Deep Learning-Based UI Design Analysis: Object Detection and Image Retrieval Using YOLOv8

Roba Alghamdi, Adel Ahmad, Fawaz alsaadi
Department of Information Technology, King Abdulaziz University, Jeddah, Saudi Arabia

***Abstract*—Data-driven design models support various types of mobile application design, such as design search, promoting a better understanding of best practices and trends. Designing the well User Interface (UI) makes the application practical and easy to use and contributes significantly to the application's success. Therefore, searching for UI design examples helps gain inspiration and compare design alternatives. However, searching for relevant design examples from large-scale UI datasets is challenging and not easily stricken. The current search approaches rely on various input types, and most of them have limitations that affect their accuracy and performance. This research proposed a model that provides a fine-grained search for relevant UI design examples based on UI screen input. The proposed model will contain two phases. Object detection was implemented using the deep learning model 'YOLOv8', achieving 95% precision and 97% average precision. Image retrieval, leveraging the cosine similarity technique to retrieve the top 3 images similar to the input. These results highlight the system's effectiveness in accurately detecting and retrieving relevant UI elements, providing a valuable tool for UI designers.**

***Keywords*—Data-driven design; YOLOv8; design search; deep learning; user interface design**

I. Introduction

Applications and mobile devices play a crucial role in the daily lives of individuals worldwide, facilitating a wide range of tasks, from simple calculations to more complex operations. Developers adhere to established guidelines and standards before releasing applications on digital marketplaces such as Apple's App Store and Google Play [1]. The development begins with defining requirements and designing the user interface, followed by creating mockups of the graphical user interface (GUI). UI/UX designers iteratively refine these mockups until a final design is achieved, which is then translated into a functional application. After undergoing rigorous testing, the application is released to users, emphasizing the increasing focus on mobile application quality [1].

The user interface (UI) is a fundamental element in determining the success of a mobile application, as it provides an interactive environment for users to engage with the software. The quality of UI design significantly influences user experience, acceptance, and overall app success [2]. In the highly competitive mobile application market, the UI design and app icons play a key role in differentiating an application from competitors, attracting downloads, minimizing user complaints, and enhancing retention rates. A well-designed UI balances visual appeal, efficiency, and ease of use, considering factors such as color harmony, layout organization, and overall design style [3].

Visual composition in UI design is a fundamental aspect of software development. The design process typically starts with wireframing based on user requirements and then structuring visual elements to ensure optimal interaction between the user and the application. Designers iteratively refine these wireframes by referencing existing online examples before applying high-fidelity visual effects, such as colors and typography, and incorporating relevant text and imagery [4].

Data-driven design models contribute to the development of mobile applications by predicting design performance and identifying the best practices and trends [5]. These models support various aspects of mobile application design, including interaction modeling, design search, and UI code generation. Recent advancements in machine learning and data analytics have significantly transformed UI design, allowing designers to explore extensive collections of UI designs based on specific criteria such as layout, color schemes, and functionality [5]. As technology advances, these tools will continue empowering designers to create innovative and user-centered UIs.

Leading mobile application marketplaces offer over six million applications, with projected revenue exceeding $935 billion by 2024—nearly twice the revenue generated in 2020 [6], [7]. The graphical user interface (GUI) is a core component of application success, serving as the interaction point between users and the application's functionalities. Well-designed GUIs go beyond aesthetics, improving usability and enhancing user satisfaction. With increasing competition in app marketplaces, creating engaging and intuitive interfaces has become a top priority for developers. However, designing high-quality GUIs remains challenging and labor-intensive, requiring extensive testing and iteration to ensure usability [8].

For novice and experienced designers, navigating the design space efficiently remains difficult. Seeking design examples has become essential for gaining inspiration and understanding UI trends for specific application functions. However, locating relevant examples within large-scale UI datasets is challenging due to the time-consuming nature of random searches, which may not provide accurate insights into modern UI trends, including layout options, visual components, and effects. An effective solution involves developing approaches that retrieve similar UI designs from large datasets using deep learning technologies. These methods transform GUI development by enabling efficient retrieval and comparison of visual components across extensive datasets [5].

Various studies have attempted to address the issue of UI design retrieval using different input methods such as keywords, sketches, wireframes, and UI images. Most existing studies rely on keyword-based input and often yield irrelevant examples due to mismatched user requirements. A more effective approach is needed to provide designers with practical examples that align with their needs. Research has

explored input-based image retrieval that considers UI content, hierarchical structure, and visual layout. However, current methods face challenges in performance, generalization, and applicability to diverse UI designs, affecting their accuracy and usability. For example, Swire [9] has a limitation in the approach performance because it retrieves irrelevant UI images that do not consider the UI content. The approach proposed by Deka et al. [6] retrieves similar results based on text and image content only.

In contrast, the approach of Liu et al. [10] limits the generalization and precludes the approach from working on any new unseen images because it relies on specified UI content. While others are limited in detecting small objects in UIs, such as icons and checkboxes, the VINS [11] approach restricted its application to specific UIs with a specified set of components. Detecting objects at different scales is challenging, particularly small objects. Data augmentation techniques are ways to solve the problem of detecting small objects on UI screens. These techniques are utilized to ensure the dataset does not lack sufficient training data or uneven class balance within the datasets and are adopted as an effective solution to improve the performance of object detection models. Many recent studies have shown that combining the YOLOv8 model with multiple data augmentation strategies led to significant improvements in accuracy and other performance metrics, as used in study [12]. Similar techniques were applied to improve the accuracy of brain tumor detection using MRI images, where data augmentation significantly increased brain tumor detection accuracy, improved the model's generalization ability, and reduced errors. This study confirms the effectiveness of data augmentation techniques in improving the performance of YOLOv8. This study highlights how to propose a practical model capable of efficiently detecting the most common UI components and retrieving UI images using deep learning and computer vision techniques. It also investigates the integration with custom data augmentation techniques to improve detection performance and address these limitations.

This research aims to overcome the limitations of previous methods by introducing a fine-grained UI search system that retrieves similar UI designs based on a given UI screen. The proposed system leverages deep learning techniques to help designers quickly understand design spaces, draw inspiration from existing applications, and enhance their UI designs to ensure application success. The primary objectives of this study include:

- Developing a model capable of retrieving relevant UI designs from large-scale datasets based on input images using deep learning and computer vision techniques.
- Improving accuracy in retrieving relevant UI examples by refining the search process to better align with designer needs.
- Evaluating the proposed model against the existing approach "VINS" to assess its effectiveness in enhancing the UI design retrieval process.

This study contributes to developing and evaluating a framework for fine-grained UI search. The YOLOv8 model was chosen for this study due to its architectural and technical improvements, which make it suitable for detecting user interface elements within images. Appreciation to its improved accuracy, faster performance, and ability to recognize objects of various sizes compared to previous versions, it can extract deeper and more effective features, making it the preferred choice, as demonstrated in this study [13]. YOLOv8 efficiently recognizes objects even in complex or diversely designed images, making it suitable for analyzing user interfaces containing multiple, closely spaced elements.

The proposed model integrates deep learning (YOLOv8) and computer vision techniques (Cosine Similarity) to effectively detect and retrieve highly similar UI designs with high precision. This work presents the first model that integrates these technologies within this domain, which makes it an advantage for this work. A new dataset was also created from the Rico dataset, incorporating preprocessing techniques and categorization into 21 classes—a novel contribution in this area. Data augmentation techniques were applied to ensure the dataset does not lack sufficient training data or uneven class balance within the datasets, resulting in 19,000 UI images. The proposed model surpasses the baseline model by 12% in UI image search accuracy [9]. This contribution highlights the proposed model's effectiveness and potential impact on advancing UI design research.

The rest of the paper is structured as follows: the "Related Work" in Section I examines previous studies, while the "Materials and Methods" in Section II describes the proposed methodology and dataset. The "Experiments" in Section III presents the experimental setup and findings. Lastly, the "Conclusion and Future Work" in Section IV highlights the principal results and suggests possible future research directions.

II. Related Work

Various approaches have been proposed in UI design retrieval, utilizing different input types to enhance search efficiency and accuracy. Traditional keyword-based search methods have evolved into more advanced techniques, such as natural language queries, image-based searches, and deep learning-driven retrieval models. Studies have compared these methods based on usability, effectiveness, and retrieval accuracy, showcasing significant improvements with deep learning techniques. Cardenas et al. [5] introduced GUIGLE, a framework for GUI search that facilitates the conceptualization process for UI design. GUIGLE enables advanced searches using natural language queries, incorporating UI components, on-screen text, color schemes, and application names. The framework consists of three main components: data collection, quality filtering, and indexing, achieving a 68.8% retrieval relevance rate. Chen et al. [14] proposed Gallery D.C, a large-scale UI component gallery that leverages computer vision techniques and reverse engineering. This system categorizes 11 UI components across 25 Android application categories and provides search, comparison, and summarization tasks. It employs Faster R-CNN for UI element detection, demonstrating superior design sharing and information retrieval performance. The main limitation of using keywords is retrieving design examples with UI content and visual layout structure different from user requirements.

Beyond keyword-based searches, researchers have explored image-based retrieval using wireframes, sketches, and UI

screens as input methods, which are faster to specify and easier to learn than keywords. Huang et al. [9] developed Swire, a deep learning-based UI retrieval model trained on a large-scale UI sketch dataset. Swire utilizes VGG-A sub-networks to match UI screens with sketches, achieving a 60% accuracy rate for retrieving relevant UIs. Chen et al. [15] proposed a wireframe-based UI design search engine using deep learning. This approach includes reverse engineering to construct a large-scale UI dataset, encoding visual semantics with a CNN-based autoencoder, and employing KNN for UI design search. The evaluation results demonstrated superior performance compared to existing search methods that rely on different input types. The main drawback of these two models is that they return user interface designs that are generally similar to the query user interfaces, so they may miss some UI components or return irrelevant results. To solve this problem, examples of design that will help designers in practice must be found. Deka et al. [6] introduced the Rico dataset, the largest UI dataset to date, containing 72,219 UI screenshots from 9.7K Android applications across 27 categories. The dataset includes structural, textual, visual, and interactive UI properties, supporting deep learning applications for design retrieval. Rico's autoencoder-based UI layout similarity model demonstrated strong retrieval capabilities using text and image-based content only, which is considered a weakness. Liu et al. [10] extended the Rico dataset by introducing an automated approach for generating semantic annotations, identifying UI components' structural and functional roles. These annotations were applied to the 72,219 UI screens, enabling enhanced UI retrieval through autoencoder-based similarity searches. The model relies on a pre-defined hierarchy of UI content, so it does not work on any new, unseen images. Bunian et al. [11] proposed VINS, a visual search framework for retrieving UI design examples based on wireframes or UI screens. The framework integrates an object detection model using SSD for UI component identification and an attention-based autoencoder for image retrieval, achieving a mean average precision (mAP) of 76% for component detection and up to 90% precision in retrieving similar UI designs. This model's shortcoming is that it only detects 11 classes of UI components and is noticeably unable to detect small objects within the UI. The proposed model tried to solve these shortcomings by generalizing as much as possible to the most significant number of UI components, focusing on improving the detection of small components by applying some augmentation techniques and improving the model's performance to work on any input image.

Sun et al. [16] introduced a UI component recognition model using CNN techniques. The approach involved pre-processing UI images through grayscale conversion, noise removal, segmentation, and CNN-based classification into 14 UI component types. While effective, the model struggled with complex, user-defined UI components and misclassification of similar elements. Nguyen et al. [17] developed REMAUI, a reverse engineering framework for UI design analysis. The system detects UI components and generates static applications using computer vision, Optical Character Recognition (OCR), and mobile-specific heuristics. The primary limitation of this approach lies in its binary classification of UI elements, restricting its ability to distinguish between different component types. Moran et al. [18] proposed a machine learning-based prototype for GUI analysis in mobile applications, incorporating detection, classification, and assembly tasks. The detection phase employs computer vision techniques and OCR to identify GUI components, utilizing edge detection, dilation, and contour bounding boxes to refine object recognition. All of these approaches evaluate detection accuracy by a small number of GUIs.

CNN-based classification further enhances detection accuracy, outperforming previous GUI analysis methods. Recent advancements in deep learning have significantly improved object detection, image classification, and semantic segmentation for UI design retrieval. Faster R-CNN, employed in Gallery D.C, achieved a recall of 0.65, a precision of 0.73, and a mean average precision (mAP) of 0.69 at an Intersection over Union (IoU) threshold of 0.6. Meanwhile, SSD, used in VINS, demonstrated superior performance with an mAP of 76.39% and an Area Under the Curve (AUC) of 79.02% at IoU=0.5. The trade-off between two-stage detectors like Faster R-CNN, which yield higher accuracy but require substantial computational power, and one-stage detectors like YOLO and SSD, which are faster and more efficient for real-time mobile applications, remains a critical consideration in UI retrieval research. Since it is important to focus on performance efficiency in addition to speed, the Yolo model was adopted in the model proposed in this paper.

The Rico dataset, the largest in UI research with 72,000 images, requires refinement to remove duplicates caused by user interaction traces. Previous studies have used limited class annotations, restricting their applicability. While deep learning techniques have improved UI retrieval based on user preferences, challenges remain in performance, generalization, and adaptability. To overcome these issues, a YOLO-based model was proposed that enhances detection by re-annotating the Rico dataset into 21 classes, making UI retrieval more accurate and comprehensive.

III. Materials and Methods

This section describes the methodology and architecture of the proposed fine-grained search system, designed to retrieve relevant UI design examples by integrating deep learning (DL) models. Fig. 1 illustrates the general methodology used in this system:

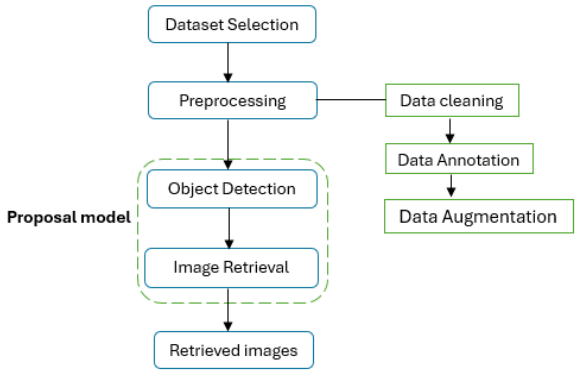

Fig. 1. The proposed system steps.

A. Data Preprocessing

Data preprocessing is a crucial step in machine learning that ensures the dataset's quality, consistency, and usability. Leveraging the Rico dataset as the cornerstone of the research investigation, the researchers meticulously navigate through these preparatory stages to ensure the data's integrity, richness, and relevance. This process consists of three main steps: Data Cleaning, Data Annotation, and Data Augmentation, which refine the dataset for optimal model training.

1) Data cleaning: Since raw datasets often contain inconsistencies, redundant images, and noise, a systematic filtering process was applied to remove incomplete, irrelevant, or low-quality data. The Rico dataset, the largest dataset in UI research with 72,000 UI images, required refinement to eliminate duplicate images caused by user interaction traces. The filtering criteria included:

- Removing images with less than two UI components.
- Eliminating empty or non-English UI screens.
- Removing duplicate UI screens to avoid bias.

Fig. 2 shows examples of deleted images that did not meet the dataset's quality standards.

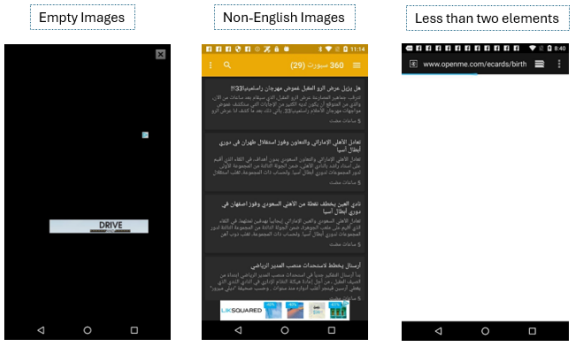

Fig. 2. Examples of deleted photos.

2) Data annotation: To ensure accurate and granular classification, the dataset was annotated into 21 distinct UI component classes using the Roboflow platform. These classes include BackgroundImage, BottomNavigation, Button, Card, Checkbox, Drawer, Edit Text, Icon, Image, Map, Modal, Multi Tabs, Page Indicator, Progress Bar, Radio Button, Seek Bar, Spinner, Switch, Text, Tool Bar, and Upper Task Bar. Each category was assigned a specific color to facilitate visual identification and improve model interpretability.

3) Data augmentation: Multiple augmentation techniques were applied to address class imbalance and enhance dataset diversity, leveraging the Roboflow platform. These included:

- Rotation
- Saturation adjustment
- Brightness modification
- Exposure correction
- Noise addition

This augmentation process tripled the number of instances in underrepresented classes, enhancing the trained model's robustness and generalization capability.

B. Model Architecture

The main objective of this research is to build a fine-grained search model that retrieves similar UI designs depending on a given UI screen that fits the designer query and enhances the detection performance, especially for small objects such as icons, checkboxes, and others. Fig. 3 illustrates the architecture of the model. The proposed fine-grained search model consists of five main phases:

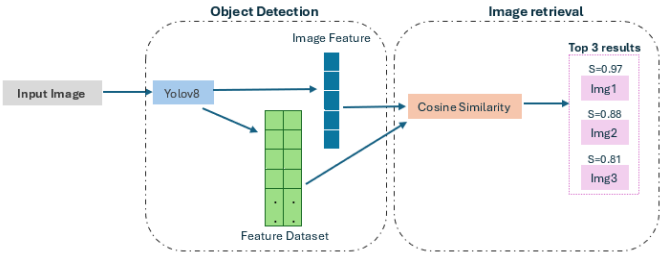

Fig. 3. Model architecture.

1) Object detection: Identifies UI components and their locations using the YOLOv8 [19] model. The YOLOv8 is one of the most advanced deep learning-based object detection models in current use, known for its high accuracy and rapid processing capabilities. The model uses convolutional layers to detect fine details like edges, shapes, and textures, dividing images into regions to identify and classify objects, such as vehicles, pedestrians, and animals, using bounding boxes and trained datasets [20]. The model processes images through the following steps:

- Resizing images to 640x640 pixels for input consistency.
- Normalizing pixel values to [0,1] range for stable neural network processing.
- Applying convolutional layers to extract shapes, edges, and textures.
- Generating bounding boxes with confidence scores to classify detected UI elements.

This methodology enhances model precision and detection speed, making it suitable for real-time UI retrieval applications.

2) Feature extraction: In computer vision, retrieving similar images based on extracted features is a powerful tool that enables applications ranging from automated tagging in media libraries to more complex uses in visual search engines and recommender systems. Extracts UI screen structure features and retrieves similar designs using cosine similarity. After detecting UI components, the YOLOv8 model generates feature vectors representing the spatial structure of each UI screen. Lower layers focus on basic elements, while deeper layers summarize complex structures, condensing these details into a global feature vector representing each image's content [21] [22]. These vectors encode:

- Bounding box coordinates (x, y, width, height).
- Class probabilities and confidence scores.

This feature vector acts as a numerical signature that reduces the complexity of images, allowing for more efficient and computationally manageable comparisons [23]. A global feature vector is derived by calculating the mean of all bounding boxes, creating a compact yet descriptive representation of the UI screen.

3) Feature dataset construction: Extracted feature vectors are stored in a structured database for efficient retrieval. This database transforms raw images into comparable vectors, streamlining retrieval without requiring direct pixel comparisons [24]. The indexing process ensures:

- Fast similarity comparisons across large UI datasets.
- Optimized storage using Python's Pickle serialization to avoid redundant computations.

This feature dataset enables the system to retrieve visually similar UI designs efficiently, supporting large-scale searches.

4) Similarity-based image retrieval: The model employs cosine similarity to retrieve images most similar to a query UI screen. This metric, which measures the cosine of the angle between two vectors, is advantageous as it focuses on the orientation of vectors rather than their magnitudes, making it suitable for high-dimensional data like feature vectors [25]. The similarity between two feature vectors f_i and f_q is calculated using the cosine similarity formula:

$$cosine_similarity(f_i, f_q) = \frac{f_i \cdot f_q}{\| f_i \| \| f_q \|} \quad (1)$$

Where $f_i \cdot f_q$ is the dot product, and $\|f_i\|\|f_q\|$ is the product of their magnitudes. Once similarity scores are computed, the system ranks images based on their cosine similarity scores, retrieving the top-k most similar UI designs [26]. This approach enhances retrieval accuracy and efficiency.

5) Highlighting key regions in retrieved images: To improve interpretability, retrieved UI images are highlighted with bounding boxes indicating significant regions the model detects. This visual emphasis helps users understand which UI elements contributed to the match.

- Bounding boxes are overlaid in distinct colors to enhance visibility.
- Solid-colored highlights (e.g., green rectangles) focus attention on key UI elements.

This method improves user experience and decision-making in UI retrieval applications.

C. Dataset

The Rico dataset is a resource for mobile app design, containing design data from over 9,772 Android apps across 27 categories [6]. It includes over 72,000 unique UI screens, documenting interactive, textual, structural, and visual design elements. The dataset provides user interaction traces, app metadata from Google Play (category, ratings, downloads), and detailed UI components such as buttons, cards, text fields, and icons. It also includes XML annotation files, but for YOLOv8, these need to be converted into TXT files representing bounding boxes with class labels and coordinates. The dataset is cleaned and preprocessed, with 19,727 images classified into 21 categories, averaging 17 annotations per image.

IV. Experiments and Results

Python was used as the primary programming language for the model's experiments due to its extensive support for machine learning and data science libraries. Python's flexibility and wide range of tools allowed us to implement the proposed methodologies efficiently. Given the computational limitations of the local machine, the Python code is executed using Google Colaboratory, a cloud-based platform that provides free GPU access [27].

Due to the large dataset size, the dataset was uploaded to the Kaggle platform which used its library to import and manage it. The Ultralytics library was employed to train the YOLOv8x model, while the sklearn.metrics. A pairwise package was utilized to implement cosine similarity in the image retrieval process. To optimize the proposed object detection model, the IoU (Intersection over Union) threshold is set to 0.7 and the confidence threshold to 0.25. Additionally, the input image size was resized to 700x700 pixels. These hyperparameters were chosen based on empirical experimentation to achieve the best trade-off between accuracy and performance.

A. Data Preprocessing

Data preprocessing is a crucial step that ensures the dataset is optimized for model training. The preprocessing steps included:

- Data Cleaning: Removing noisy or irrelevant annotations
- Annotation Conversion: Transforming XML files into YOLO TXT format
- Data Augmentation: Enhancing dataset diversity and balancing class distribution

For augmentation, the Roboflow [28] utilized a pivotal platform within the realm of computer vision and machine learning. It offers a comprehensive suite of tools tailored to streamline the data preparation pipeline. Roboflow offers various augmentation techniques, including geometric transformations, color adjustments, and specialized augmentations. After augmentation, underrepresented classes saw a threefold increase in image samples, bringing the dataset size to 19,727 images. The final category distribution is shown in Table I.

B. Model Training

The experiment leveraged the YOLOv8 [19] model for object detection to detect various elements within the research dataset. The training process was initiated with a pre-trained YOLOv8x model that utilized the robust capabilities of the Ultralytics framework. The Yolov8 had multiple versions with different parameters, speed, and mAP. The YOLOv8x was selected due to the large dataset and the importance of accuracy

TABLE I. DATASET CATEGORIES DISTRIBUTION

Category	Train	Valid	Test	Sum
BackgroundImage	1660	250	131	2041
BottomNavigation	1619	92	62	1773
Button	14042	2857	1603	18502
Card	5128	927	497	6552
Checkbox	3855	606	346	4807
Drawer	2063	212	131	2406
EditText	5032	954	570	6556
Icon	54602	10747	4873	70222
Image	20502	4429	1967	26898
Map	552	69	37	658
Modal	1466	239	146	1851
MutilTabs	2768	289	153	3210
PageIndicator	3070	112	110	3292
ProgressBar	698	24	33	755
Radiobutton	2112	458	272	2842
SeekBar	967	65	45	1077
Spinner	4952	708	517	6177
Switch	1135	165	161	1461
Text	116927	23407	11147	151481
ToolBar	8710	1825	823	11358
UpperTaskBar	13706	2762	1374	17842

over speed, the most robust version among the YOLOv8 models. The dataset was split into three subsets:

- 70% for training
- 20% for validation
- 10% for testing

This split allowed for efficient model training while preventing overfitting. The dataset was specified in a custom configuration file, and hyperparameter tuning was conducted to optimize model performance.

C. Evaluation Metrics

For the proposed model, the Precision (P) and Average Precision (AP) are used as key metrics to evaluate the model:

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

where: TP (True Positives): Correctly detected objects. FP (False Positives): Incorrectly detected objects. FN (False Negatives): Missed objects.

The Mean Average Precision (mAP), a widely used metric for object detection tasks, was computed as follows:

$$mAP = \frac{\sum AP_n}{N} \tag{3}$$

where AP_n represents the average precision for class n.

D. Results

After training, the YOLOv8x model was tested on the validation set using a confidence threshold of 0.5, meaning only detections with a confidence score of 50% or higher were considered valid. This threshold was chosen to balance the results, reducing false positives while ensuring that true detections were not missed. The model achieved a 95% precision and a 97% average precision for the object detection component in all classes. Table II. presents the class-wise precision and AP values.

TABLE II. PRECISION AND MAP FOR EACH CLASS

Class	P	mAP50
BackgroundImage	0.934	0.976
BottomNavigation	1	0.951
Button	0.968	0.983
Card	0.898	0.974
Checkbox	0.975	0.981
Drawer	0.97	0.991
EditText	0.921	0.96
Icon	0.96	0.969
Image	0.963	0.974
Map	0.939	0.988
Modal	0.951	0.994
MutilTabs	0.969	0.985
PageIndicator	0.929	0.983
ProgressBar	0.949	0.973
Radiobutton	0.987	0.986
SeekBar	0.968	0.945
Spinner	0.921	0.949
Switch	0.956	0.981
Text	0.951	0.973
ToolBar	0.954	0.976
UpperTaskBar	0.965	0.981
ALL	0.954	0.975

For image retrieval, the model successfully identified the three most similar UI designs to the input image using cosine similarity, displaying results as a ranked list of top-matching UIs. The colored bounding boxes in the second image represent each part of the UI query image, each color representing a different component. Fig. 4 shows an example of this step.

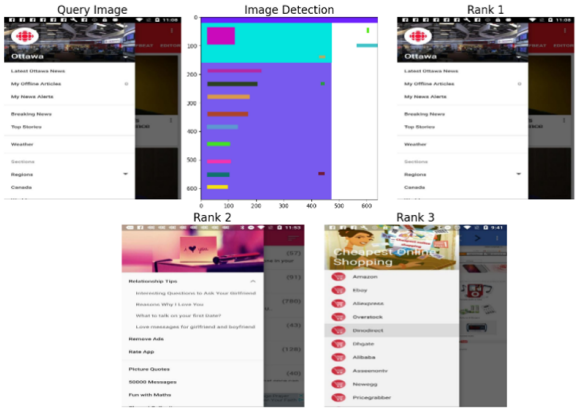

Fig. 4. The Retrieval results of out model.

E. Comparison with Baseline Model

To highlight the improvements of the proposed model, a comparison with the baseline model VINS [9] was made, which combined the SSD model with Autoencoder to build the entire model. The proposed model relied on Yolov8 for feature detection, extraction, and storage and used Cosine similarity to retrieve images similar to the one given. At the same time, we relied on Yolov8 for feature detection, extraction, and storage and used Cosine similarity to retrieve similar images to the given one. Due to the unavailability of the model's source code, it was impossible to reapply the model to the dataset used in this research. Therefore, a rough comparison was made based on the mAP metric described in that study. However, it should be noted that the dataset used in this work is more extensive and diverse in terms of the number of classes. Although a direct comparison is not possible, the proposed system significantly outperforms VINS, achieving an mAP of 97% compared to VINS's 76.39% as a result reported in the literature [9], demonstrating superior accuracy and effectiveness in object detection and UI retrieval. This comparison is approximate and should be interpreted in the context of the different datasets and models used. The comparison results are presented in Table III.

TABLE III. COMPARISON WITH THE BASELINE MODEL

Model	mAP50	Technique used
VINS	76.39%	SSD and Autoencoder
The proposed Model	97%	Yolov8 and Cosine similarity

V. DISCUSSION

The results confirm the effectiveness of YOLOv8 in detecting UI elements. To determine the best version of Yolo models in the dataset, all versions were trained on the data and assigned an epoch value of 20 and a batch of 16. The results in Table IV indicate that YOLOv8x achieved the highest accuracy, making it the optimal choice for the dataset.

TABLE IV. VARIANCE MODELS RESULTS

Model	Precision	mAP50
YOLOv8n	0.783	0.796
YOLOv8s	0.798	0.844
YOLOv8m	0.808	0.843
YOLOv8l	0.803	0.866
YOLOv8x	0.827	0.866

The proposed model successfully learns how to extract relevant features from images and then classifies and locates objects effectively across all images in the dataset. The model was trained on the Revised Rico data, and annotations were made on each image to identify the exact locations of objects within it based on the dataset categories. In addition, augmentation was performed on some under-representation classes to ensure the quality of the model and increase the chance of the model detecting these objects. The results of testing the proposed model using data extracted from the Rico dataset showed high performance in different categories, where the mAP reached 98% of overall categories, indicating that the model outperformed VINS [9] by about 20%. This is consistent with the results reported in the study [21], where the performance of their model also improved. However, the model still faces challenges predicting some categories, such as the "ProgressBar" and "SeekBar." This could be due to the imbalance of the data, which biases the model towards the higher-ranked categories. Even after trying to balance the dataset using a downsampling technique by removing images that collect objects from the majority class to achieve partial convergence with the rest of the classes and applying augmentation techniques to the minority class images, there was still a noticeable difference in the distribution of images between classes. These lower-ranked classes are often associated with some highly-ranked classes, which limits the possibility of creating balanced data.

Overall, this study demonstrates significant advancements in UI detection and retrieval, outperforming previous models and providing an effective tool for analyzing mobile UI designs.

VI. CONCLUSION

In this study, we developed an object detection and image retrieval model leveraging the YOLOv8 framework and cosine similarity to analyze UI components. The dataset, sourced from the Rico dataset, was preprocessed, annotated, and augmented to enhance model performance. TThe hyperparameters were optimized through rigorous experimentation to achieve high precision and accuracy in detecting and classifying UI elements. The results demonstrated the effectiveness of the YOLOv8x model in object detection, outperforming the baseline VINS model by a significant margin. The proposed approach achieved an overall mean Average Precision (mAP) of 97%, compared to 76.39% for the VINS model, highlighting its robustness in accurately identifying UI components.

Additionally, the integration of cosine similarity facilitated efficient image retrieval, allowing the system to suggest visually and structurally similar UI designs. Despite the promising results, some challenges remain, particularly in predicting underrepresented UI components such as "ProgressBar" and "SeekBar." While augmentation and balancing techniques improved performance, disparities in category distribution persisted.

Future work could focus on further dataset balancing strategies, exploring alternative deep learning architectures, and refining feature extraction techniques to enhance retrieval accuracy. Expanding the dataset to include various UI designs from different platforms (e.g., iOS, web applications) and categories could enhance the system's versatility. The model can also be integrated with an Android app, making it easier for designers to leverage a variety of UI designs. The proposed approach represents a significant advancement in UI element detection and retrieval, offering a valuable tool for mobile UI designers and developers to analyze and refine interface layouts efficiently. The study underscores the potential of deep learning techniques in automating UI analysis, paving the way for more intelligent and adaptive design systems.

REFERENCES

[1] Adefris, B. B., Habtie, A. B. and Taye, Y. G. [2022], "Automatic code generation from low fidelity graphical user interface sketches using deep learning", in 2022 International Conference on Information and

Communication Technology for Development for Africa (ICT4DA)", IEEE, pp. 1–6.

[2] Alomari, H. W., Ramasamy, V., Kiper, J. D. and Potvin, G. [2020], "A user interface (ui) and user experience (ux) evaluation framework for cyberlearning environments in computer science and software engineering education", Heliyon 6(5).

[3] Bachmann, D., Weichert, F. and Rinkenauer, G. [2018], "Review of threedimensional human-computer interaction with focus on the leap motion controller", Sensors 18(7), 2194.

[4] Altinbas, M. D. and Serif, T. [2022], "Gui element detection from mobile ui images using yolov5", in International Conference on Mobile Web and Intelligent Information Systems", Springer, pp. 32–45.

[5] Bernal-Cárdenas, C., Moran, K., Tufano, M., Liu, Z., Nan, L., Shi, Z. and Poshyvanyk, D. [2019], "Guigle: A gui search engine for android apps", in 2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)", IEEE, pp. 71–74.

[6] 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", in Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology", pp. 845–854.

[7] Statista Research Department, Mobile app usage [2024], https://www.statista.com/topics/1002/mobile-app-usage/#statisticChapter. Accessed: 12-Desmber-2024.

[8] Chen, J., Xie, M., Xing, Z., Chen, C., Xu, X., Zhu, L. and Li, G. [2020], "Object detection for graphical user interface: old fashioned or deep learning or a combination?", in Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering", pp. 1202–1214.

[9] Huang, F., Canny, J. F. and Nichols, J. [2019], "Swire: Sketch-based user interface retrieval", in Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems", pp. 1–10.

[10] Liu, T. F., Craft, M., Situ, J., Yumer, E., Mech, R. and Kumar, R. [2018], "Learning design semantics for mobile apps, in "Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology", pp. 569–579.

[11] Bunian, S., Li, K., Jemmali, C., Harteveld, C., Fu, Y. and Seif El-Nasr, M. S. [2021], "Vins: Visual search for mobile user interface design", in Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems", pp. 1–14.

[12] R. S. Passa, S. Nurmaini, and D. P. Rini, "YOLOv8 based on data augmentation for MRI brain tumor detection," Scientific Journal of Informatics, vol. 10, no. 3, pp. 363–370, Aug. 2023.

[13] H. Wang, X. Guo, S. Zhang, G. Li, Q. Zhao, and Z. Wang, "Detection and recognition of foreign objects in Pu-erh Sun-dried green tea using an improved YOLOv8 based on deep learning," PLOS ONE, vol. 20, no. 1, Art. no. e0312112, Jan. 2025. [Online]. Available: https://doi.org/10.1371/journal.pone.0312112

[14] Chen, C., Feng, S., Xing, Z., Liu, L., Zhao, S. and Wang, J. [2019], "Gallery dc: Design search and knowledge discovery through auto-created gui component gallery", Proceedings of the ACM on Human-Computer Interaction 3(CSCW), 1–22.

[15] Chen, J., Chen, C., Xing, Z., Xia, X., Zhu, L., Grundy, J. and Wang, J. [2020], "Wireframe-based ui design search through image autoencoder", ACM Transactions on Software Engineering and Methodology (TOSEM) 29(3), 1– 31.

[16] Sun, X., Li, T. and Xu, J. [2020], "Ui components recognition system based on image understanding, in 2020", IEEE 20th International Conference on Software Quality, Reliability and Security Companion (QRS-C)", IEEE, pp. 65–71.

[17] 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)", IEEE, pp. 248–259.

[18] Moran, K., Bernal-Cárdenas, C., Curcio, M., Bonett, R. and Poshyvanyk, D. [2018], "Machine learning-based prototyping of graphical user interfaces for mobile apps", IEEE Transactions on Software Engineering 46(2), 196–221.

[19] Rath, Sovit. [2023], "Yolov8 ultralytics: State-of-the-art yolo models", LearnOpenCV. Retrieved March

[20] Shen, Lingyun and Lang, Baihe and Song, Zhengxun.[2023], "DS-YOLOv8-based object detection method for remote sensing images", Ieee Access, 125122–

[21] M. Talib, A. H. Al-Noori, and J. Suad, "YOLOv8-CAB: Improved YOLOv8 for real-time object detection," Karbala International Journal of Modern Science, vol. 10, no. 1, p. 5, 2024.

[22] J. Farooq, M. Muaz, K. Khan Jadoon, N. Aafaq, and M. K. A. Khan, "An improved YOLOv8 for foreign object debris detection with optimized architecture for small objects," Multimedia Tools and Applications, vol. 83, no. 21, pp. 60921–60947, 2024.

[23] G. Yang, J. Wang, Z. Nie, H. Yang, and S. Yu, "A lightweight YOLOv8 tomato detection algorithm combining feature enhancement and attention," Agronomy, vol. 13, no. 7, Art. no. 1824, 2023.

[24] Y. Yang and J. Wang, "Research on breast cancer pathological image classification method based on wavelet transform and YOLOv8," Journal of X-Ray Science and Technology, Preprint, pp. 1–11, 2024.

[25] D. Wan, R. Lu, B. Hu, J. Yin, S. Shen, and X. Lang, "YOLO-MIF: Improved YOLOv8 with multi-information fusion for object detection in gray-scale images," Advanced Engineering Informatics, vol. 62, Art. no. 102709, 2024.

[26] S. Chen, Y. Li, Y. Zhang, Y. Yang, and X. Zhang, "Soft X-ray image recognition and classification of maize seed cracks based on image enhancement and optimized YOLOv8 model," Computers and Electronics in Agriculture, vol. 216, Art. no. 108475, 2024.

[27] Google [n.d.], 'Colab.google', https://colab.google//. Online; accessed 15-February-2025.

[28] Brucal, Stanley Glenn E and de Jesus, Luigi Carlo M and Peruda, Sergio R and Samaniego, Leonardo A and Yong, Einstein D. [2023], "Development of tomato leaf disease detection using YoloV8 model via RoboFlow 2.0", IEEE 12th Global Conference on Consumer Electronics (GCCE),692–694