Image Understanding of GUI Widgets for Test Reuse

Yash Mishra
School of Computer Science and Engineering
VIT University
Vellore, India
yash.mishra2020@vitstudent.ac.in

Prithak Gajurel
School of Computer Science and Engineering
VIT University
Vellore, India
prithak.gajurel2020@vitstudent.ac.in

Prajwal Lamsal
School of Computer Science and Engineering
VIT University
Vellore, India
prajwal.lamsal2020@vitstudent.ac.in

Swarnalatha P
School of Computer Science and Engineering
VIT University
Vellore, India
pswarnalatha@vit.ac.in

***Abstract*— Testing of UI after its development is an essential activity. Due to the complexity of the task of developing new tests for UI, it is an expensive one. Different approaches for reusing tests in similar scenarios have been created. These different methods employ different information available in the GUI to decide the appropriate transfer of tests from one UI to another. Similar GUI events are related together by these systems to decide the transferability of tests. The usage of semantic knowledge during the matching of GUI events has proven to be an important asset in understanding their similarities. The usage of textual information available in GUI and their semantic meanings in this task have promising results. However, the usage of semantic information that can be obtained from images and icons has remained unexplored. This study presents the integration of semantic annotation of icons, widgets and images for semantic matching of GUI events. Proposed findings demonstrate the importance of image understanding of GUI widgets in test reuse and offer practical insights for software developers and researchers. In addition to improving semantic matching, the integration of semantic annotation of icons, widgets, and images can also enhance the interpretability and maintainability of UI tests.**

Keywords—software testing, natural language processing, mobile applications, test reuse

I. INTRODUCTION

The study utilizes mobile applications to perform a wide array of tasks on a daily basis. Due to this, it is mandatory to rigorously test these apps so as to ensure that it is capable of performing their intended tasks well. Manual creation of test scenarios becomes very costly due to the involvement of humans in building and running the test cases [1]. A test case is composed of two parts: (1) a trace of events that interact with the UI, and (2) assertion oracles that provide information on the UI state [2]. Manual creation of UI tests is a very tedious task, especially considering the involvement of rapid app-development lifecycle in the building of mobile apps [3]. Hence, to help reduce the time consumption and manual efforts researchers are working on creating techniques that can generate test cases automatically [4]. However, despite many contributions made to the field of automated test case generation, usage of manually created test cases is still preferred in the industry. The reasons for this occurrence are mainly: (1) Lack of context-aware text inputs, (2) Failing to generate expressive tests, and (3) Absence of test oracles [5].

Researchers have discovered that by using the semantic similarities between different GUIs, effective test migration from one GUI to another becomes possible [4]. Figure 1 depicts an example in which the existing test (a) of the application EveryDollar: Budgeting is successfully migrated to the application Mint: Budget, Bills, Finance, and the reused test (b) is obtained. As Figure 1 shows, events e^s_1, and e^s_2 in test (b) are similar to e^t_1, and e^t_2 in test (a), respectively. ATM [1] and CraftDroid [5] are designed to incorporate test generation with the semantic matching of UI events. By using word embedding techniques on the textual descriptors of events in the GUI widgets, semantic matching of GUI events finds semantically related events in source and target apps. In order to move GUI tests from the source app to the target app, test creation takes advantage of the similarities found with semantic matching. The success of semantic GUI event matching has a major impact on the overall effectiveness of test reuse. The matching of the events between the source and the target test is driven by semantic matching. [6]. The FrUITeR framework was proposed to compare test reuse methodologies. FrUITeR contrasts test reuse methods as a whole but does not evaluate semantic matching separately [7].

Instead of merely employing textual descriptors of events, the UI's images and icons can also be employed as semantic descriptors. It carry extra information that may be relevant to completing this assignment. A variety of machine learning methods [8], [9] have been developed with the goal of extracting information from widgets and icons. A framework was provided in [6] for studying the performance of different training sets, word embedding techniques and semantic matching algorithms based on the textual descriptors present in the GUI widgets. In this study, the same framework is utilized for studying the performance difference in these different dimensions when the same set of GUI events are used with and without semantically annotated icons and images. This data was obtained by using the Rico dataset [10]. The semantic annotations provided in the dataset are designed to categorize the information into 25 UI component categories, 197 text button concepts and 99 icon classes [11].

II. RELATED WORK

The design of mobile applications is created to allow easy interaction between the user and the interface. Testing the UI while being an essential task can also be daunting. Due to this reason, there has been a gradual rise in the research on the

reusability of tests for testing. AppFlow is a system where developers could write libraries and modules of tests and the system would create full test cases for the UI based on the screens provided [12]. While GUI between two apps can be very different, they might share similarities that enable test transfer between them. AppTestMigrator (ATM) [1] and CraftDroid [5] are both based on this principle. Both approaches try to achieve mapping from source events and oracles to target events and oracles in order to perform successful test migration. However, they utilize some different attributes of GUI elements and utilize different aggregation functions while calculating similarity scores causing differences in produced results [6]. Both ATM and CraftDroid utilize greedy approaches to perform their tasks. AdaptDroid uses an evolutionary approach to solve the same problem and introduces another viable direction for finding a solution [2]. Test Reuse based on Adaptive Semantic Matching (TRASM) is another proposed technique that performs test adaptation to create new tests for a target app based on its similarities to the source app [4]. All of these techniques focus on test migrations within different apps on the same platform (Android). Test migration in sibling apps across different platforms is achieved by MAPIT by building models of the source and target UI and performing mapping between these models [3].

Testing of a test reusability technique itself is also very essential to understanding which technique is best suited to the task. FrUITeR is a framework for evaluating and testing UI test reuse techniques. It builds on the existing evaluation techniques used in AppFlow, ATM and CraftDroid [7]. FrUITeR performs the evaluation as a whole and cannot decide the reason behind the better or worse performance of a technique. To enable understanding of the contribution of individual attributes a framework was introduced in [6] which can evaluate the reasons behind better and worse performances. It does so by grouping the attributes used by ATM and CraftDroid into four categories: (1) ATM, (2) CraftDroid, (3) Intersection of ATM and CraftDroid and (4) Union of ATM and CraftDroid. It also analyzes the significance of different word embedding models in successful test migration.

Classification of widgets is an important subproblem in understanding the semantics of icons and widgets. IconIntent is a framework that analyzes the program as well as studies the UI design through computer vision to classify the widgets into eight sensitive categories [8]. Classification of icons and widgets has shown to have other applications as well. LabelDroid uses CNN and transformers to provide categories to image-based buttons so that these categories can be used to help make these apps more accessible to blind people [9]. This label assignment indicates that these icons and widgets also carry a large amount of necessary information about the UI. The same technique is integrated with a fill algorithm and utilized for the intent generation of widgets from UI screenshots in [13]. ReDraw is also another system that analyzes screen captures with a fully dynamic program and performs GUI component classification [14]. [15] tries to solve the issue of mobile fragmentation during test migration as the same tests may yield different results on different devices. It does so by using a feature-based analysis by introducing the Fullscreen algorithm and the Cropped algorithm.

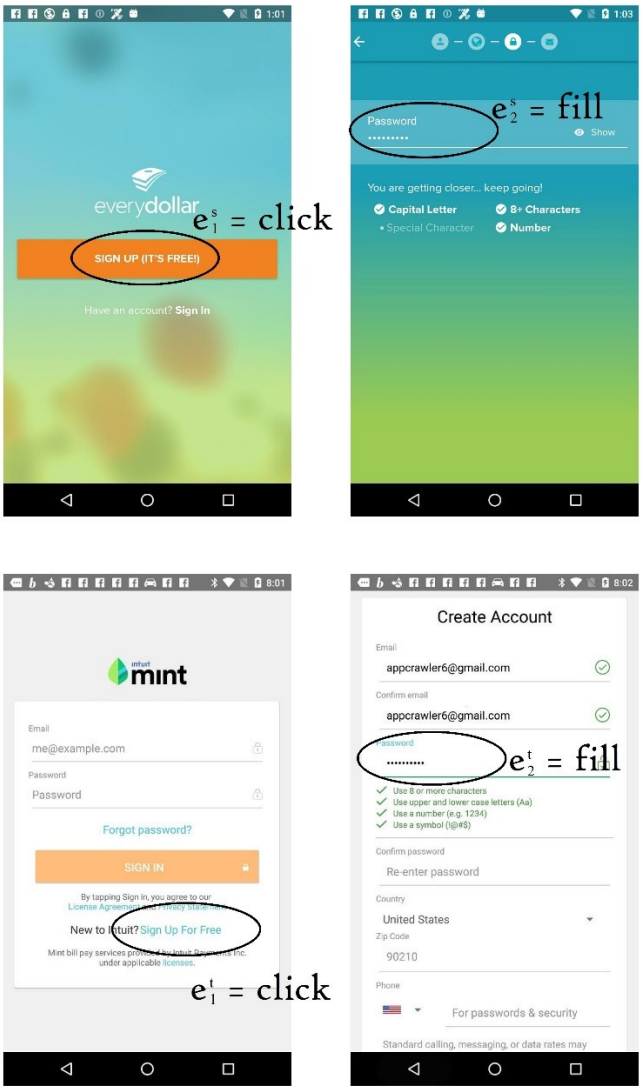

Fig. 1. Matching events in apps EveryDollar: Budgeting (top) and Mint: Budget, Bills, Finance (bottom)

A new semantic matching algorithm SemFinder is introduced in [6]. Then it utilizes a framework to study the importance of different attributes of events in test migration. The framework also performs a comparison between ATM, CraftDroid and SemFinder algorithms. The integration of information added using semantic understanding of icons, images and widgets however remains an unexplored area of study. This paper utilizes the same framework to study the change brought about by using images, icons and widgets as semantic descriptors along with using other textual information such as in [6].

III. Methodology

The main goal of the study is to analyze the differences between using only textual descriptors of events in GUI events as semantic descriptors during UI event matching for test reuse and including semantically annotated icons and images as well in semantic descriptors. This is performed by creating both of these forms of data from the same GUI event traces and then using the framework proposed in [6] for evaluation. The major aspects of the method can be divided into five components: (1) dataset of GUI, (2) corpus of documents, (3) word embeddings, (4) event descriptor extractor, and (5) semantic matching algorithms.

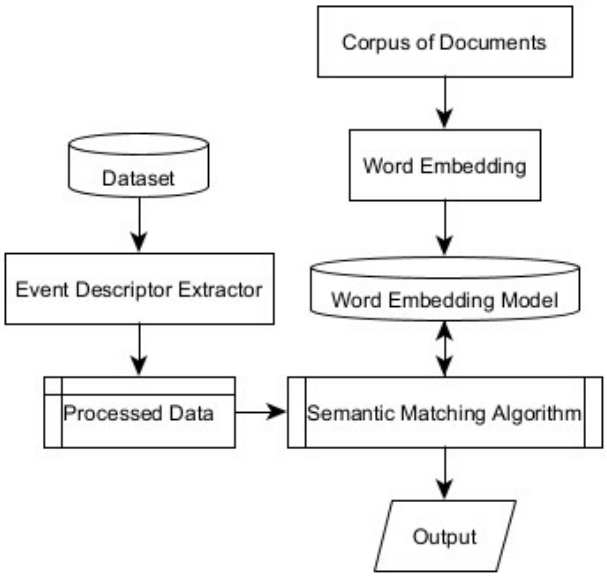

Fig. 2. Proposed System

The Rico dataset[10] has been used as the dataset for obtaining sample GUI data. 50 individual GUI event traces were selected from the dataset as data for the experiment. The corpus of documents is used to build the word embedding models and provide descriptions for the semantics of words and sentences in the corpus. The event descriptor extractor is used to extract relations and mapping between source event traces and target event traces from the GUI event traces. Semantic matching algorithms calculate similarity scores between the attributes of source events and target events. The three semantic algorithms used are ATM [1], CraftDroid [5] and SemFinder [6]. The components of the method are depicted in Figure 2.

A. *Dataset of GUI*

The Rico dataset consists of 10,811 GUI event traces and 72,219 unique GUI screenshots from 9,772 Android applications spanning 27 categories in the Play Store [10]. Each trace file consists of the UI screenshots of the trace, JSON files describing the hierarchies in the corresponding screenshots and a 'gestures.json' file depicting where the user clicked on the screenshot. Also for each screenshot, a corresponding image describing the semantically annotated descriptions is provided along with a JSON file describing the hierarchies of the UI. This JSON file also consists of information related to the semantic annotation of each component. The screenshots and hierarchies provided in the dataset can categorize the information into 25 UI component categories, 197 text button concepts and 99 icon classes [11]. An example of a UI screenshot and its corresponding semantically annotated image is shown in Figure 3.

B. *Corpus of Documents*

The study considers the same corpora of English documents that were considered in [6]: (1) Blog Authorship Corpus (Blogs) [16] that contains 681,288 posts from 19,320 bloggers, (2) User Manuals of Android apps (Manuals) [1] that contains user manuals of 500 Android applications, and (3) Apps Descriptions (Google-play) [6] that contains information of 900,805 Android apps in the Google Play Store.

Fig. 3. Example of a screenshot and its corresponding semantic annotation in the Rico dataset

Good quality corpus of documents is essential for building good word embedding models. To achieve this a corpus of documents must consist of a maximum number of unique words and the words used in the corpus must be related to the task in question.

C. *Word Embeddings*

The study uses the following word embedding techniques:

- Word Mover's distance (WM) [17]: a technique based on measuring the minimum distance between embedded words of one text to another. It relies on the fact that semantic relationships are generally preserved even after converting words to vectors.
- Word2vec [18]: a popular technique that utilizes shallow neural networks to convert words to a continuous vector form.
- Neural Network Language Model (NNLM) [19]: a neural network technique that utilizes a continuous vector space for word embedding and uses single hidden layer networks for estimating probabilities.
- Universal Sentence Encoder (USE) [20]: a technique that proposes the conversion of entire sentences to vectors.
- GlobalVectors (Glove) [21]: a technique that performs the word embedding task by capturing the global corpus statistics.
- Fast Text (Fast) [22]: a technique that tries to address the out-of-vocabulary issue by using n-grams within words.

D. *Event Descriptor Extractor*

The event descriptions to be fed into the framework are in the form of CSV files with the columns: event_index, id, text, semantic_rep, file_name, class, type, content_desc, hint, parent_text, sibling_text, fillable_neighbor, neighbors, label, activity and atm_neighbor. For each event trace, all of these values are extracted from their corresponding hierarchy files related to traces and semantic annotations. The semantic annotations for icons, text buttons and other general components are provided in the semantic annotation files as iconClass, textButtonClass and componentLabel respectively. The framework deals with these attributes by grouping them into four categories: ATM(text, id, content_desc, hint, file_name, neighbors), CraftDroid(text, id, content_desc, hint, activity, parent_text, sibling_text), Intersection(text, id, content_desc, hint) and Union(text, id, content_desc, hint, file_name, activity, neighbors, parent_text, sibling_text).

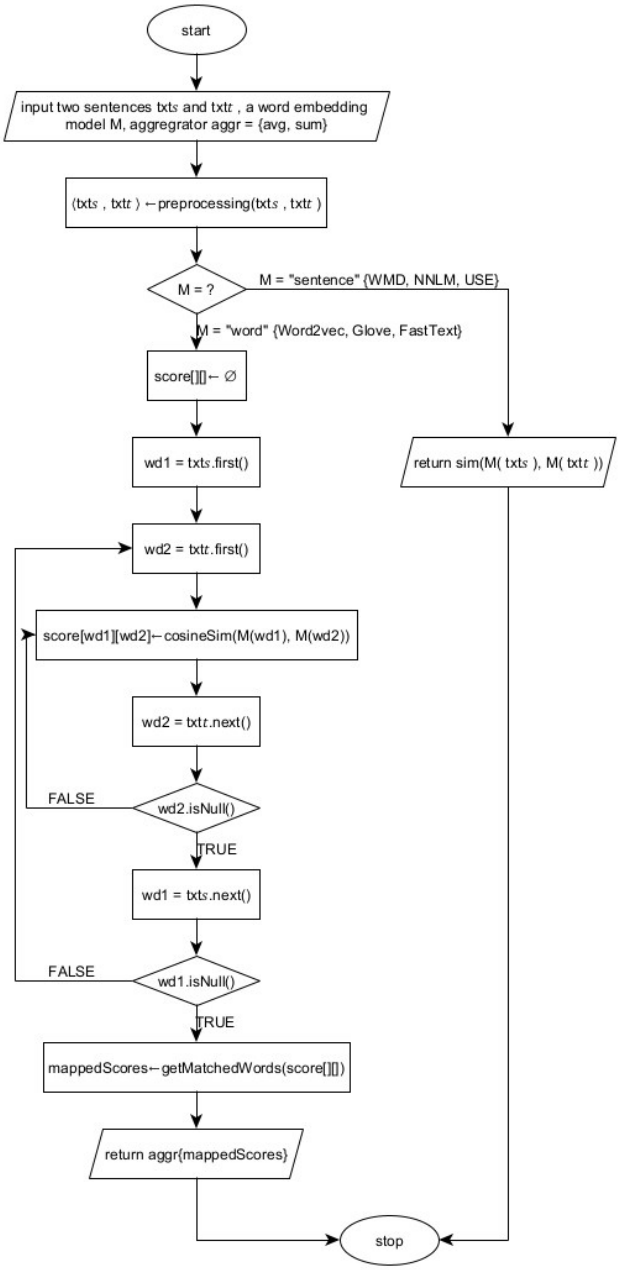

Fig. 4. Flowchart describing getSimScore algorithm

E. Semantic Matching Algorithm

ATM, CraftDroid, and SemFinder algorithms are used for test reuse approaches by the framework. These algorithms determine how to generate target test cases by analyzing lists of target events based on similarity scores computed between the source and target event descriptors. They rely on a word embedding model to compute the semantic similarity scores among attribute values. The shared function in these algorithms is "getSimScore", which calculates the similarity scores between two sentences, "txt*s*" and "txt*t*," obtained from the textual attributes of the source and target descriptors. The function preprocesses the sentences, removes stop words, performs lemmatization, and handles camel case notation. It then computes the similarity scores by comparing word vectors using cosine similarity. The function returns a similarity score using the specified aggregation function (average or sum). A flowchart describing the getSimScore function is given in Figure 4.

The differences and similarities between the algorithms have been described in Figure 5. The key differences among the algorithms can be summarized in three points: (1) Attributes compared: ATM and CraftDroid have restrictions on the attribute types they compare, while SemFinder considers all attributes regardless of their type. (2) Aggregation of similarity scores: ATM uses the maximum or sum, CraftDroid uses the average, and SemFinder averages scores at the sentence level. (3) Aggregation of word-level word embedding models: ATM uses sum, CraftDroid uses average, and SemFinder uses average as an aggregation function.

IV. EXPERIMENT

The study aims to compare the use of textual descriptors alone versus incorporating semantically annotated icons and images as semantic descriptors in GUI event matching for test reuse. The Rico dataset[10] provides GUI event traces and semantically annotated screenshots. Various word embedding techniques are employed. The event descriptor extractor extracts attributes from GUI event traces, and the semantic matching algorithms compute similarity scores. Evaluation is based on MRR and TOP1 metrics.

A. Experimental Setup

The experiments were conducted using different combinations of corpora and word embedding techniques. 12 pairwise combinations of three corpora (Manuals, Blogs, and Google-play) and four word embedding techniques (Word2vec, WMD, Glove, and Fast), resulting in 12 word embedding models were considered. Detailed depictions of the combinations used are shown in Figure 6.

Before applying the word embedding techniques, pre-processing steps similar to those described in the "getSimScore" algorithm were used. Pretrained models provided by the authors of the techniques, which were trained on different corpora (e.g., versions of Google News and Twitter datasets) that are not publicly available were utilized by the framework. For certain word embedding techniques such as USE, and NNLM, the framework relies solely on the pre-computed models and does not build models using the three corpora mentioned earlier.

Two syntactic approaches for computing the syntactic similarity of words/sentences: edit-distance based similarity (ES) and the Jaccard Similarity index (JS) have also been considered in the framework. Since both ES and JS do not utilize the corpus of documents, they are not combined with the three corpora. ES calculates the normalized similarity of two words using the "Levenshtein distance" metric, which measures the minimum number of operations required to transform one word into another. JS computes the similarity of two sets (sentences) by dividing the number of shared elements by the total number of unique elements in both sets.

B. Evaluation Metrics

The evaluation of the semantic matching effectiveness for each of the 228 combinations is performed using two metrics: Mean Reciprocal Rank (MRR) and the proportion of queries for which the correct answer ranks at one (TOP1).MRR, based on reciprocal ranks, is calculated as the average of the reciprocal ranks of the queries. It considers the multiplicative inverse of the rank of the first correct answer, ranging from 1 for first place to 1/n for the nth place.

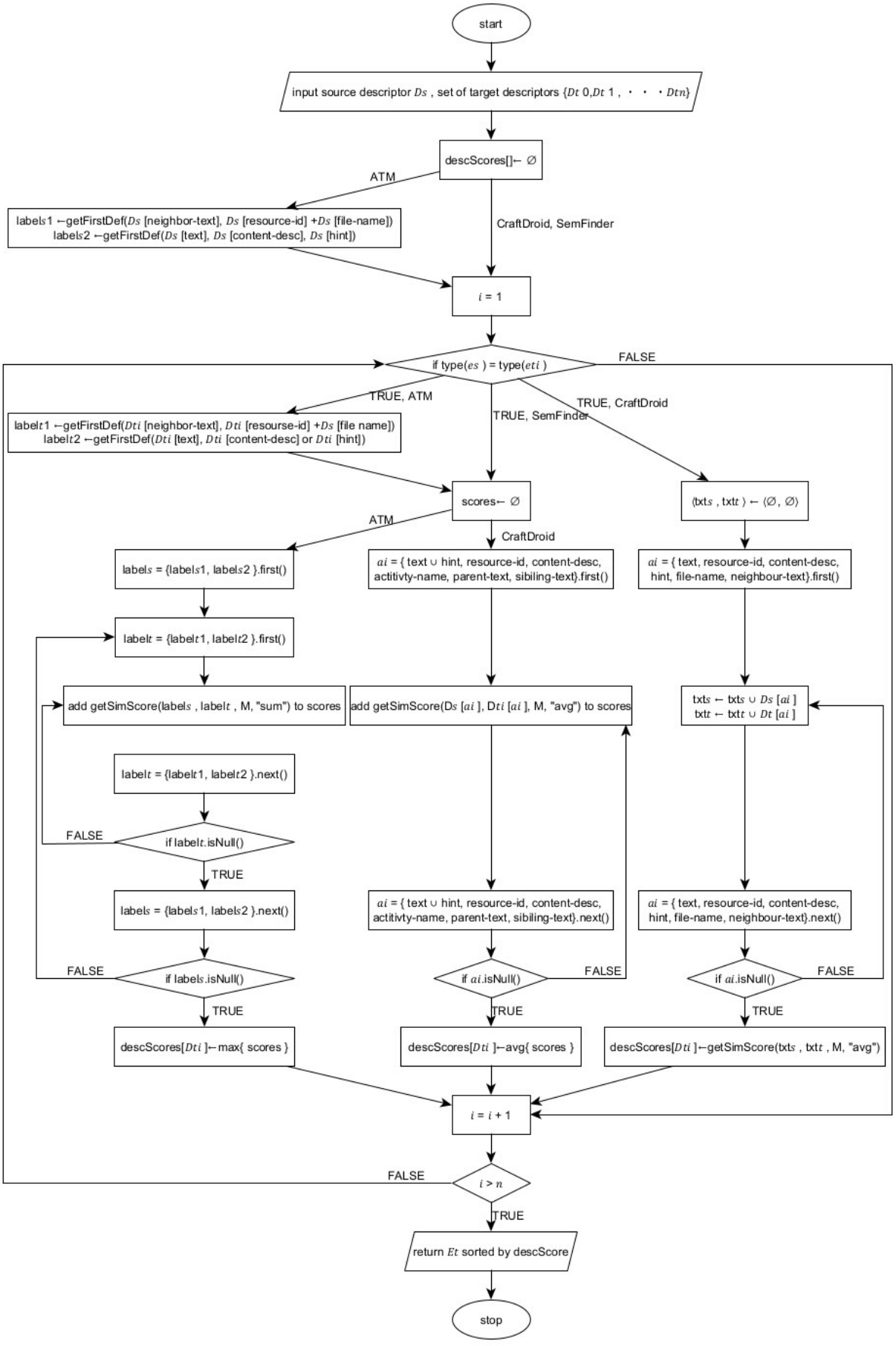

Fig. 5. Flowchart describing semantic matching algorithms

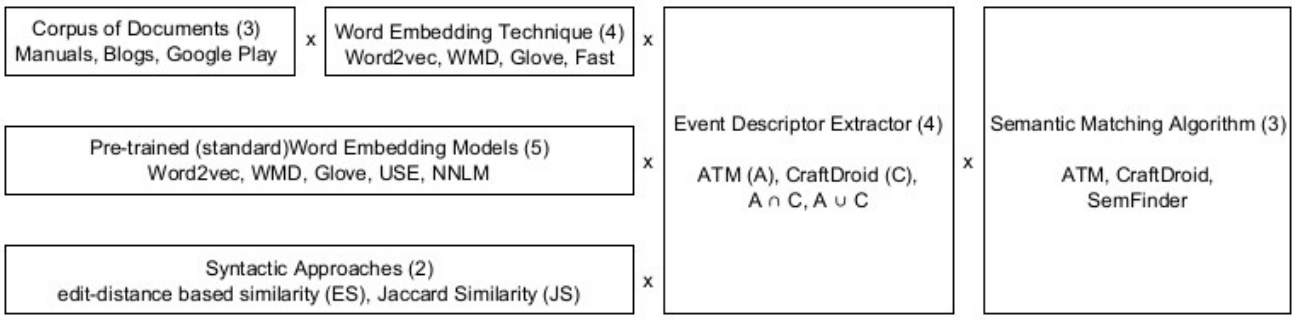

Fig. 6. Diagram describing different configurations used in the framework

TOP1 is the ratio of queries where the ground truth ($et\ gt$) is positioned first in the returned list of events. It indicates the percentage of queries where the correct answer is at the top position.

$$MRR = \frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{rank_i} \in (0;1] \quad (1)$$

$$TOP1 = \frac{1}{|Q|}\sum_{i=1}^{|Q|}\begin{cases}1 & if\ rank_i = 1\\ 0 & otherwise\end{cases} \in [0;1] \quad (2)$$

Both MRR and TOP1 are used to assess the effectiveness of the semantic matching in retrieving the correct answer among the candidate events in the list.

V. Results

The event queries were run through 228 configurations. The values of MRR without images ranged from 0.471 to 0.696. The values of MRR with images ranged from 0.491 to 0.745. The values of Top1 without images ranged from 0.045 to 0.409. The values of Top1 with images ranged from 0.071 to 0.455.

The configuration which showed the most improvement in MRR value after integration of semantic annotation of images, icons and widgets was [ATM_a, Intersection, Google Play, WM] with an increment of 0.062. The configuration which showed the most improvement in Top1 value after integration of semantic annotation of images, icons and widgets was [SemFinder, Union, Standard, USE] with an increment of 0.107. All configurations showed small increments in their performances after the incorporation of semantic annotations of images.

The top 10 models with the biggest increases in MRR values are shown in Figure 7. The distributions of the various component instances for three percentiles are shown in Table 1. The top 3 entries received 1%, the top 12 entries received 5%, and the top 23 entries received 10%. The percentage of entries in the chosen percentile (column) that employ a specific component (row) is shown by the values in the cells. For example, in the list ordered by the difference in MRR measure, every entry in the first percentile (1%) employs the ATM algorithm.

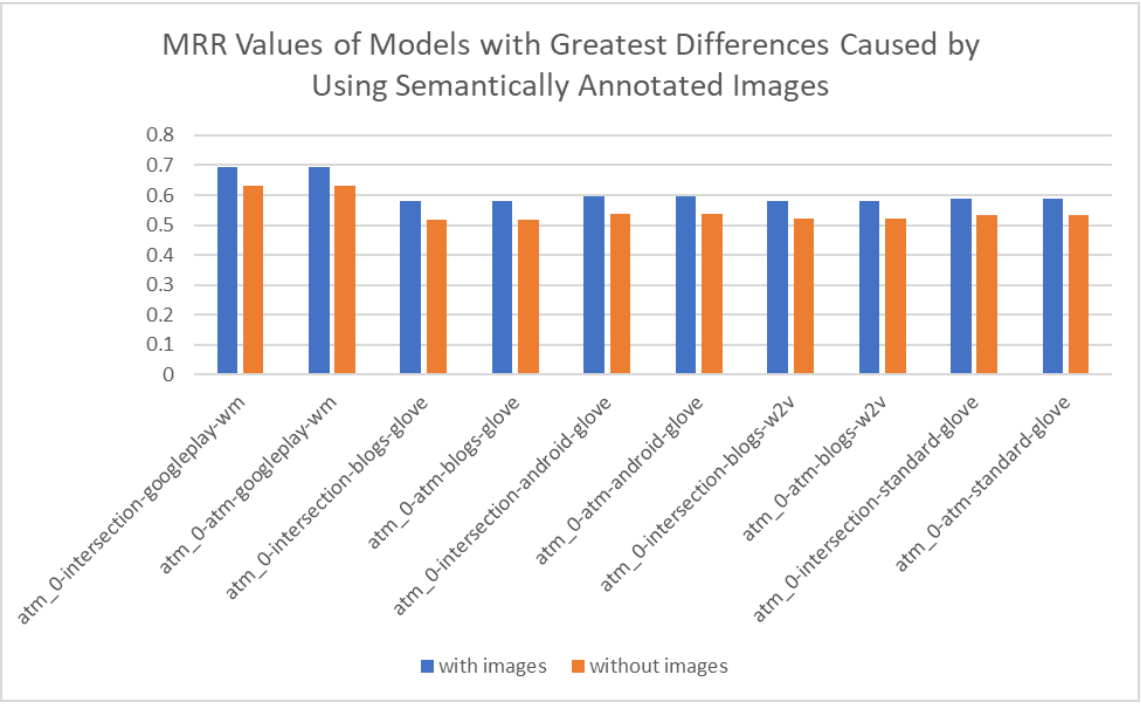

Fig. 7. MRR values of the top 10 models with the highest increments

TABLE I. DISTRIBUTIONS OF THE 228 CONFIGURATIONS SORTED ACCORDING TO MRR DIFFERENCE AND TOP 1 DIFFERENCE BASED ON THE PERCENTILES 1%, 5%, AND 10% IN PERCENTAGE

Type	**Instance**	**MRR**			**Top1**		
		1%	***5%***	***10%***	***1%***	***5%***	***10%***
Training Set	blogs	33	33	17	33	17	17
	manuals	0	0	0	0	0	0
	google play	67	33	17	0	17	22
	standard	0	17	35	67	33	35
	android	0	17	31	0	33	26
Word Embedding	w2v	0	17	26	0	33	26
	glove	33	50	26	33	17	34
	wm	67	17	17	0	0	9
	fast	0	17	13	0	17	13
	nnlm	0	0	9	0	17	9
	use	0	0	9	67	17	9
	js	0	0	0	0	0	0
	es	0	0	0	0	0	0
Descriptor	atm	33	50	43	0	42	43
	craftdroid	0	0	4	33	8	4
	intersection	67	50	48	33	42	48
	union	0	0	4	33	8	4
Algorithm	atm_a	100	100	91	33	67	74
	craftdroid_a	0	0	0	0	0	0
	semfinder_a	0	0	9	67	33	26

VI. DISCUSSION

The results of the experiment provide valuable insights into the performance of different configurations and techniques for semantic matching of GUI events. 228 configurations were utilized from the framework to judge our proposed approach. The incorporation of semantic annotations of images, icons, and widgets showed an improvement in the Mean Reciprocal Rank (MRR) and Top1 values, albeit with small increments in performance.

When comparing MRR values without images to those with images, the inclusion of image semantics resulted in higher MRR values across all configurations. The range of MRR values with images (0.491 to 0.745) is higher than that without images (0.471 to 0.696). This suggests that leveraging the semantic information from images contributes to a better understanding and matching of GUI events.

Similarly, when examining the Top1 values, the inclusion of image semantics led to higher Top1 values compared to configurations without image semantics. The range of Top1 values with images (0.071 to 0.455) surpasses the range without images (0.045 to 0.409). This further supports the idea that considering image semantics improves the accuracy of matching GUI events.

Among the different configurations, the one that demonstrated the most significant improvement in MRR value after integrating semantic annotation of images, icons, and widgets was [ATM_a, Intersection, Google Play, WM]. This particular configuration achieved an increment of 0.062 in MRR. On the other hand, the configuration that showed the largest improvement in Top1 value was [SemFinder, Union, Standard, USE], with an increase of 0.107. These configurations highlight the effectiveness of specific combinations of event descriptors and semantic matching algorithms when incorporating image semantics.

Analyzing the top 10 models with the biggest increases in MRR values allows us to identify specific combinations that have shown substantial improvement. This information can guide future research and development efforts, focusing on these successful configurations to enhance the semantic matching of GUI events.

The distribution analysis of component instances for different percentiles provides insights into the utilization of various components in the chosen configurations. For example, in the first percentile (1% of the top-performing models based on MRR difference), all entries employ the ATM algorithm. This indicates that the ATM algorithm has shown consistent effectiveness in achieving higher MRR values. Similarly, the distribution of other components can be examined to understand their impact on the performance of semantic matching.

While the incorporation of semantic annotations of images, icons, and widgets resulted in improvements, it is worth noting that the performance increments were relatively small. This suggests that while image semantics provide valuable information, it might not be the sole determinant for accurate semantic matching of GUI events. Other factors, such as text-based semantics and syntactic approaches, still play a significant role in achieving better matching results.

VII. CONCLUSION

The aim of this study was to investigate the impact of incorporating semantic annotations of images, icons, and widgets on the performance of semantic matching of GUI events. The results demonstrated that leveraging image semantics led to improvements in the Mean Reciprocal Rank (MRR) and Top1 values, indicating enhanced accuracy in matching GUI events.

The comparison between configurations with and without image semantics revealed that all configurations benefited from the inclusion of image semantics, as evidenced by the higher MRR and Top1 values observed in the configurations that incorporated image semantics. This finding emphasizes the importance of considering visual information when performing semantic matching in GUI event recognition.

Although the incorporation of image semantics demonstrated improvements in matching performance, it is important to note that the observed increments were relatively small. This suggests that while image semantics contribute valuable information, other factors, such as text-based semantics and syntactic approaches, should also be considered to achieve more accurate and robust semantic matching of GUI events. Future research directions may include real–time identification and classification of icons, images and widgets from screenshots by leveraging machine learning techniques such as in [13] and integrating the information while migrating test cases along with other useful information.

REFERENCES

[1] F. Behrang and A. Orso, "Test migration between mobile apps with similar functionality," in 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), San Diego, CA, USA, 2019, pp. 54-65.

[2] L. Mariani, M. Pezzè, V. Terragni and D. Zuddas, "An evolutionary approach to adapt tests across mobile apps," in 2021 IEEE/ACM International Conference on Automation of Software Test (AST), Madrid, Spain, 2021, pp. 70-79.

[3] S. Talebipour, Y. Zhao, L. Dojcilović, C. Li, and N. Medvidović, "UI test migration across mobile platforms," in 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE), Melbourne, Australia, 2021, pp. 756-767.

[4] S. Liu, Y. Zhou, T. Han, and T. Chen, "Test reuse based on adaptive semantic matching across android mobile applications," in 2022 IEEE 22nd International Conference on Software Quality, Reliability and Security (QRS), Guangzhou, China, 2022, pp. 703-709.

[5] J.-W. Lin, R. Jabbarvand, and S. Malek, "Test transfer across mobile apps through semantic mapping," in 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), San Diego, CA, USA, 2019, pp. 42-53.

[6] L. Mariani, A. Mohebbi, M. Pezzè, and V. Terragni, "Semantic matching of gui events for test reuse: are we there yet?," in Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis, July 2021, pp. 177-190.

[7] Y. Zhao, J. Chen, A. Sejfia, M. Schmitt Laser, J. Zhang, F. Sarro, and N. Medvidovic, "FrUITeR: a framework for evaluating ui test reuse," in Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, November 2020, pp. 1190-1201.

[8] X. Xiao, X. Wang, Z. Cao, H. Wang, and P. Gao, "IconIntent: automatic identification of sensitive ui widgets based on icon classification for android apps," in 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), Montreal, QC, Canada, 2019, pp. 257-268.

[9] J. Chen, C. Chen, Z. Xing, X. Xu, L. Zhu, G. Li, and J. Wang, "Unblind your apps: predicting natural-language labels for mobile gui components by deep learning," in Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering (ICSE '20), June 2020, pp. 322-334.

[10] 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 Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology (UIST '17), October 2017, pp. 845-854.

[11] T. F. Liu, M. Craft, J. Situ, E. Yumer, R. Mech, and R. Kumar, "Learning design semantics for mobile apps," in Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology (UIST '18), October 2018, pp. 569-579.

[12] G. Hu, L. Zhu, and J. Yang, "AppFlow: using machine learning to synthesize robust, reusable ui tests," in Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2018), October 2018, pp. 269-282.

[13] P. Zhu, Y. Li, T. Li, W. Yang, and Y. Xu, "GUI widget detection and intent generation via image understanding," in IEEE Access, vol. 9, pp. 160697-160707, 2021.

[14] K. Moran, C. Bernal-Cárdenas, M. Curcio, R. Bonett, and D. Poshyvanyk, "Machine learning-based prototyping of graphical user interfaces for mobile apps," in IEEE Transactions on Software Engineering, vol. 46, no. 2, pp. 196-221, 1 Feb. 2020.

[15] L. Ardito, A. Bottino, R. Coppola, F. Lamberti, F. Manigrasso, L. Morra, and M. Torchiano, "Feature matching-based approaches to improve the robustness of android visual gui testing," ACM Transactions on Software Engineering and Methodology (TOSEM), vol. 31, no. 2, Article No. 21, pp. 1-32, 2021.

[16] J. Schler, M. Koppel, S. Argamon, and J. Pennebaker, "Effects of age and gender on blogging," in AAAI Spring Symposium: Computational Approaches to Analyzing Weblogs, vol. 6, pp. 199-205, 2006.

[17] M. J. Kusner, Y. Sun, N. I. Kolkin, and K. Q. Weinberger, "From word embeddings to document distances," in Proceedings of the International Conference on International Conference on Machine Learning (ICML '15), 2015, pp. 957-966.

[18] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," arXiv preprint arXiv:1301.3781, 2013.

[19] E. Arisoy, T. N. Sainath, B. Kingsbury, and B. Ramabhadran, "Deep neural network language models," in Proceedings of the NAACL-HLT 2012 Workshop: Will We Ever Really Replace the N-gram Model? On the Future of Language Modeling for HLT, 2012, pp. 20-28.

[20] D. Cer et al., "Universal sentence encoder," arXiv preprint arXiv:1803.11175, 2018.

[21] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), pp. 1532-1543, Oct. 2014.

[22] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," Transactions of the Association for Computational Linguistics, vol. 5, pp. 135-146, 2017.