GenGUI: A Dataset for Automatic Generation of Web User Interfaces Using ChatGPT

Mădălina Dicu[1][a], Enol García González[2][b], Camelia Chira[1][c] and José R. Villar[2][d]
[1]*Faculty of Mathematics and Computer Science, Babeș-Bolyai University, Str. Mihail Kogălniceanu nr. 1, Cluj-Napoca 400084, Romania*
[2]*Department of Computer Science, University of Oviedo, C. Jesús Arias de Velasco, s/n, Oviedo 33005, Spain*
{*madalina.dicu, camelia.chira*}*@ubbcluj.ro*, {*garciaenol, villarjose*}*@uniovi.es*

Keywords: User Interface Recognition, Dataset, Computer Vision, ChatGPT, Object Detection.

Abstract: The identification of elements in user interfaces is a problem that can generate great interest in current times due to the significant interaction between users and machines. Digital technologies are increasingly used to carry out almost any daily task. Computer vision can be helpful in different applications, such as accessibility, testing, or automatic code generation, to accurately identify the elements that make up a graphical interface. This paper focuses on one problem that affects almost any Deep Learning and computer vision problem, which is the generation and annotation of datasets. Few contributions in the literature provide datasets to train vision models to solve this problem. Moreover, analyzing the literature, most datasets focus on generating images of mobile applications, all in English. In this paper, we propose GenGUI, a new dataset of desktop applications that presents various contents, including multiple languages. Furthermore, this contribution will train different versions of YOLO models using GenGUI to test their quality with reasonably good results.

1 INTRODUCTION

The great technological advances of the last years have made us increasingly dependent on digital devices such as computers or smartphones to carry out multiple daily tasks efficiently. The increased use of digital tools has led to the development of a new problem in recent years: detecting elements in graphical interfaces. Automating the detection of elements in the graphical interfaces of daily applications is essential for developing and using digital tools. Some examples where this problem is relevant are the testing of user interfaces (Bielik et al., 2018; Qian et al., 2020; White et al., 2019; Yeh et al., 2009), the analysis and improvement of the accessibility (Zhang et al., 2021; Miñón et al., 2013; Xiao et al., 2024), the autogeneration of code for interfaces from images (Chen et al., 2018; Moran et al., 2020; Nguyen and Csallner, 2015), and the search for content within user interfaces (Deka et al., 2017; Reiss, 2014).

The problem of element detection in user interfaces poses a situation in which, starting from an image of a user interface, it is necessary to detail the elements that compose it, including the position, size, and type of each element present in the user interface. It is, therefore, a problem in the field of computer vision. To develop a good model that works well in detecting user interfaces, it is essential to have a large and varied dataset to train the models. However, the datasets currently available in the literature are insufficient, as they contain only a limited number of images. For example, see datasets (Bunian et al., 2021) and (Dicu et al., 2024a). Moreover, these present limitations in the variety of elements and languages.

The current work aims to present a novel way to generate user interfaces using ChatGPT (OpenAI, 2024) automatically. The goal is to build a large dataset, named GenGUI, with many user interfaces and a wide variety of elements and languages to develop a good computer vision model capable of successfully recognizing elements in user interfaces. With this use of ChatGPT, we have artificially generated a dataset composed of 250 websites, of which more than 20,000 elements can be annotated. To conclude the paper, different versions of the YOLO model have been evaluated to identify these elements, obtaining good results with YOLOv9.

[a] https://orcid.org/0009-0001-3877-527X
[b] https://orcid.org/0000-0001-7125-9421
[c] https://orcid.org/0000-0002-1949-1298
[d] https://orcid.org/0000-0001-6024-9527

Dicu, M., González, E. G., Chira, C. and Villar, J. R.
GenGUI: A Dataset for Automatic Generation of Web User Interfaces Using ChatGPT.
DOI: 10.5220/0013177200003890
Paper published under CC license (CC BY-NC-ND 4.0)
In *Proceedings of the 17th International Conference on Agents and Artificial Intelligence (ICAART 2025) - Volume 3*, pages 707-714
ISBN: 978-989-758-737-5; ISSN: 2184-433X
Proceedings Copyright © 2025 by SCITEPRESS – Science and Technology Publications, Lda.

The paper is organized as follows: Section 2 reviews existing datasets and their characteristics. Section 3 introduces our approach for automatically generating web user interfaces using ChatGPT. Section 4 describes the labeling process and dataset features. Section 5 presents the experimental analysis of computer vision models trained on the dataset. Finally, Section 6 summarizes the findings and outlines future work.

2 EXISTING DATASETS

When searching the literature, we found that there are not many published works that present datasets to address the problem of element detection in user interfaces. We can mainly find several contributions that are the most relevant and used by other authors.

One of the most widely used datasets is RICO (Deka et al., 2017), which contains over 72,000 screenshots of Android mobile applications. It identifies a broad range of interface elements, making it a foundational resource for UI element detection. However, RICO has a major drawback: all the screenshots are configured in English, which limits the dataset's generalizability to multilingual contexts. Building on RICO, the Enrico dataset (Leiva et al., 2020) refines the labeing of 10,000 images to improve annotation quality. Despite this enhancement, Enrico inherits RICO's limitations in terms of scope and diversity.

To expand beyond RICO's focus, the VINS dataset (Bunian et al., 2021) includes images from Android and iOS applications. While this improves device variety, VINS is significantly smaller, containing only 4,543 images and covering fewer element types. Like its predecessors, VINS also includes only English-language interfaces, limiting its use in multilingual environments where UI layouts often vary with language.

A different approach is introduced by the UICVD dataset (Dicu et al., 2024b), which shifts focus from mobile applications to websites. This dataset consists of images from 121 websites, marking a move towards web-based interfaces. While UICVD provides valuable insights into website UI elements, it shares the same limitations as RICO, Enrico, and VINS—all the content is exclusively in English. Table 1 summarizes the most relevant characteristics studied in these datasets.

While these datasets are valuable, they exhibit several limitations that reduce their applicability in broader contexts. First, they primarily focus on mobile applications, neglecting desktop applications, which are typically more complex and feature advanced UI elements such as multi-window interactions and toolbars. Second, the exclusive use of English in these datasets limits their generalizability to multilingual environments. In multilingual contexts, interface layouts and element structures can vary significantly based on the language, further limiting the effectiveness of models trained on English-only datasets.

To address these gaps, our work proposes the creation of a new dataset that focuses on desktop applications and incorporates diverse languages. This expansion is crucial for improving the robustness of UI element detection models, ensuring they can perform effectively in more varied, real-world scenarios. By covering these previously neglected areas, our dataset will offer a more comprehensive resource for training and evaluating models in user interface recognition.

3 AUTOMATIC GENERATION OF WEB USER INTERFACES

The primary goal of this work is the automatic generation of user interfaces. The dataset generation process consists of two main phases. In the first phase, ChatGPT is used to automatically generate the source code for multiple user interfaces. In the second phase, each generated interface is opened, and a screenshot is captured for further use.

The most relevant part of the first part is the generation of the source code of the user interfaces. For this first part, ChatGPT was used with the GPT-4o model (OpenAI, 2024) to generate the code for the GenGUI dataset. The decision to generate code using ChatGPT (specifically, the GPT-4o model) instead of relying on pre-existing web templates was driven by several factors. One of the primary reasons is that while many publicly available templates may appear visually similar, the underlying code structures can vary significantly. This inconsistency in code structure can create issues when trying to build a unified dataset for UI element detection. After analyzing multiple web templates, we observed variations in the way HTML, CSS, and JavaScript were implemented, which could complicate efforts to create a cohesive dataset suitable for training machine learning models. By using ChatGPT to generate the code, we ensured that the structure remained consistent across all generated interfaces, adhering to our exact requirements in terms of layout, functionality, and visual diversity. Furthermore, generating the code ourselves eliminated concerns over licensing issues and provided the flexibility to tailor designs to our specific needs, including multilingual support and custom UI components.

Table 1: Summary of the characteristics of the similar datasets studied.

Dataset	Number of images	Number of classes	Language	Device	Reference
RICO	∼72000	23	English	Android	(Deka et al., 2017)
UICVD	121	16	English	PC / Websites	(Dicu et al., 2024b)
VINS	4543	12	English	Android+iOS	(Bunian et al., 2021)
Enrico	10000	23	English	Android	(Leiva et al., 2020)

The conversation with this automatic generation model always started with the prompt: *"Use Bootstrap to create a web page for a fictitious company. The company intends to use it A. Make sure the website includes a navigation bar B. You can include some of the following elements on the site: Icons, Text Fields, Buttons, Images, Table, Containers, Section Title, Search bar, Checkboxes, Dropdown, Radio buttons, Text areas, Forms, Graphs. Support as many interface elements as possible with icons from the Font Awesome library. Make the page content in C. Replace placeholder images with real images. In the output include only the HTML code, you don't need to explain anything"*.

The prompt to initiate the conversation has a series of gaps –A, B, and C–, which will be filled with different content to generate varied user interfaces. Table 2 shows the different values that have been used to fill in the prompt gaps. Once a chat conversation with the generation model is started, the statements *"Generate me another site with the same characteristics, but a different look and feel"*, *"Generate another site that looks different"* and *"Generate another different website"* are used to request the source code of more web sites, until ten web sites with similar characteristics, but different look and feel, are obtained.

In terms of content structure, GPT-4o occasionally favored content-heavy pages, which might not always be desirable for certain applications. To overcome this, we refined the prompts further to generate web pages with varied densities of content, ranging from simple, clean layouts to more complex, information-rich interfaces. This iterative process of adjusting the prompts and reviewing the generated code allowed us to arrive at a balanced approach for diversity in UI designs.

Once the source code of the user interfaces was generated, a manual inspection was performed to check that there were no interfaces with errors or that were very similar. In the generated websites, few interfaces presented this problem, but a small number of interfaces were eliminated to avoid contaminating the dataset. In addition, during this manual inspection, the images and graphics included in the websites were replaced with images and graphics generated by Bing Image Creator and MS Excel, as some websites had been generated with a gray image to mark the site where an image should be included. Bing Image Creator was used for the more decorative images with prompts such as *"Generate me an image of a real person"*, *"Generate me an image of an office"* or *"Generate me an image of a marketplace"*. MS Excel was used for more business-oriented sites. In Excel, a matrix of random numbers was generated and many different types of graphs were drawn with those numbers.

Once the source code of the websites was available and the manual inspection had been done to eliminate errors and replace images, we moved on to the second phase of the generation of the user interfaces for the GenGUI dataset, which consists of opening the web interfaces and taking a screenshot of them, since the dataset will be composed of images, not code. As the user interfaces were developed as web interfaces, Firefox (Mozilla Foundation, 2024) and Selenium (SeleniumHQ, 2024) were used for this part. Selenium is a web testing framework that allows the control of the Firefox web browser to be automated. This phase consisted of opening a website in Firefox, extracting a screenshot of the complete site using Selenium, and moving on to do the same with the following image until the complete dataset was processed.

4 DATASET CREATION

As described in the previous section, the process of generating user interfaces was automated using a GPT-4o model and the Bootstrap framework. Screenshots of these interfaces were then used to create a dataset aimed at training computer vision models. The dataset creation involved several stages: automatic interface generation, the elimination of inappropriate or redundant images, and finally, manual annotation of the individual elements. In this section, we detail the annotation process, the criteria used for labeling, and the final structure of the dataset.

4.1 Element Annotation

To ensure a high-quality dataset, every image from the generated interfaces was manually annotated. Although we had access to the HTML code of these in-

Table 2: Different options to fill in the gaps in the prompt used to start a conversation with ChatGPT. The different options are separated by the / character.

A	Promote the company and publicize its services / Promote a product / Manage the company internally / as an intranet for employees to carry out tasks such as consulting payroll and requesting days off.
B	Horizontal / Lateral
C	English / Spanish / Romanian / German / French / Italian / Dutch / Swedish / Norwegian / Portuguese

terfaces, we opted for manual annotation because the HTML structure does not always accurately reflect how elements are visually displayed on the screen. Specifically, the visual layout can be influenced by CSS styles and JavaScript and dynamic or hidden elements can complicate the automatic annotation process.

We chose manual annotation to guarantee accuracy, particularly because a study conducted by (Dicu et al., 2024a) demonstrated that manual annotations yield better results for training visual detection models, especially when dealing with complex elements like icons. This approach also provides a solid foundation for potential automation in the future. Additionally, manual annotation allowed us to correct errors and establish clear criteria for labeling. Each visual element was identified and labeled according to a well-defined classification, covering both visual and functional aspects.

Figure 1 presents an example from the dataset, showing both the raw interface image and its annotated version, where each visual element is correctly labeled.

4.2 Annotation Criteria

To establish a coherent and unified annotation process for the GenGUI dataset, we developed a set of specific criteria. These criteria focused on the following aspects:

- **Accuracy in Label Positioning.** We aimed to place labels as close as possible to the corresponding visual elements, ensuring that they were delineated and did not interfere with other elements in the interface.
- **Granularity.** We sought to distinguish elements not only based on their visual characteristics but also their functionality.
- **Functional Context.** Some elements may serve multiple roles. For example, each button was labeled accordingly, but internal elements, such as text or icons within the button, were annotated separately. This allows vision models to distinguish between the different visual components within a single functional element.

We used LabelStudio (Tkachenko et al., 2022), an open-source data annotation platform, which allowed us to perform manual labeling in line with the established criteria.

4.3 Dataset Structure

The final dataset consists of 250 PNG images with variable resolutions. The width of all images is fixed at 1920 pixels, while the height ranges from 533 to 3285 pixels. This variation in height enabled us to better simulate real-world scenarios, where user interfaces can have different dimensions depending on the content. We eliminated certain images that either contained errors during generation or featured elements that were difficult to annotate, ensuring a high-quality dataset. Although this process resulted in an unequal number of images per language, we considered this necessary to achieve relevant experimental results.

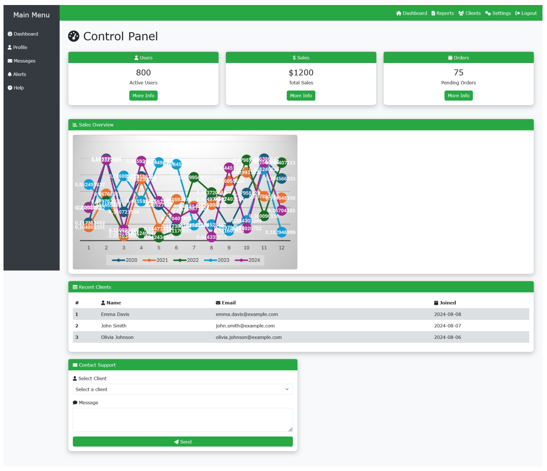
After the elimination process, the dataset contains a total of 250 images and 20,484 annotations, distributed across 13 main classes and 29 subclasses, as detailed in Table 3:

We chose these classes and subclasses to reflect the diversity and complexity of graphical components found in user interfaces. The dataset includes essential elements such as images, text, buttons, and icons, as well as more complex components like input fields, menus, and tables. This classification allows for a greater level of granularity in detecting and classifying elements, ensuring that the dataset can be used for a wide range of computer vision applications.

The decision to divide elements into classes and subclasses was driven by the need to cover as many scenarios as possible in modern graphical interfaces. For instance, text appears in various forms and functions—from titles to text buttons or menu sections—which is why we introduced several subclasses to capture these variations. Similarly, we differentiated icons from other visual elements to offer more precision in the annotation process.

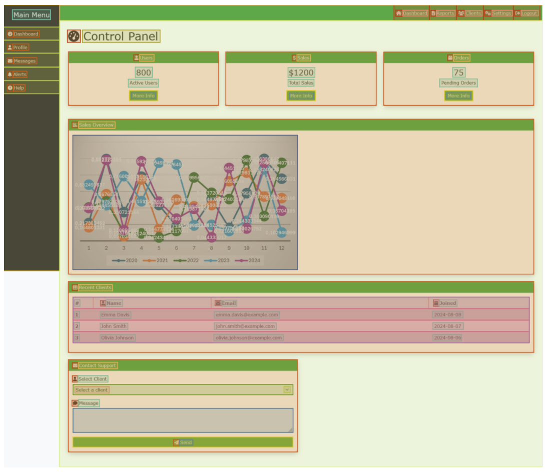
5 EXPERIMENTS AND RESULTS

To validate the quality and utility of the created dataset, we conducted a series of experiments using



#	Name	Email	Joined
1	Emma Davis	emma.davis@example.com	2024-08-08
2	John Smith	john.smith@example.com	2024-08-07
3	Olivia Johnson	olivia.johnson@example.com	2024-08-06

(a) Image 29 from dataset



(b) The annotated Image 29 from dataset

Figure 1: Example from the dataset: the first image shows the raw user interface, while the second image illustrates the same interface with manually annotated elements.

two of the most advanced object detection models, YOLOv8 (Jocher et al., 2023) and YOLOv9 (Wang et al., 2024) . The purpose of these experiments was to observe how these models perform on the 13 main classes in our dataset, providing a concrete evaluation of their effectiveness in detecting graphical user interface elements. Additionally, these experiments will help identify potential limitations and areas for improvement in future expansions of the dataset.

5.1 You Only Look Once (YOLO) Models

The YOLO (You Only Look Once) architecture (Redmon et al., 2016) has transformed object detection by combining speed and accuracy in a single-stage process. By dividing the input image into a grid and predicting bounding boxes and class probabilities simultaneously, YOLO enables real-time detection, making it a widely used approach for a variety of applications.

YOLOv8 (Jocher et al., 2023) improves on earlier versions by enhancing accuracy with the C2f module, particularly for detecting small or overlapping objects. Its anchor-free design and decoupled head optimize detection, classification, and regression tasks, ensuring precision without compromising speed.

YOLOv9 (Wang et al., 2024) extends these advancements by incorporating attention mechanisms and Feature Pyramid Networks (FPN) for better detection across various object sizes. It also uses a hybrid training strategy, combining supervised and unsupervised learning, which improves performance on datasets with limited labeled data.

We selected these models for their balance of speed and accuracy, as well as their effectiveness in detecting small and overlapping objects—key in graphical interfaces with buttons, icons, and text fields. Testing on our dataset evaluates performance and highlights areas for improvement, particularly in class balance and diversity, providing a strong foundation for future work.

5.2 Experimental Setup and Evaluation Metrics

To objectively compare the performances of YOLOv8 and YOLOv9, we used the same parameters across both experiments, employing the default versions of these models without major modifications, as the goal was to assess their general performance on our dataset. Both models were trained for 100 epochs with a batch size of 2, adapted to the relatively small size of our dataset. We used SGD (Stochastic Gradient Descent) as the optimizer, with a learning rate of 0.01 and an image size of 1024x1024 pixels. The image augmentations applied were the default ones provided by the YOLO framework, ensuring consistency in performance evaluation.

The experiments were conducted on the Google Colaboratory (Bisong and Bisong, 2019) platform, utilizing a T4 GPU, which enabled efficient processing of the dataset and training of the models over multiple epochs.

In terms of performance evaluation, we used several key metrics. IoU (Intersection over Union), with a fixed value of 0.5, was employed to measure the overlap between the predicted and actual bounding

Table 3: Distribution of annotations across classes and subclasses in GenGUI dataset.

Class	Number of Annotations	Subclass	Number of Annotations
Text	8927	text	5231
		sectionTitle	1227
		textForNavigationBar	889
		textForButton	792
		textForSidebar	335
		title	209
		statusLabel	125
		textForDropdown	119
Icon	4566	icon	4331
		dropdownIcon	121
		checkBox	71
		radioButton	43
Container	1370	container	1370
MenuItem	1227	navigationItem	893
		sideMenuItem	334
Button	1036	button	1036
InputField	771	inputField	582
		dropdown	120
		datePicker	69
TableColumn	754	tableColumn	754
Row	718	row	550
		tableHeader	168
Menu	309	navigationBar	247
		sideBar	62
WorkingArea	250	workingArea	250
Image	228	image	122
		graph	106
Table	168	table	168
Footer	160	footer	160

boxes. Additionally, we calculated AP (Average Precision) for each class based on the precision-recall curve and mAP (mean Average Precision) to provide an overall performance measure across all classes. The models were evaluated using a single confidence threshold of 0.25, ensuring consistent filtering of predictions.

5.3 Results

The primary goal of the experiment is to assess the performance of YOLOv8 and YOLOv9 by training them on the 13 main classes of the dataset. We focused on the main classes to gain a general understanding of how the dataset performs in object detection and to simplify the analysis of the results.

The data was randomly split into three subsets: 80% for training, 10% for validation, and 10% for testing. This split ensures that the models have sufficient data for learning while allowing for a representative evaluation. Table 4 presents the distribution of images and annotations across the dataset.

Following the experiments on our dataset, we obtained the performances shown in Tables 5 and 6 for the YOLOv8 and YOLOv9 models.

The experimental results reveal a significant performance gap between YOLOv9 and YOLOv8, with YOLOv9 achieving a mAP of 57.78%, nearly double that of YOLOv8's 30.44%. This demonstrates YOLOv9's ability to process the variability and complexity of our dataset more effectively.

In general, YOLOv9 performed much better across all classes, especially in classes that require high detection accuracy, such as Text, Icon, and Button. The strong performance in these classes can be explained by their high representation in the dataset, with a large number of annotations providing more learning opportunities for the models, leading to better generalization. For example, in the Text class, YOLOv9 achieved an impressively high AP, demonstrating the model's ability to handle different types of text in graphical interfaces, such as titles or button labels.

In contrast, both models struggled with underrepresented classes like Footer, Row, and WorkingArea, where AP scores were low or nonexistent. The limited number of annotations and the contextual variability of these elements likely contributed to the

Table 4: Distribution of images and annotations across training, validation, and test sets.

Number of Images				Number of Annotations			
Total	**Train**	**Validation**	**Test**	**Total**	**Train**	**Validation**	**Test**
250	200	25	25	20484	16480	2082	1922

Table 5: Results of YOLOv8 and YOLOv9 models trained on the dataset (Part 1), showing the mean Average Precision (mAP) and Average Precision (AP) per class. The models were evaluated using a confidence threshold of 0.25 and an Intersection over Union (IoU) of 0.5.

Model	mAP	Button	Container	Footer	Icon	Image	InputField
YOLOv8	30.44	65.71	26.84	0.00	74.68	70.13	3.27
YOLOv9	57.78	93.97	57.60	0.00	95.66	79.31	90.00

Table 6: Results of YOLOv8 and YOLOv9 models trained on the dataset (Part 2), showing the mean Average Precision (mAP) and Average Precision (AP) per class. The models were evaluated using a confidence threshold of 0.25 and an Intersection over Union (IoU) of 0.5.

Model	Menu	MenuItem	Row	Table	TableColumn	Text	WorkingArea
YOLOv8	3.03	36.12	0.00	0.00	31.15	84.64	0.21
YOLOv9	9.92	86.87	0.00	63.36	77.05	96.51	0.83

weak performance, as the models lacked sufficient data to learn their distinctive features. A particularly notable example is the InputField class, where YOLOv9 achieved strong results while YOLOv8 underperformed, highlighting YOLOv9's ability to handle complex visual contexts more effectively.

These findings highlight the importance of addressing annotation imbalance and improving class diversity to enhance detection accuracy and model generalization. Underrepresented classes such as Footer, Row, and WorkingArea require additional attention, as their limited presence impacts the models' ability to learn their distinct features effectively.

The current dataset represents a solid foundation for detecting elements in desktop graphical interfaces. However, its class imbalance reflects the natural distribution of these elements in real-world applications. To build a more comprehensive and balanced resource, it will be necessary to expand the dataset with a broader range of images and annotations, particularly for rarer elements. This effort will not only improve detection performance for underrepresented classes but also strengthen the models' ability to generalize across diverse scenarios and graphical applications.

6 CONCLUSIONS AND FUTURE WORK

This paper introduces GenGUI, a new dataset for detecting elements in graphical user interfaces. Accurate detection of these elements is essential for automating the visualization and processing of user interfaces. Generated using the GPT-4o model and Bootstrap framework, GenGUI includes a variety of visual elements, such as text, buttons, icons, and input fields. Unlike existing datasets, which focus primarily on mobile interfaces and English-language content, GenGUI includes diverse desktop interfaces in multiple languages, addressing a major gap in the literature.

Experiments with YOLOv8 and YOLOv9 demonstrate the dataset's effectiveness in identifying UI elements, though challenges remain for underrepresented classes like WorkingArea and Footer. Expanding the dataset and increasing annotations for these classes will help address these issues.

In the future, we plan to expand the dataset by adding more diverse and complex interfaces, along with a wider variety of graphical elements, to better capture real-world scenarios. We also aim to develop an automated labeling method based on the current annotations, which will serve as a reliable ground truth and help reduce the need for manual work. By improving the dataset and streamlining the annotation process, we hope to create a more valuable and practical resource for researchers and developers, contributing to the advancement of graphical interface element detection and understanding.

The dataset is available at the following link: https://github.com/MadaDicu/GENGUI

ACKNOWLEDGEMENTS

The authors, Enol García González and José R. Villar, acknowledge support from the Spanish Ministry of Economics (PID2020-112726RB-I00), the Spanish

Research Agency (PID2023-146257OB-I00), Principado de Asturias (SV-PA-21-AYUD/2021/50994), the Council of Gijón, and Fundación Universidad de Oviedo (FUO-23-008, FUO-22-450).

REFERENCES

Bielik, P., Fischer, M., and Vechev, M. (2018). Robust relational layout synthesis from examples for android. 2(OOPSLA).

Bisong, E. and Bisong, E. (2019). Google colaboratory. *Building machine learning and deep learning models on google cloud platform: a comprehensive guide for beginners*, pages 59–64.

Bunian, S., Li, K., Jemmali, C., Harteveld, C., Fu, Y., and El-Nasr, M. S. (2021). Vins: Visual search for mobile user interface design.

Chen, C., Su, T., Meng, G., Xing, Z., and Liu, Y. (2018). From ui design image to gui skeleton: A neural machine translator to bootstrap mobile gui implementation. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pages 665–676.

Deka, B., Huang, Z., Franzen, C., Hibschman, J., Afergan, D., Li, Y., Nichols, J., and Kumar, R. (2017). Rico: A mobile app dataset for building data-driven design applications. UIST '17, page 845–854, New York, NY, USA. Association for Computing Machinery.

Dicu, M., González, E. G., Chira, C., and Villar, J. R. (2024a). The impact of data annotations on the performance of object detection models in icon detection for gui images. In *International Conference on Hybrid Artificial Intelligence Systems*, pages 251–262. Springer.

Dicu, M., Sterca, A., Chira, C., and Orghidan, R. (2024b). Uicvd: A computer vision ui dataset for training rpa agents. In *ENASE*, pages 414–421.

Jocher, G., Chaurasia, A., and Qiu, J. (2023). YOLO by Ultralytics. https://github.com/ultralytics/ultralytics. Accessed: June 20, 2024.

Leiva, L. A., Hota, A., and Oulasvirta, A. (2020). Enrico: A high-quality dataset for topic modeling of mobile UI designs. In *Proc. MobileHCI Adjunct*.

Miñón, R., Moreno, L., and Abascal, J. (2013). A graphical tool to create user interface models for ubiquitous interaction satisfying accessibility requirements. *Univers. Access Inf. Soc.*, 12(4):427–439.

Moran, K., Bernal-Cárdenas, C., Curcio, M., Bonett, R., and Poshyvanyk, D. (2020). Machine learning-based prototyping of graphical user interfaces for mobile apps. *IEEE Transactions on Software Engineering*, 46(2):196–221.

Mozilla Foundation (2024). *Mozilla Firefox*. Web browser, Version 118.

Nguyen, T. A. and Csallner, C. (2015). Reverse engineering mobile application user interfaces with remaui (t). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 248–259.

OpenAI (2024). Chatgpt (october 2024 version). https://openai.com/chatgpt. Large language model.

Qian, J., Shang, Z., Yan, S., Wang, Y., and Chen, L. (2020). Roscript: a visual script driven truly non-intrusive robotic testing system for touch screen applications. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, ICSE '20, page 297–308, New York, NY, USA. Association for Computing Machinery.

Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788.

Reiss, S. P. (2014). Seeking the user interface. In *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, ASE '14, page 103–114, New York, NY, USA. Association for Computing Machinery.

SeleniumHQ (2024). *Selenium WebDriver*. Testing framework for web applications.

Tkachenko, M., Malyuk, M., Holmanyuk, A., and Liubimov, N. (2020-2022). Label Studio: Data labeling software. Open source software available from https://github.com/heartexlabs/label-studio.

Wang, C.-Y., Yeh, I.-H., and Liao, H.-Y. M. (2024). Yolov9: Learning what you want to learn using programmable gradient information. *arXiv preprint arXiv:2402.13616*.

White, T. D., Fraser, G., and Brown, G. J. (2019). Improving random gui testing with image-based widget detection. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2019, page 307–317, New York, NY, USA. Association for Computing Machinery.

Xiao, S., Chen, Y., Song, Y., Chen, L., Sun, L., Zhen, Y., Chang, Y., and Zhou, T. (2024). UI semantic component group detection: Grouping UI elements with similar semantics in mobile graphical user interface. *Displays*, 83(102679):102679.

Yeh, T., Chang, T.-H., and Miller, R. C. (2009). Sikuli: using gui screenshots for search and automation. In *Proceedings of the 22nd Annual ACM Symposium on User Interface Software and Technology*, UIST '09, page 183–192, New York, NY, USA. Association for Computing Machinery.

Zhang, X., de Greef, L., Swearngin, A., White, S., Murray, K., Yu, L., Shan, Q., Nichols, J., Wu, J., Fleizach, C., Everitt, A., and Bigham, J. P. (2021). Screen recognition: Creating accessibility metadata for mobile applications from pixels. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, CHI '21, New York, NY, USA. Association for Computing Machinery.