteristics such as the use of different themes, languages, and screen sizes, (3) extend TANGO for detecting duplicate bug reports that contain multimedia information (text, images, and videos), (4) evaluate TANGO using data from additional apps, and (5) assess the usefulness of TANGO in industrial settings.

VIII. DATA AVAILABILITY

Our online appendix [42] includes the collected video-based bug reports with duplicates, TANGO's source code, trained models, evaluation infrastructure, TANGO's output, and detailed evaluation results.

ACKNOWLEDGEMENTS

We thank Winson Ye for experimenting with different CNN models, the study participants for their time and useful feedback, and the companies who responded to our initial internal survey. This research was supported in part by the NSF CCF-1955853 grant. Any opinions, findings, and conclusions expressed herein are the authors' and do not necessarily reflect those of the sponsors.

References

[1] Tesseract ocr library https://github.com/tesseract-ocr/tesseract/wiki.
[2] Antennapod https://github.com/AntennaPod/AntennaPod, 2019.
[3] Droidweight https://github.com/sspieser/droidweight, 2019.
[4] Gnucash https://github.com/codinguser/gnucash-android, 2019.
[5] Growtracker https://tinyurl.com/yy9oezom, 2019.
[6] Time tracker https://github.com/netmackan/ATimeTracker, 2019.
[7] Token https://github.com/markmcavoy/androidtoken, 2019.
[8] Bird eats bugs https://birdeatsbug.com/, 2020.
[9] Bug squasher https://thebugsquasher.com/, 2020.
[10] Bugclipper http://bugclipper.com, 2020.
[11] Bugreplay https://www.bugreplay.com/, 2020.
[12] Bugsee https://www.bugsee.com/, 2020.
[13] Instabug https://instabug.com/screen-recording, 2020.
[14] Lucene's tfidfsimilarity javadoc - https://tinyurl.com/ybhqqrqm, 2020.
[15] Outklip https://outklip.com/, 2020.
[16] Python tesseract https://github.com/madmaze/pytesseract, 2020.
[17] Snaffu https://snaffu.squarespace.com/, 2020.
[18] Testfairy https://testfairy.com, 2020.
[19] Ubertesters https://ubertesters.com/bug-reporting-tools/, 2020.
[20] M. Alahmadi, A. Khormi, B. Parajuli, J. Hassel, S. Haiduc, and P. Kumar. Code localization in programming screencasts. *EMSE'20*, 25(2):1536–1572, 2020.
[21] J. H. Andrews, L. C. Briand, and Y. Labiche. Is mutation an appropriate tool for testing experiments? In *ICSE'05*, pages 402–411, 2005.
[22] L. Bao, J. Li, Z. Xing, X. Wang, and B. Zhou. scvripper: video scraping tool for modeling developers' behavior using interaction data. In *ICSE'15*, pages 673–676. IEEE Press, 2015.
[23] H. Bay, T. Tuytelaars, and L. Van Gool. Surf: Speeded up robust features. volume 3951, pages 404–417, 07 2006.
[24] T. Beltramelli. pix2code: Generating code from a graphical user interface screenshot. In *EICS'18*, page 3. ACM, 2018.
[25] C. Bernal-Cárdenas, N. Cooper, K. Moran, O. Chaparro, A. Marcus, and D. Poshyvanyk. Translating video recordings of mobile app usages into replayable scenarios. In *ICSE'20*, 2020.
[26] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann. What makes a good bug report? In *FSE'08*, pages 308–318, New York, NY, USA, 2008. ACM.
[27] N. Bettenburg, R. Premraj, T. Zimmermann, and S. Kim. Duplicate bug reports considered harmful... really? In *ICSM'08*, pages 337–345, Sept 2008.
[28] N. Bettenburg, R. Premraj, T. Zimmermann, and S. Kim. Extracting structural information from bug reports. In *MSR'08*, pages 27–30, 2008.
[29] M. Borg, P. Runeson, J. Johansson, and M. V. Mäntylä. A Replicated Study on Duplicate Detection: Using Apache Lucene to Search Among Android Defects. In *ESEM'14*, pages 8:1–8:4, 2014.
[30] A. Caetano, N. Goulart, M. Fonseca, and J. Jorge. Javasketchit: Issues in sketching the look of user interfaces. In *SSS'02*, pages 9–14, 2002.
[31] Y. Cai, L. Yang, W. Ping, F. Wang, T. Mei, X.-S. Hua, and S. Li. Million-scale near-duplicate video retrieval system. In *MM'11*, page 837–838, New York, NY, USA, 2011.
[32] J. Carreira and A. Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. In *CVPR'17*, 2017.
[33] T.-H. Chang, T. Yeh, and R. Miller. Associating the visual representation of user interfaces with their internal structures and metadata. In *UIST '11*, page 245–256, New York, NY, USA, 2011. ACM.
[34] O. Chaparro, C. Bernal-Cárdenas, J. Lu, K. Moran, A. Marcus, M. Di Penta, D. Poshyvanyk, and V. Ng. Assessing the quality of the steps to reproduce in bug reports. In *ESEC/FSE'19*, Bergamo, Italy, Ausgust 2019 2019.
[35] O. Chaparro, J. M. Florez, and A. Marcus. On the vocabulary agreement in software issue descriptions. In *ICSME'16*, pages 448–452, 2016.
[36] O. Chaparro, J. M. Florez, U. Singh, and A. Marcus. Reformulating queries for duplicate bug report detection. In *SANER'19*, pages 218–229, 2019.
[37] G. Chechik, V. Sharma, U. Shalit, and S. Bengio. Large scale online learning of image similarity through ranking. *JMLR'10*, 11:1109–1135, Mar. 2010.
[38] C. Chen, T. Su, G. Meng, Z. Xing, and Y. Liu. From ui design image to gui skeleton: a neural machine translator to bootstrap mobile gui implementation. In *ICSE'18*, pages 665–676. ACM, 2018.
[39] J. Chen, M. Xie, Z. Xing, C. Chen, X. Xu, and L. Zhu. Object detection for graphical user interface: Old fashioned or deep learning or a combination? In *ESEC/FSE'20*, page to appear, 2020.
[40] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton. A simple framework for contrastive learning of visual representations. In *ICML'20*, pages 1597–1607, 2020.
[41] X. Chen and C. Zhang. An interactive semantic video mining and retrieval platform–application in transportation surveillance video for incident detection. In *ICDM'06*, pages 129–138, 2006.
[42] N. Cooper, C. Bernal-Cárdenas, O. Chaparro, K. Moran, and D. Poshyvanyk. Tango's online appendix https://github.com/ncoop57/tango, 2020.
[43] B. Deka, Z. Huang, C. Franzen, J. Hibschman, D. Afergan, Y. Li, J. Nichols, and R. Kumar. Rico: A mobile app dataset for building data-driven design applications. In *UIST'17*, 2017.
[44] M. Dixon, D. Leventhal, and J. Fogarty. Content and hierarchy in pixel-based methods for reverse engineering interface structure. In *CHI'11*, pages 969–978, 05 2011.
[45] M. Douze, H. Jegou, and C. Schmid. An image-based approach to video copy detection with spatio-temporal post-filtering. *TMM'10*, 12(4):257–266, 2010.
[46] C. Escobar-Velásquez, M. Osorio-Riaño, and M. Linares-Vásquez. Mutapk: Source-codeless mutant generation for android apps. In *ASE'19*, pages 1090–1093, 2019.
[47] D. Freedman, R. Pisani, and R. Purves. *Statistics (4th edn.)*. W. W. Norton & Company, 2007.
[48] C. Frisson, S. Malacria, G. Bailly, and T. Dutoit. Inspectorwidget: A system to analyze users behaviors in their applications. In *CHI'16*, pages 1548–1554. ACM, 2016.
[49] D. Gusfield. Algorithms on stings, trees, and sequences: Computer science and computational biology. *Acm Sigact News*, 28(4):41–60, 1997.
[50] Y. Hao, T. Mu, R. Hong, M. Wang, N. An, and J. Y. Goulermas. Stochastic multiview hashing for large-scale near-duplicate video retrieval. *TMM'17*, 19(1):1–14, 2017.
[51] E. Hatcher and O. Gospodnetic. *Lucene in Action*. Manning Publications, 2004.
[52] J. He, L. Xu, M. Yan, X. Xia, and Y. Lei. Duplicate bug report detection using dual-channel convolutional neural networks. In *ICPC'20*, page to appear, 2020.
[53] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR'16*, pages 770–778, 2016.
[54] A. Hindle, A. Alipour, and E. Stroulia. A contextual approach towards more accurate duplicate bug report detection and ranking. *EMSE'16*, 21(2):368–410, 2016.
[55] A. Hindle and C. Onuczko. Preventing duplicate bug reports by continuously querying bug reports. *EMSE'18*, pages 1–35, 2018.
[56] G. Hu, L. Zhu, and J. Yang. AppFlow: using machine learning to synthesize robust, reusable UI tests. In *ESEC/FSE'18*, pages 269–282. ACM Press.
[57] Y. Jiang and J. Wang. Partial copy detection in videos: A benchmark and an evaluation of popular methods. *TBD'16*, 2(1):32–42, 2016.
[58] Y.-G. Jiang, C.-W. Ngo, and J. Yang. Towards optimal bag-of-features for object categorization and semantic video retrieval. In *IVR'07*, pages 494–501, 07 2007.
[59] W. Jing, X. Nie, C. Cui, X. Xi, G. Yang, and Y. Yin. Global-view hashing: harnessing global relations in near-duplicate video retrieval. *WWW'19*, 22(2):771–789, 2019.
[60] Jing Huang, S. R. Kumar, M. Mitra, Wei-Jing Zhu, and R. Zabih. Image indexing using color correlograms. In *CVPR'97*, pages 762–768, 1997.
[61] N. Jones. Seven best practices for optimizing mobile testing efforts. Technical Report G00248240, Gartner.
[62] H. Jégou, M. Douze, C. Schmid, and P. Pérez. Aggregating local descriptors into a compact image representation. In *CVPR'10*, pages 3304–3311, 2010.
[63] L. Kang. Automated Duplicate Bug Reports Detection - An Experiment at Axis Communication AB. Master's thesis, 2017.
[64] N. Klein, C. S. Corley, and N. A. Kraft. New Features for Duplicate Bug Detection. In *MSR'14*, pages 324–327. ACM, 2014.
[65] G. Kordopatis-Zilos, S. Papadopoulos, I. Patras, and I. Kompatsiaris. Near-duplicate video retrieval by aggregating intermediate cnn layers. In *MMM'17*, volume 10132, pages 251–263, 01 2017.
[66] G. Kordopatis-Zilos, S. Papadopoulos, I. Patras, and I. Kompatsiaris. Fivr: Fine-grained incident video retrieval. *TMM'19*, 21(10):2638–2652, 2019.

[67] G. Kordopatis-Zilos, S. Papadopoulos, I. Patras, and Y. Kompatsiaris. Near-duplicate video retrieval with deep metric learning. In *ICCVW'17*, pages 347–356, 2017.

[68] W. Kraaij and G. Awad. Trecvid 2011 content-based copy detection: Task overview. in Online Proc. TRECVid, 2010,2011.

[69] W. S. Lasecki, J. Kim, N. Rafter, O. Sen, J. P. Bigham, and M. S. Bernstein. Apparition: Crowdsourced user interfaces that come to life as you sketch them. In *CHI'15*, page 1925–1934, New York, NY, USA, 2015. Association for Computing Machinery.

[70] A. Lazar, S. Ritchey, and B. Sharif. Improving the Accuracy of Duplicate Bug Report Detection Using Textual Similarity Measures. In *MSR'14*, pages 308–311, 2014.

[71] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[72] J. Lerch and M. Mezini. Finding Duplicates of Your Yet Unwritten Bug Report. In *CSMR'13*, pages 69–78, 2013.

[73] M. Li, L. Shi, and Q. Wang. Are all duplicates value-neutral? an empirical analysis of duplicate issue reports. In *QRS'19*, pages 272–279. IEEE, 2019.

[74] K. Liu, H. B. K. Tan, and H. Zhang. Has this bug been reported? In *WCRE'13*, pages 82–91, 2013.

[75] D. Lowe. Distinctive image features from scale-invariant keypoints. *JCV'04*, 60:91–, 11 2004.

[76] L. MacLeod, M. Storey, and A. Bergen. Code, camera, action: How software developers document and share program knowledge using youtube. In *ICPC'15*, pages 104–114, 2015.

[77] K. Mao, M. Harman, and Y. Jia. Crowd intelligence enhances automated mobile testing. In *ASE'17*, pages 16–26, Piscataway, NJ, USA, 2017. IEEE Press.

[78] K. Moran, M. Linares-Vásquez, C. Bernal-Cárdenas, and D. Poshyvanyk. Auto-completing bug reports for android applications. Bergamo, Italy, August-September 2015.

[79] K. Moran, M. Linares-Vásquez, C. Bernal-Cárdenas, C. Vendome, and D. Poshyvanyk. Automatically discovering, reporting and reproducing android application crashes. In *ICST'16*, pages 33–44. IEEE, 2016.

[80] K. P. Moran, C. Bernal-Cárdenas, M. Curcio, R. Bonett, and D. Poshyvanyk. Machine Learning-Based Prototyping of Graphical User Interfaces for Mobile Apps. *TSE'18*, 2018.

[81] M. Nayebi. Eye of the mind: Image processing for social coding. In *ICSE'20*, page 49–52, 2020.

[82] A. T. Nguyen, T. T. Nguyen, T. N. Nguyen, D. Lo, and C. Sun. Duplicate Bug Report Detection with a Combination of Information Retrieval and Topic Modeling. In *ASE'12*, pages 70–79, 2012.

[83] T. A. Nguyen and C. Csallner. Reverse engineering mobile application user interfaces with remaui. In *ASE'15*, pages 248–259, Washington, DC, USA, 2015.

[84] A. Panichella. A systematic comparison of search algorithms for topic modelling—a study on duplicate bug report identification. In *SSBSE'19*, pages 11–26. Springer, 2019.

[85] L. Ponzanelli, G. Bavota, A. Mocci, R. Oliveto, M. D. Penta, S. Haiduc, B. Russo, and M. Lanza. Automatic identification and classification of software development video tutorial fragments. *TSE'19*, 45(5):464–488, 2019.

[86] M. S. Rakha, C. Bezemer, and A. E. Hassan. Revisiting the performance evaluation of automated approaches for the retrieval of duplicate issue reports. *TSE'18*, 44(12):1245–1268, 2018.

[87] M. S. Rakha, C.-P. Bezemer, and A. E. Hassan. Revisiting the performance of automated approaches for the retrieval of duplicate reports in issue tracking systems that perform just-in-time duplicate retrieval. *EMSE*, 23(5):2597–2621, 2018.

[88] J. Revaud, M. Douze, C. Schmid, and H. Jégou. Event retrieval in large video collections with circulant temporal encoding. In *CVPR'13*, pages 2459–2466, 2013.

[89] I. M. Rodrigues, D. Aloise, E. R. Fernandes, and M. Dagenais. A soft alignment model for bug deduplication. In *MSR'20*, pages 43–53, 2020.

[90] P. Runeson, M. Alexandersson, and O. Nyholm. Detection of Duplicate Defect Reports Using Natural Language Processing. In *ICSE'07*, pages 499–510, 2007.

[91] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., USA, 1986.

[92] Sivic and Zisserman. Video google: a text retrieval approach to object matching in videos. In *CCV'03*, pages 1470–1477 vol.2, 2003.

[93] C. Sun, D. Lo, S. C. Khoo, and J. Jiang. Towards more accurate retrieval of duplicate bug reports. In *ASE'11*, pages 253–262, 2011.

[94] C. Sun, D. Lo, X. Wang, J. Jiang, and S.-C. Khoo. A Discriminative Model Approach for Accurate Duplicate Bug Report Retrieval. In *ICSE'10*, pages 45–54, 2010.

[95] A. Sureka and P. Jalote. Detecting Duplicate Bug Report Using Character N-Gram-Based Features. In *ASPEC'10*, pages 366–374, 2010.

[96] F. Thung, P. S. Kochhar, and D. Lo. DupFinder: Integrated Tool Support for Duplicate Bug Report Detection. In *ASE'14*, pages 871–874, 2014.

[97] Y. Tian, C. Sun, and D. Lo. Improved Duplicate Bug Report Identification. In *CSMR'12*, pages 385–390, 2012.

[98] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri. Learning spatiotemporal features with 3d convolutional networks. In *ICCV'15*, pages 4489–4497, 2015.

[99] J. Wang, M. Li, S. Wang, T. Menzies, and Q. Wang. Images don't lie: Duplicate crowdtesting reports detection with screenshot information. *IST*, 110:139–155, 2019.

[100] J. Wang, Y. Yang, T. Menzies, and Q. Wang. isense2. 0: Improving completion-aware crowdtesting management with duplicate tagger and sanity checker. *TOSEM*, 29(4):1–27, 2020.

[101] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun. An Approach to Detecting Duplicate Bug Reports Using Natural Language and Execution Information. In *ICSE'08*, pages 461–470, 2008.

[102] X. Wu, A. G. Hauptmann, and C.-W. Ngo. Practical elimination of near-duplicates from web video search. In *MM'07*, page 218–227, New York, NY, USA, 2007. Association for Computing Machinery.

[103] S. Yadid and E. Yahav. Extracting code from programming tutorial videos. In *Onward!'16*, page 98–111, New York, NY, USA, 2016. ACM.

[104] D. Zhao, Z. Xing, C. Chen, X. Xia, and G. Li. Actionnet: vision-based workflow action recognition from programming screencasts. In *ICSE'19*, pages 350–361. IEEE, 2019.

[105] D. Zhao, Z. Xing, C. Chen, X. Xu, L. Zhu, G. Li, and J. Wang. Seenomaly: Vision-based linting of gui animation effects against design-don't guidelines. In *ICSE'20*, 2020.

[106] G. Zhao and M. Pietikäinen. Dynamic texture recognition using local binary patterns with an application to facial expressions. *PAMI'07*, 29:915–28, 07 2007.

[107] J. Zhou and H. Zhang. Learning to Rank Duplicate Bug Reports. In *CIKM'18*, pages 852–861, 2012.