Date of publication SEP-30, 2025, date of current version MAY-26, 2025.
www.computingonline.net / computing@computingonline.net

Print ISSN 1727-6209
Online ISSN 2312-5381
DOI 10.47839/ijc.24.3.4183

MASC: A Dataset for the Development and Classification of Mobile Applications Screens

MOHEB R. GIRGIS, ALAA M. ZAKI, ENAS ELGELDAWI, MOHAMED M. ABDALLAH, ALI A. AHMED

Computer Science Department, Faculty of Science, Minia University, El-Minia, Egypt,

Corresponding author: Moheb R. Girgis (e-mail: moheb.girgis@mu.edu.eg)

ABSTRACT Mobile applications (apps) have become an integral part of our daily lives, offering a wide range of functionalities and services. Understanding the diversity of mobile app screens is crucial for optimizing user experience and delivering personalized content. This paper presents a novel dataset, called MASC (Mobile App Screens Classification) consisting of 7065 images, representing various types of mobile apps screens. MASC dataset is collected from the well-known Rico dataset. These screens were carefully manually classified into ten unique classes to capture the diverse nature of app interfaces. Based on the MASC dataset, this paper presents a proposed framework for applying machine learning (ML) algorithms to the classification of mobile apps screens. The paper presents a feature extraction algorithm that extracts, from each screenshot image of an app screen, key characteristics related to visual elements, text, and keywords. Using the proposed framework, the paper also presents a comprehensive study of the classification of mobile apps screens using ML algorithms. Several classification algorithms including XGBoost, Gradient Boosting, Random Forest, SVM, Logistic Regression, and others were trained and evaluated on MASC. Results showed high accuracy scores, above 93%, for top models like Gradient Boosting, indicating that ML algorithms with the MASC dataset provide an effective approach to mobile app screen classification. This study contributes to the field of mobile app analysis and user interface understanding. In addition, the proposed mobile app screens classification framework is a promising development that can enhance the accuracy and efficiency of mobile app screens classification. The complete code is available on GitHub to ensure reproducibility and foster further research: https://github.com/Ali-Aahmed/MASC-Dataset.

KEYWORDS Mobile applications; Activities Classification; UI Screens Classification; MASC Dataset; Wireframes; Machine Learning Algorithms.

I. INTRODUCTION

Mobile applications are an integral part of our daily lives, providing us with convenience, entertainment, and productivity at our fingertips. With the widespread adoption of mobile apps, it has become essential to understand the nuances of different app screens, as their layout, design, and functionality have a significant impact on user experience and engagement Wang, Li [1]. Today, millions of mobile apps are available for download in various app stores, offering a wide range of services, from communication to social media to entertainment to productivity. The development of these apps usually begins with an idea that is presented to a design team, who is responsible for designing both the user experience and the user interface architecture [2].

Android is the most popular mobile operating system in the world, with many devices, users, and rich apps. With the proliferation of 5G and the accelerated research and development of 5G smart devices by major brands, global smartphone shipments are expected to increase slightly in the coming years [1].

The screen type classification along with a screen summary can assist users of screen readers in rapidly forming a conceptual understanding of unfamiliar mobile user interfaces, eliminating the need to wait for the mobile UI reader to painstakingly navigate through each element [3, 4].

Existing UI datasets have some common limitations that hinder comprehensive research on mobile app UI:

- Lack of accurate, manually validated classifications of screen types across diverse app categories
- Lack of a unified set of features extracted from UI elements for machine learning purposes

These limitations have hindered research in several ways:

- Training accurate screen classification models is difficult due to inconsistent classification
- Developing automated testing methods that address different UI types is difficult

- Studying design patterns and user interactions across different screen types is limited

This research provides a UI dataset, named MASC, which addresses these gaps by:

- Providing manually validated classifications of 10 major screen types, ensuring high-quality classifications for machine learning
- Providing a unified set of 11 features extracted from each UI, facilitating robust and consistent analysis
- Covering a diverse range of app screens, enabling comprehensive studies of UI design and functionality
- Supporting the development of more effective automated testing strategies for mobile apps.

Due to the increasing spread of mobile apps on the Internet and the challenges of mobile app analytics, this research has the potential to significantly improve mobile app testing efficiency, enhance user interface design practices, and ultimately lead to better user experiences across a wide range of mobile applications. The contributions of this research can be summarized in the following two main points:

1. Building a dataset of mobile app screens and then manually classifying them.
2. Training machine learning (ML) algorithms with this dataset to detect the type of unseen mobile screens.

This paper is organized as follows: Section II presents related work review; Section III describes the methodology of building and analyzing the new dataset; Section IV describes the proposed ML model for UI screens classification based on the MASC dataset; Section V presents the classification algorithms evaluation metrics; Section VI presents the results of evaluating the efficiency of applying different ML algorithms, with the new dataset, in the classification of UI screens; and Section VII presents conclusion and future work.

II. RELATED WORK

Currently, there is no publicly available dataset that links UI designs to specific layout topics. Previous design mining applications and platforms, such as Webzeitgeist [5] and GUIfetch [6], allow users to explore and query web design datasets, but UI layout modeling is not possible due to the lack of labeled data. Other datasets provide fine-grained app categories, but they do not provide information about UI design. For example, Berardi et al. [7] automatically inferred app categories using app metadata analysis, and Zhu et al. [8] annotated 680 mobile apps according to two general-purpose levels, with Games (Level 1) including Action and Strategy (Level 2). There is also a mobile app dataset categorized according to Reiss' profiles, such as honor, idealism, and power, but these categories have little practical use for UI designers [9]. This section examines the most used open-source datasets in the current literature on incorporating AI assistance into UI classification. Table 1 lists the datasets discussed in this section and their attributes.

The largest repository of UI datasets is the RICO dataset [10]. It contains more than 72,000 Android smartphone UI screenshots and their corresponding Android View hierarchy, which were collected using a crowd-sourcing methodology.

Liu et al. [11] extended RICO by adding semantic annotations for UI screenshots. Using the automatic method, the authors categorized the UI components found in the RICO dataset into 25 types. They further categorized these UI element types into 135 icon classes and 197 text button concepts. This dataset has the disadvantage that the annotations are unreliable because the UI components were classified and tagged automatically.

Activity recognition is based on the premise that different activities in an Android application share a similar interface structure. To exploit this similarity, researchers have used machine learning techniques to classify each activity in the application into one of seven pre-defined activity types [12] . Wang et al.[13] collected 112,085 human-annotated English summarization for 22,417 unique UI screens dataset from Rico and found that deep learning models outperform heuristic approaches. Human evaluation also favored their full model.

Several other datasets have been proposed for classifying mobile application screens and tasks. For example, DroidTask[14] offers a dataset of 158 labeled tasks across 13 open-source apps. While this dataset provides a foundation for task-based analysis, However, its small dataset size and limited diversity in application types restrict its generalizability Similarly LlamaTouch[15] offers a more extensive dataset with over 3,500 tasks and expert-curated metadata, significantly improving the coverage of UI interactions. However, it primarily focuses on task-level interactions rather than structural screen classification, making it less effective for studies requiring in-depth UI component analysis.

In our research, we assembled a dataset, named MASC, based on the well-known Rico dataset. This dataset includes a total of 7,065 UI images representing a wide range of mobile app activities. These activities were accurately manually classified into 10 distinct classes. We have developed a feature extraction technique to extract the vital features from each image accurately, which allowed us to train machine learning algorithms employed in predicting activity types. This capability enables us to automatically identify the specific activity in which the user is currently engaged. It has the potential to enhance various tasks, including improving user experience and providing personalized content. It also allows developers to create a specific testing method for each of these 10 types. In what follows, we will discuss in detail the contents of our database, followed by a comprehensive study of applying ML algorithms, with the MASC dataset, in the classification of mobile app screens.

MASC offers a unique contribution to the field by providing manually verified classifications for 10 distinct screen types, along with a standardized set of 11 extracted features per screen. This comprehensive approach allows for a more nuanced analysis of UI elements associated with specific functionalities across different types of app screens. The combination of verified classifications and standardized features makes MASC particularly well-suited for training machine learning models for app screen classification. Furthermore, this dataset enables researchers and developers to study design patterns more effectively, potentially supporting the development of intelligent design tools and optimizing user experiences in ways that were not possible with previous datasets.

Table 1. List of UI datasets and their attributes.

Dataset	Description	No. of Apps	No. of UIs	Annotations	Advantages	Disadvantages
Shirazi, et al. [16]	UIs created through XML layout files	400	29K	Twenty-one different apps categories	• Large-scale structural data • Covers various app categories • Useful for analyzing layout patterns	• Limited to XML layouts, not visual data • May not capture dynamic UI changes • Potentially outdated (published in 2013)
ERICA [17]	UI screenshots, UI wireframes, and view hierarchy (JSON file)	2.4k	18.6k	indexed more than 3000 flow examples by using machine learning classifiers	• Includes UI flows and interactions • Combines visual and structural data • ML-based flow indexing	• No specific screen type classification. • Large size may be challenging to process.
RICO [10]	UI screenshots, view hierarchy (JSON file) and UI wireframes	9.7k	72.2K	spanning 27 apps categories	• Largest UI dataset available • Rich in UI diversity and patterns • Includes screenshots and hierarchies	• UI elements categorized by automated means, unreliable annotations • No classification for each screen • Challenging to process due to size
Liu, et al. [11]	expanded RICO by adding semantic annotations to the UIs.	9.7k	72.2K	25 types of UI components, 197 text button concepts, and 135 icon classes shared across apps.	• Detailed semantic annotations for UI elements • Useful for design semantics understanding	• Automated annotation process may introduce errors • Focused on individual UI elements rather than overall screen types
Screen2Words [13]	screen summaries in CSV format for screens from the RICO	6.3k	22.5k	112,085 screen summaries for 22,417 screens	• Textual descriptions of UI elements • Useful for NLP tasks related to UI • Combines visual and textual information	• Not specifically designed for screen type classification • May contain subjective or inconsistent human annotations • No predefined categories for screen types • Primarily focused on generating textual descriptions, not classified.
Enrico [18]	UI screenshots, view hierarchy (JSON file) and UI wireframes	1.1k	1.4k	20 categories of UI design topics	• Focused on UI design topics • Manually labeled dataset • Includes wireframes and hierarchies	• The number of UIs in each category is small, • No specific feature extraction method for classification • the total number of UIs is also small, which affects the efficiency of using this dataset in UI classification by AI.
MASC (Ours)	UI screenshots, view hierarchy (JSON file) , UI wireframes, and Vectors.CSV	3.4k	7.1k	10 categories of UI design topics	• Manually classified into 10 distinct categories • Focused dataset easier to manage and analyze • Custom feature extraction algorithm for each UI • Validated through comprehensive machine learning experiments • Specifically designed for mobile app screen classification	• Limited to 10 predefined categories . • Primarily based on Android, but the classification approach can be potentially applied to other platforms.

III. METHODOLOGY

The framework, shown in Fig. 1, provides an overview of the different phases involved in building a new dataset, called MASC (a shorthand for **Mobile App Screens Classification**) and employs an ML approach for classifying mobile applications screens

As shown in Fig. 1, the architecture of the proposed framework includes several phases. First, data about the selected screens is collected from a variety of sources, such as screenshot UI screens, semantic wireframes, and JSON files, then data is preprocessed and filtered to remove noise. Next, feature extraction is performed to convert the data into a numeric vector that has 11 features for each UI screenshot. Then, the dataset is split into training, validation, and testing sets. The training set is used to train the machine learning model. The validation set is used to evaluate the performance of the model and select the best parameters for it. The testing set is used to evaluate the final performance of the model on unseen data. The framework can be used to optimize classification efficiency for classifying mobile app screens.

The MASC dataset serves as the foundation for developing machine learning models to detect and classify screen types. MASC dataset building process includes three phases: Dataset Collection, Data Classification, and Feature Extraction.

A. DATASET COLLECTION

To build a mobile UI dataset, we randomly collected 30k screenshots from three open-source data sources: the first is the Rico dataset [10] containing 72k UIs; the second is the Screen2Words dataset [13] containing 22k screenshots taken from the Rico-SCA dataset; and the Enrico dataset [16] containing 1,460 screenshots of mobile app screens taken from the Rico dataset. We carefully selected screenshots from a various set of applications to ensure a diverse representation of mobile UIs. We manually classified the 30K UIs and selected only the UIs that belong to ten different design topics (see Table 2) to arrive at a high-quality dataset containing 7065 screenshots of mobile app UIs taken from over 3400 apps, which we named the MASC dataset.

B. DATASET CLASSIFICATION

After collecting the dataset in the first step, three persons were assigned to manually review each of the collected screens and classify them into the appropriate class from the ten classes. To ensure consistent and accurate classification, specific criteria were established for each screen type. Table 2 outlines these criteria in detail, providing a comprehensive guide used by the researchers during the manual classification process.

These detailed classification criteria enabled the three persons to systematically categorize the diverse range of mobile app screens, ensuring consistency and accuracy throughout the dataset creation process. Based on these criteria, we classified our dataset as shown in Table 3, which presents the number of UI designs belonging to each class along with a brief description.

Table 2. Classification Criteria for Mobile App Screen Types

Class	Classification Criteria
Chat	Displays conversation threads or message bubbles - Includes a text input field for sending messages - May show online status of contacts.
Home	Central hub for app navigation - Displays main features or content summary - Often has a bottom navigation bar or prominent action buttons.
List	Presents data in a scrollable column format - May include items like articles, products, or contacts - Often have options to sort or filter the list.
Login	Contains username and password input fields - May include "Sign In" or "Log In" buttons - Often has options for password recovery or new account creation.
Map	Displays geographic information - Includes interactive map controls (zoom, pan) - May show location markers or route information.
Menu	Displays a hierarchical list of choices or app sections - May be a side drawer or full-screen list - Often includes icons next to menu items.
Profile	Shows user information and preferences - May include profile picture, bio, and personal stats - Often has options to edit profile information.
Search	Features a prominent search bar - May include recent search history or suggested search terms - Often has a magnifying glass icon.
Setting	Lists configurable options for the app - May include toggles, dropdown menus, or input fields for various settings - Often has a gear or cog icon.
Welcome	Introductory or onboarding content for new users - May include app highlights or brief tutorials - Often has a "Get Started" or similar call-to-action button.

During the classification process, each screen was only included in the dataset if at least two of the three classifiers agreed on its category, ensuring consensus and reducing individual bias. Screens without agreement between at least two classifiers were excluded, maintaining data quality at the expense of quantity. This conservative approach ensured only screens with clear classifications were included, which is crucial for the dataset's reliability in machine learning applications.

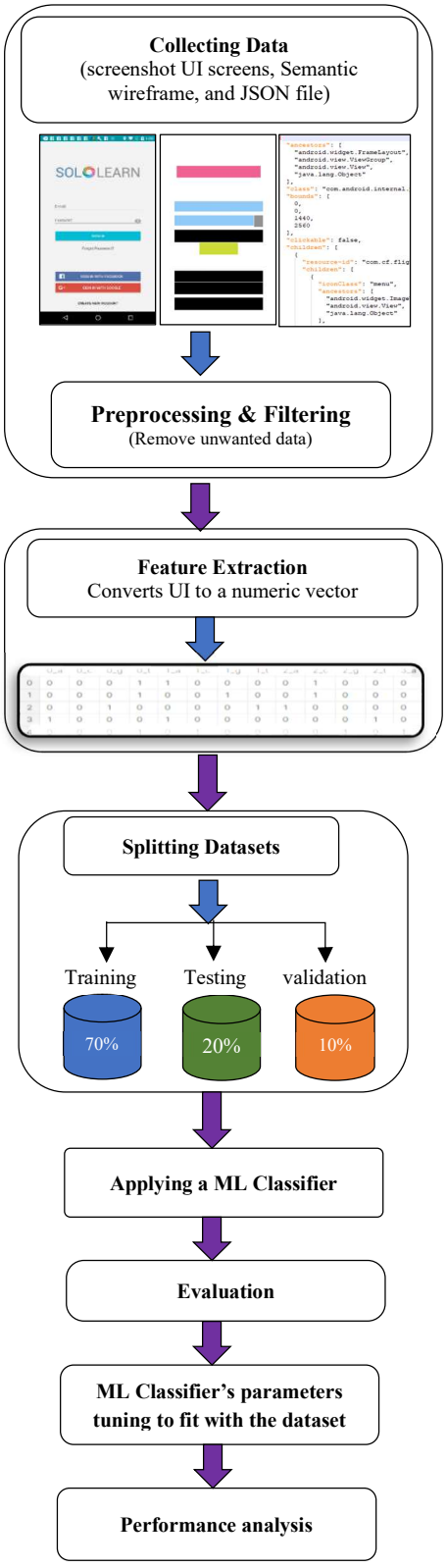

Figure 1. The proposed framework for building the MASC dataset and using it in a ML approach for classifying mobile apps screens.

Table 3. Ten classes, in alphabetical order, and the number of UI designs that belong to each class.

Class	No. of Screens	Description
Chat	329	Chat communication
Home	866	App navigation
List	960	Data visualization in a column
Login	889	User authentication
Map	500	Geographic display
Menu	557	Item selection
Profile	526	User profile information
Search	725	Content discovery
Setting	629	App configuration
Welcome	1084	First-run experience

Regarding challenging cases, screens with complex or hybrid functionalities were likely the most difficult to classify. For instance, a screen combining search functionality with a menu layout might have caused disagreement among classifiers. This requires careful consideration of the underlying function of the screen to determine the most appropriate category.

In the dataset, each screen is accompanied with a screenshot of the user interface (UI), a JSON file of the view hierarchy, a semantic wireframe, and a JSON file of the hierarchy of this semantic UI, as shown in Fig. 2. View hierarchies include various UI elements, such as images, buttons, and a map view, each with specific properties like class, bounds, and clickability, and the relationships between them. The view hierarchy offers insights into how these elements are organized within the app's user interface, facilitating development and design.

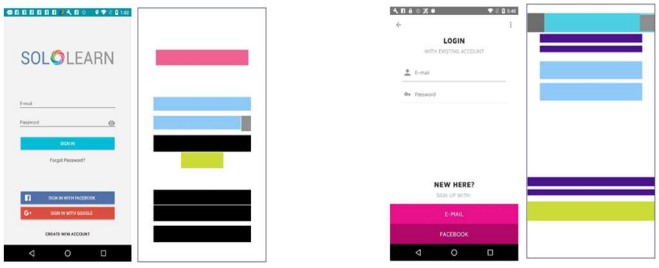

Figure 2. (a) screenshot UI screens, (b) Semantic wireframe images.

Each activity has a variety of elements that are connected to the user interface components it includes, such as the distinct classes of elements, their set of attributes, their relative placement in the activity, the number of elements shown in the activity, and so on. An activity may also have a navigation drawer, which is a panel on the left side of the screen that displays the application's major navigation choices. It is usually hidden, but it is shown when the user swipes their finger from the left side of the screen or clicks on a designated button.

C. FEATURE EXTRACTION

The extraction of important features is an essential phase in the presented model since irrelevant features might reduce the efficacy of the ML classifier. Careful features selection improves classification accuracy and shorten model training time. Feature extraction involves breaking down huge amounts of raw data into smaller groups for processing, as processing these huge datasets needs substantial computing resources due to the large number of variables.

While building the features vector, we have to choose those aspects that are most informative and may vary depending on the type of activity. Wang, et al. [13] have proven that an activity can often be identified by its visual elements, which are the elements that the user can interact with directly, such as buttons, text boxes, and their location on the screen. Thus, it is assumed that the interactive elements for the user appropriately represent the activity. By examining the core activity templates from Android Studio's activity design guidelines, we noticed that each activity screen can be artificially divided into three parts: the top, middle, and bottom. We utilize the following screen guideline division: 15%-70%-15% from top to bottom, as illustrated in Fig. 3. where headers or navigation elements are typically at the top, main content in the middle, and action buttons or navigation bars at the bottom. This division allows us to capture the spatial distribution of UI elements, which can be crucial for distinguishing between screen types.

Our feature selection was influenced by common UI design patterns and elements typically found in various screen types. For example, we quantify clickable elements, text fields, and swipeable components, as these often indicate the interactivity level and purpose of a screen. The number of text input fields, for instance, is usually higher on login screens, while list screens tend to have more vertically swipeable elements. Additionally, we incorporated keyword analysis, selecting common words for each class based on frequency analysis across multiple apps. For example, terms like "username" and "password" are often associated with login screens, while "settings" and "preferences" are common in settings screens. This multi-faceted approach to feature extraction, combining element counts, spatial information, and textual analysis, contributes significantly to the accuracy of our classification model by providing a comprehensive representation of each screen's characteristics.

As a result, we concentrate on the following interactive element groupings, which might exist in each of the three activity sections:

- Clickable elements: Elements that the user can tap on to interact with the activity.
- Swipeable elements: Horizontal swipeable elements are those that can be swiped by the user left and right, while vertical swipeable elements are swiped up and down.
- Text field elements: Elements where the user can enter text.

We use the number of elements from each of the above element groups, in the part of the activity where they are most likely to occur, as features. Thus, the first set of features includes 6 features, representing the number of clickable and text field elements in each of the three activity sections of the screen. The second set of features includes 4 features: the first feature is the total number of elements displayed on the screen, irrespective of their grouping or location; the second feature is set to 1 if the activity contains a navigation drawer; the third feature is the number of horizontal scrollable elements in the screen's center; and the last feature is the number of vertical scrollable elements in the screen's center. The final set includes only one feature, which is the "keywords" found in the text on the screen. The 11 features extracted from each screen are used to form its feature vector, as shown in Fig. 4.

To assist in extracting the 11 features from any given activity, we have created lists of common clickable, text field, vertical swipe, and horizontal swipe elements. In addition, we have created a dictionary of common words for each of the 10 classes mentioned in Table 3. This is achieved by examining many screens of applications belonging to each class and identifying the most used words for this class. For example, for the login class, the most common keywords are "login", "username", and "password".

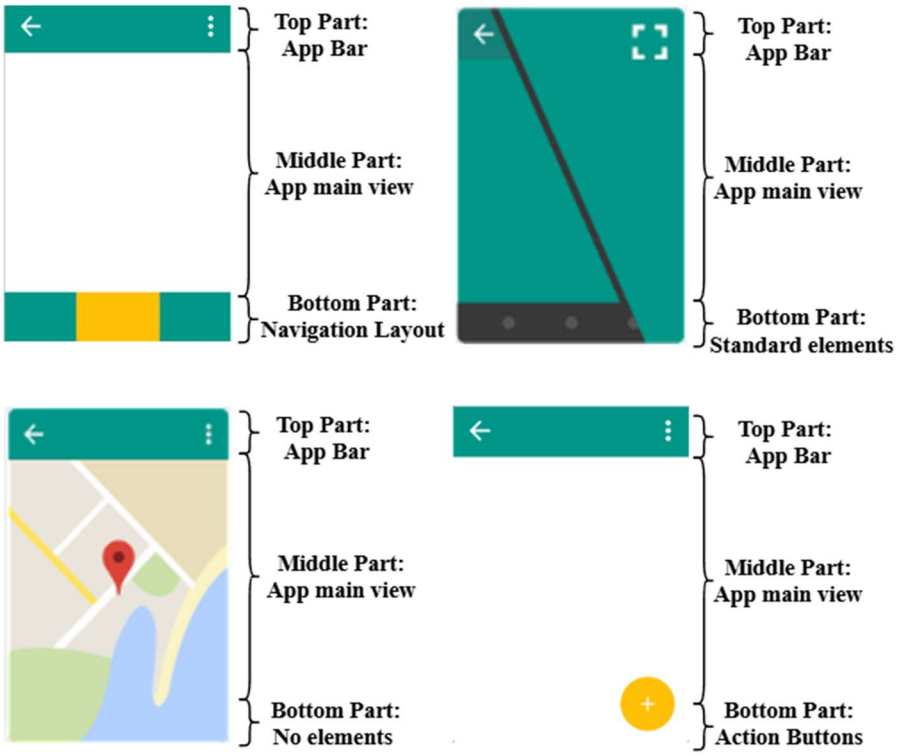

Figure 3. Screen divisions across common activities templates.

Features List										Keywords
1	2	3	4	5	6	7	8	9	10	
No. of Clickable Elements - Middle	No. of Clickable Elements - Bottom	No. of Clickable Elements - Top	No. of Text Field Elements - Middle	No. of Text Field Elements - Bottom	No. of Text Field Elements - Top	No. of all Elements	Does the activity contain a navigation drawer?	No. of horizontal swipeable Elements	No. of vertical swipeable Elements	keywords

Figure 4. The feature vector structure for an activity.

Fig. 5 shows the steps of our extract features algorithm, **Extract_Features**, which extracts the 11 features from any given activity. The inputs to this algorithm are: the JSON file, which describes the view hierarchy of the activity to be processed, act; the created lists of common elements: clickable_types, text_field_types, vertical_swipe_types, and horizontal_swipe_types; and the dictionary of common words: **common_words**. The output of the algorithm is the feature vector for the given activity act. The algorithm works as follows: Line 1 creates two empty lists, **FeaturesList** and **Keywords**, to hold the extracted features; Line 2 reads the JSON file of act; Line 3 calls a procedure, named Extract_Children(), developed to extract all child elements of act from the JSON file into a list: child_elements; Line 3 calculates the height percentages for the top, middle, and bottom parts of the screen act; Lines 7-38 calculate the first 10 features for the given activity act, by checking the type of each child in **child_elements** and incrementing the corresponding elements in **FeaturesList**; Line 39 calls a procedure, named **Extract_Words()**, developed to extract important text from the JSON file into a list: words; Lines 40-45 creates the last feature (Keywords) for act, which is a list of the keywords found in the text on the screen, by checking each word in words and adding it to Keywords, if it is found in the dictionary **common_words**; finally, Line 47 writes the feature vector (**FeaturesList** + **Keywords**) of act to a CSV file.

The procedure **Extract_Words**() creates the list words as follows: the text from each screen element and the text that describes that element, is extracted from the JSON file; then the extracted text is cleaned by removing unwanted content, such as symbols, numbers, and common words, and stored in the list words.

Table 4 shows the obtained feature vectors for some mobile screen samples of four different classes from the MASC dataset, and it also shows the screen type based on the manual classification, where each screen has a unique id.

```
Algorithm Extract_Features
Input:  JSON file (hierarchy for an activity act);
        Lists of common clickable, text input, vertical swipe, and
        horizontal swipe elements: clickable_types, text_input_types,
        vertical_swipe_types, horizontal_swipe_types; Dictionary of
        common words for each of the 10 classes: common_words.
Output: The feature vector (FeaturesList and Keywords) for the given
activity act
Begin
 1. Create two empty lists to hold the extracted features: FeaturesList,
    Keywords
 2. Read JSON file of act.
 3. child_elements = Extract_Children(JSON File).   //Extract all child
    elements from JSON file
 4. Calculate the height percentages for the top, middle, and bottom parts of
    the screen act.
 5. For each child ∈ child_elements do
 6. Begin
 7.    If child is on top        // Determine the location of the child on act
 8.        location = top
 9.    Else If child is on middle
10.        location = middle
11.    Else
12.        location = bottom
13.    End If
14.    FeaturesList[7]++;     // Increment no. of all elements in act
15.    If child ∈ clickable_types  then  // check if child is a clickable element
16.      If location == middle then
17.        FeaturesList[1]++;    // Increment no. of clickable elements at middle
18.      Else If location == bottom then
19.        FeaturesList[2]++;    // Increment no. of clickable elements at bottom
20.      Else If location == top then
21.        FeaturesList[3]++;    // Increment no. of clickable elements at top
22.      End If
23.    Else If child ∈ text_input_types then
24.      //check if child is a text input element
25.      If location == middle then
26.        FeaturesList[4]++;    // Increment no. of text field elements at middle
27.      Else If location == bottom then
28.        FeaturesList[5]++;    // Increment no. of text fields elements at bottom
29.      Else If location == top then
30.        FeaturesList[6]++; // Increment no. of text field elements at top
31.      End If
32.    Else IF child is navigation drawer then
33.      // child is a navigation drawer control
34.      FeaturesList[8] =1;
35.    Else IF child ∈ horizontal_swipe_types then
36.      // child is a horizontal swipeable element
37.      FeaturesList[9]++;
38.      // Increment no. of horizontal swipeable elements
39.    Else IF child ∈ vertical_swipe_types then
40.      //child is a vertical swipeable element
41.      FeaturesList[10]++;  // Increment no. of vertical swipeable
   elements
42.    End If
43.    words = Extract_Words(JSON file)
44.    // Get important text from the JSON file
45.    For each word ∈ words do
46.    Begin
47.      If word ∈ common_words then
48.        // word is found in the dictionary of common keywords
49.        Add word to Keywords
50.      End If
51.    End For
52. End For
53. Write feature vector (FeaturesList + Keywords) of act to a CSV file.
54. End
```

Figure 5. The Extract_Features Algorithm.

D. MASC APPLICATIONS

Here are the potential applications of the MASC dataset, organized by functional categories:

1) **Testing and Validation:**
   - UI Testing Automation: MASC dataset can be used to train models for automated mobile app UI testing, which can significantly lower the manual effort required for quality assurance and avoid expensive rework due to early bug detection.
   - Design System Verification: MASC dataset can be used to build automated tools that verify compliance with design guidelines, identify anomalies, and maintain consistency across large-scale applications.
2) **Development and Generation:**
   - Auto UI Generation: MASC dataset can be used to train generative models that automatically create UI designs based on specifications, significantly accelerating the prototyping process and reducing initial development time.
   - Code Generation Tools: MASC dataset can be used to develop systems that automatically generate implementation code from UI designs, streamlining the development process and reducing manual coding effort.
3) **Accessibility and User Experience:**
   - UI Accessibility Enhancement: MASC dataset can be used to develop models that improve app accessibility, enabling features like voice commands, gesture navigation, and text-to-speech capabilities to make apps more inclusive for users with disabilities.
4) **Analysis and Research:**
   - UI Pattern Analysis & Classification: MASC dataset can be used to study UI evolution and classify screens by type, helping identify successful design patterns and trends that improve user experience.
   - Performance Optimization: MASC dataset can be used to study relationships between UI patterns and app performance, helping identify optimal design approaches that balance aesthetics with technical efficiency.
   - Competitor Analysis Tools: MASC dataset can be used to build systems that analyze and compare UI patterns across competing apps, providing insights for strategic design decisions and market differentiation.
   - Mobile App Security Analysis: Leveraging UI classifications, the dataset can help identify potential vulnerabilities in app interfaces, such as insecure input fields or exposed sensitive actions.

IV. ML APPROACH FOR CLASSIFYING UI SCREENS

The proposed ML model for UI screens classification based on the MASC dataset consists of three phases: Train/Test Splitting, Classification Algorithms, and Parameters Tuning.

A. TRAIN/TEST SPLITTING

To ensure an unbiased evaluation of the ML model, we randomly partitioned the dataset into three non-overlapping subsets

- **Training Set (70%, n = 4946)**: Used to fit model parameters and learn the underlying patterns and relationships between interface features and their labels.
- **Validation Set (10%, n=707):** Employed for hyperparameter tuning (e.g., learning rate, regularization strength, tree depth) and model selection, without influencing the training process.
- **Test Set (20%, n = 1412)**: Held out until final evaluation to provide an unbiased benchmark of model generalization on completely unseen mobile screen images.

The training set is used to optimize the model's weights, whereas the validation set guides hyperparameter selection and early stopping, preventing overfitting before assessing final performance on the test set.

Table 4. Feature vectors extracted for screen samples.

Screen Id	No. of Clickable Elements - Middle	No. of Clickable Elements -	No. of Clickable	No. of Text Field Elements - Middle	No. of Text Field Elements - Bottom	No. of Text Field Elements	No. of all Elements	Does the activity contain a navigation drawer?	No. of horizontal swipeable	No. of vertical swipeable Elements	Keywords	Label
1054	0	1	4	0	0	0	6	0	0	1	chat contacts receive	Chat
10560	4	0	1	1	0	0	7	0	0	1	composing	Chat
11382	5	1	2	0	1	0	20	0	0	11	chat conversation	Chat
15833	5	0	2	2	0	0	9	0	0	0	username password login	Login
15869	3	0	0	1	0	0	4	0	0	0	username register	Login
47278	5	4	3	0	0	0	13	0	0	1	name age profile	Profile
47280	1	0	0	0	0	0	2	0	0	1	name age profile	Profile
18804	0	0	1	0	0	0	6	0	0	5	setting preferences	Setting
18833	0	0	1	0	0	0	6	0	0	5	setting Accessibility	Setting
18955	0	0	1	0	0	0	6	0	0	5	setting preferences	Setting
16127	15	0	3	0	0	0	19	1	0	0	list Settings notifications	Menu
16137	12	1	3	2	0	0	19	1	0	0	list Settings Help logout	Menu

B. CLASSIFICATION ALGORITHMS

After finishing the data pre-processing and splitting the dataset, we can begin deploying several classification algorithms in our model. We will evaluate the performance of 10 different algorithms to ensure a thorough evaluation of the model. The training dataset are utilized to train the model with these classification methods. Various indicators are utilized to evaluate the success of the ML algorithms in classifying Mobile application screens, with the testing and the validation datasets. For developing the algorithms, we import each ML method and several performance indicators from the Python library Scikit-learn [19], such as accuracy, recall, precision, ROC-AUC and F-score. As shown in Fig. 6, the ML algorithms used include Extreme gradient boosting (XGBoost) [20], Gradient Boosting [21], Random Forest (RF) [22], Multi-Layer Perceptron [23], AdaBoost [24] , Logistic regression (LR) [25], Decision Tree (DT) [26], Naive Bayes (NB) [27], Support Vector Machines (SVM) (SVM RBF, and SVM Linear) [28].

C. PARAMETERS TUNING

Modifying parameters before applying a training process can aid in controlling the behavior of the ML algorithms. These parameter adjustments can have a considerable influence on model training duration, accuracy, and convergence. By experimenting with different parameter settings, we can optimize the behavior of the ML algorithms and achieve better classification performance. For instance, when using the Gradient Boosting algorithm, we use max depth=3, learning rate=0.1 and n estimators=100 to achieve the highest classification accuracy. In the DT algorithm, we use max depth=5, which provides an optimal solution. For the RF model, we found that setting max depth=5 and n estimators=10 yielded the best classification results. We also employ several kernels to ensure the accuracy of SVM classification, including SVM RBF and SVM Linear.

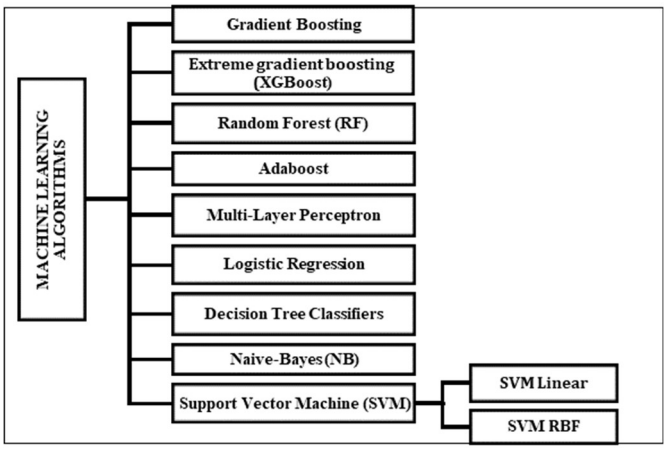

Figure 6. The used classification algorithms.

V. EVALUATION METRICS

In the evaluation phase, two widely used evaluation metrics were applied to assess the performance of each classification model: Accuracy and F1 Score. A confusion matrix [29, 30] is used in the calculation of these metrics.

	Positive (1)	Negative (0)
Positive(1)	TP	FP
Negative (0)	FN	TN

Figure 7. The Confusion Matrix.

The confusion matrix is a performance statistic for a ML classification task where the output might be two or more classes. It is a table with four alternative combinations of projected and actual values, as shown in Fig. 7, where TN refers

to the correct number of classifications of negative instances, TP refers to the correct number of classifications of positive instances, FP refers to the incorrect number of classifications of negative instances, and FN refers to the incorrect number of classifications of positive instances.

The evaluation metrics are defined below.

Accuracy measures the percentage of occurrences that are correctly classified relative to all instances. It is computed using the following formula

$$\boldsymbol{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}. \quad (1)$$

Precision calculates the proportion of accurately predicted positive outputs. The following formula is used to compute it:

$$\boldsymbol{Precision} = \frac{TP}{TP + FP}. \quad (2)$$

Recall is The percentage of accurate positive predictions based on all of the dataset's actual positives. The following formula is used to compute it:

$$\boldsymbol{Recall} = \frac{TP}{TP + FN}. \quad (3)$$

F1-Score is defined as the suitably scaled harmonic mean of a model's precision and recall. The formula is used to compute it:

$$\boldsymbol{F1\ Score} = \frac{2 * Precision * Recall}{Precision + Recall}. \quad (4)$$

ROC_AUC (Receiver Operating Characteristic - Area Under the Curve) [31] is a critical metric for assessing classification model performance. The ROC curve is generated by systematically adjusting the classification threshold of the model and plotting the True Positive Rate (TPR) against the False Positive Rate (FPR). The area under the ROC curve (ROC-AUC) is subsequently computed by integrating the ROC curve as follows:

$$\boldsymbol{TPR} = \frac{TP}{TP + FN}. \quad (5)$$

$$\boldsymbol{FPR} = \frac{FP}{FP + FN}. \quad (6)$$

The ROC_AUC is then determined by integrating the ROC curve, providing a reliable measure of the model's ability to distinguish between classes and offering a comprehensive evaluation of classification performance.

$$\boldsymbol{ROC\_AUC} = \int (recall\ (FPR))\ dFPR. \quad (7)$$

VI. EXPERIMENTAL RESULTS

We performed a comparative evaluation of the classification outcomes for application screens using ten distinct machine learning algorithms, as depicted in Figure 6. To assess their performance, the previously defined evaluation metrics—recall, accuracy, precision, F1-score, and ROC-AUC—were employed. The results of this analysis offer valuable insights into the efficacy of each algorithm in classifying mobile application screens. Detailed performance metrics for the ten algorithms are summarized in Table 5, highlighting their effectiveness in screen classification based on the established evaluation criteria.

As illustrated in Table 5, All of the assessed algorithms received high accuracy scores, ranging from 81.16% to 93.48%, underscoring their overall efficacy in screen classification tasks. Notably, Gradient Boosting, Random Forest, XGBoost, Multi-Layer Perceptron and SVM Linear demonstrated superior performance, with accuracy scores between 93.06% and 93.48%, positioning them as the most suitable algorithms for this task. Among these, Gradient Boosting consistently outperformed others, achieving the highest scores across all metrics, including accuracy, recall, precision, and F1-score. In contrast, SVM RBF exhibited the lowest accuracy (81.16%) and F1-score (80.9%), indicating comparatively weaker performance. In terms of precision, Gradient Boosting, SVM Linear, and Logistic Regression achieved the highest scores, exceeding 95%. This reflects their ability to minimize false positive predictions, a critical factor in classification tasks. Conversely, SVM RBF recorded the lowest precision score (85.52%), further highlighting its limitations in this context.

Table 5. Comparison of metrics scores of the 10 ML algorithms in app screens classification.

No.	ML algorithms	Accuracy	Precision	Recall	ROC-AUC	F1 score
1	Gradient Boosting	93.48	96.07	93.46	99.6	94.39
2	XGBoost	93.2	95.5	93.17	99.51	93.99
3	Multi-Layer Perceptron	93.2	95.8	93.25	99.56	94.13
4	Random Forest	93.06	95.51	93.16	99.06	94.04
5	Logistic Regression	92.63	95.81	92.48	99.48	93.69
6	Adaboost	83.29	85.78	84.22	98.36	83.93
7	Naive Bayes	90.65	94.63	90.77	99.43	91.9
8	Decision Tree	92.35	94.18	92.33	96.91	93.01
9	SVM RBF	81.16	85.52	79.02	98.04	80.9
10	SVM Linear	93.2	95.99	93.28	99.48	94.27

For recall, most algorithms performed well, with scores above 90%, indicating their ability to correctly identify the majority of positive cases. The exceptions were Adaboost and SVM RBF, which had lower recall scores in the range of 79% to 85%, suggesting that these algorithms were less effective at detecting positive cases.

The ROC-AUC scores for all algorithms achieved high values, ranging from 96.91% to 99.6%, indicating excellent discriminative power. Among them, The Gradient Boosting method offered the best balance between true positive rate and false positive rate, as evidenced by its highest ROC-AUC score of 99.6%.

Based on the results provided in Table 5, Gradient Boosting achieved the best performance with a test accuracy of 93.48%. This outstanding performance can be attributed to several reasons: (1) its ability to handle non-linear data by building sequential models that correct errors from previous models, enabling it to capture complex relationships in the data; (2) its resistance to overfitting, achieved through gradual learning and tree pruning techniques; (3) its capability to handle mixed features, including both numerical and textual data extracted from mobile app screens; (4) its strong performance in multi-class classification tasks, which aligns well with the ten-category structure of the MASC dataset; and (5) its high precision (96.07%) and recall (93.46%), the highest among all models, demonstrating its ability to accurately identify and classify instances across all classes.

In comparison, SVM with RBF kernel demonstrated lower performance, achieving a test accuracy of 81.16%. This suboptimal performance can be attributed to several factors: (1) the difficulty in determining optimal parameters, as SVM heavily relies on selecting appropriate hyperparameters, which can be challenging to optimize for complex datasets; (2) its inherent limitations in multi-class classification tasks, since SVM is fundamentally a binary classifier and requires additional strategies to extend its functionality to multi-class problems; and (3) its lower precision (85.52%) and recall (79.02%), the lowest among all models, indicating its struggle to accurately identify and classify instances across all classes.

It is worth noting that XGBoost, SVM Linear, and Multi-Layer Perceptron also performed well, with test accuracies of 93.2%, 93.06%, and 93.2% respectively. These algorithms share some similarities with Gradient Boosting in their ability to handle complex data and perform well in multi-class problems. The high ROC-AUC scores (above 99% for most models except Decision Tree and SVM RBF) indicate that most models have excellent discriminative ability across all classes. Figures 8-12 show comparisons between the Accuracy, ROC-AUC scores, F1-scores, Precision, and Recall scores, respectively, obtained for the 10 ML algorithms on the same dataset.

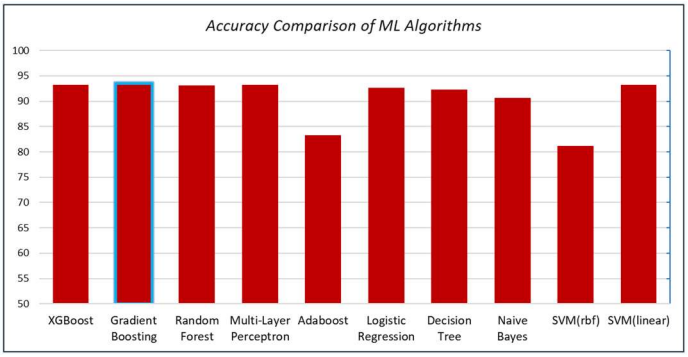

Figure 8. A comparison between the Accuracy of the 10 ML algorithms

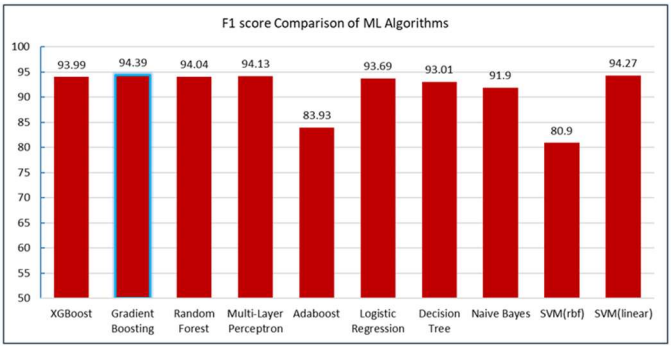

Figure 9. A comparison between the F1 score for the 10 ML algorithms.

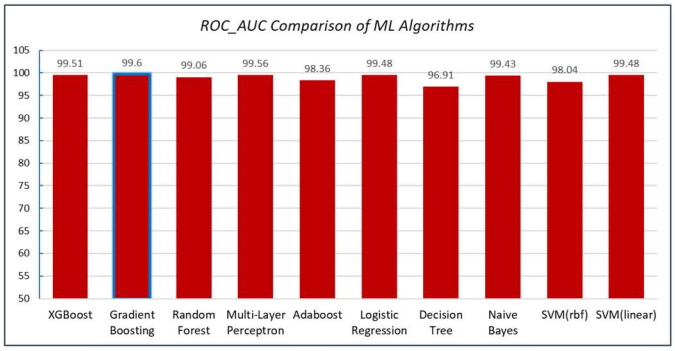

Figure 10. A comparison between the ROC-AUC for the 10 ML algorithms.

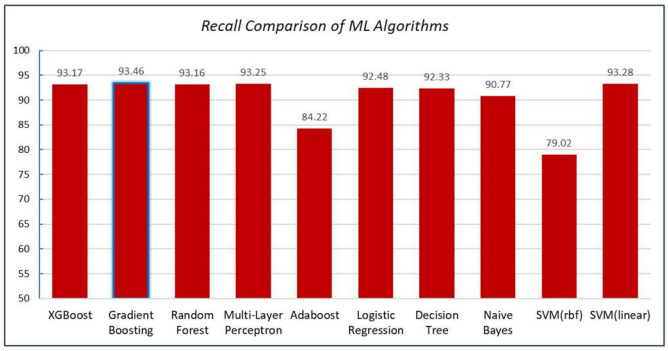

Figure 11. A comparison between the Recall for the 10 ML algorithms.

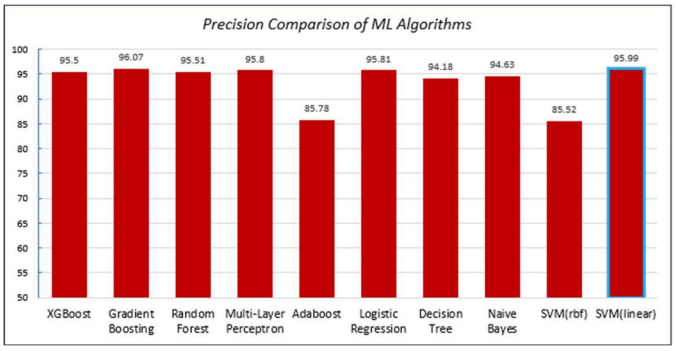

Figure 12. A comparison between the Precision for the 10 ML algorithms.

To provide a comprehensive visual representation of the performance of our ML models, we present confusion matrices for the ten algorithms evaluated in this study. These matrices offer valuable insights into the classification accuracy across different mobile app screen categories and highlight areas of strength and potential misclassification for each model. Each matrix shows the true labels vertically and the predicted labels horizontally, With color intensity signifying the quantity of occurrences in each cell. Figures 13-22 show the confusion matrices for the 10 ML algorithms on the same dataset.

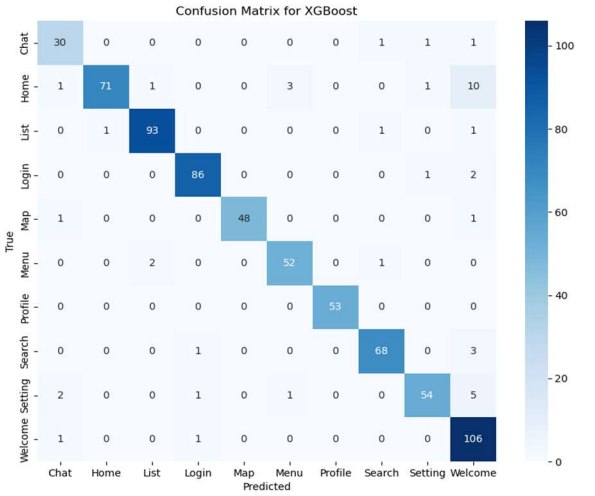

Figure 13. A XGBoost Confusion Matrix.

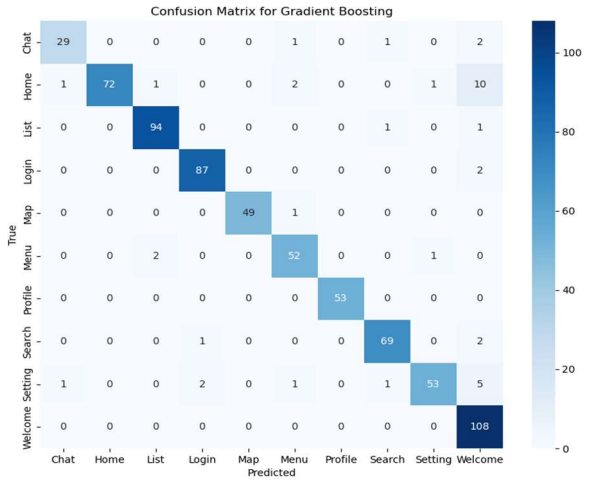

Figure 14. A Gradient Boosting Confusion Matrix.

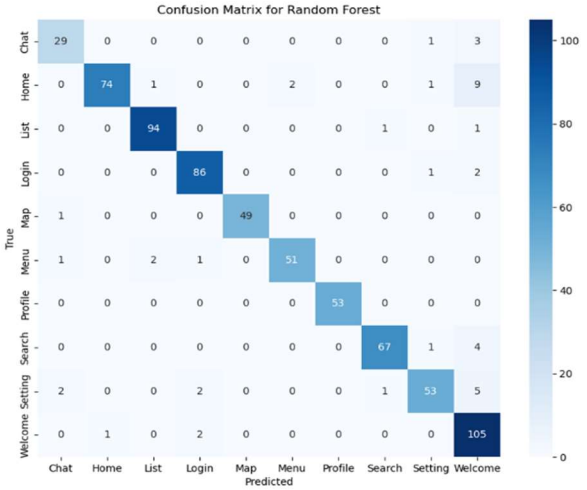

Figure 15. A Random Forest Confusion Matrix.

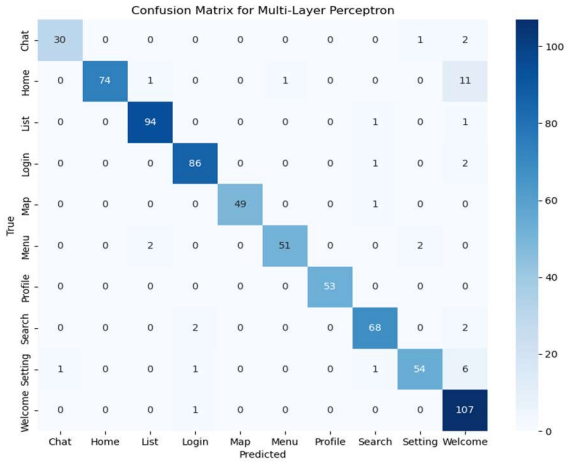

Figure 16. A Muli-Layer perception Confusion Matrix.

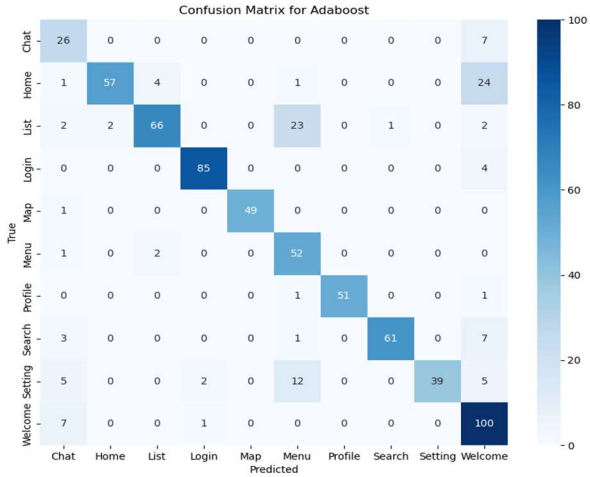

Figure 17. An AdaBoost Confusion Matrix.

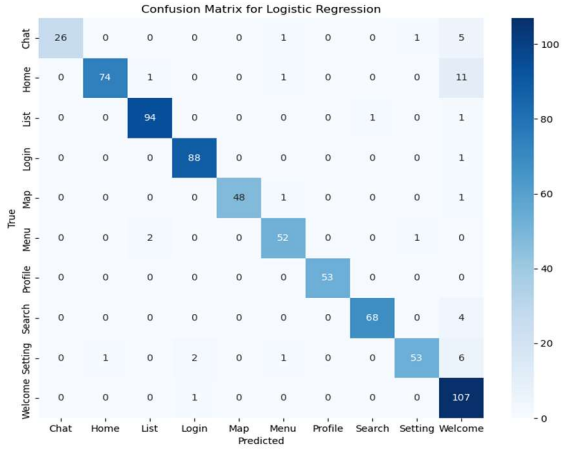

Figure 18. A Logistic Regression Confusion Matrix.

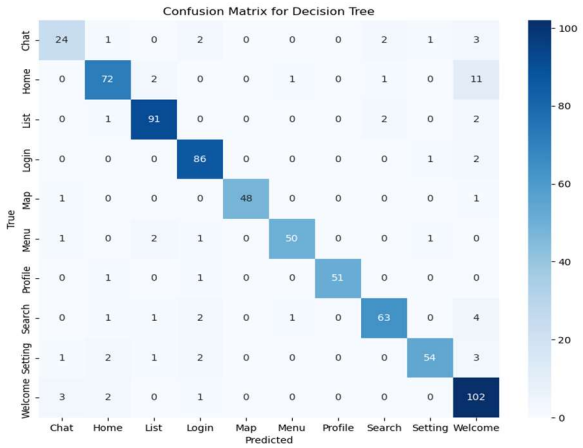

Figure 19. A Decision Tree Confusion Matrix.

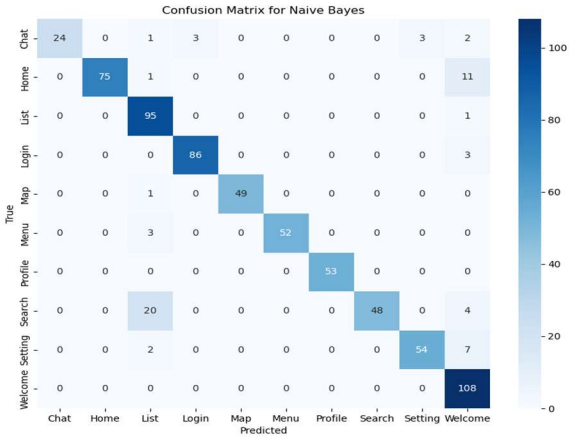

Figure 20. A Naive Bayes Confusion Matrix.

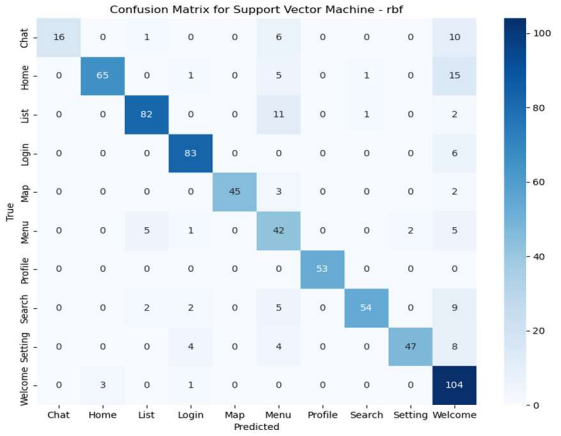

Figure 21. A Confusion Matrix for SVM RBF.

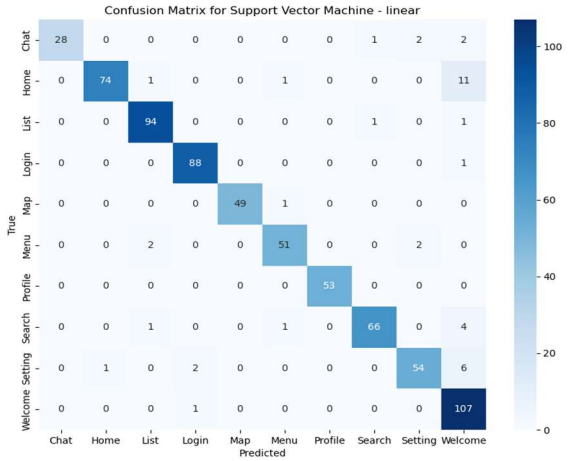

Figure 22. A Confusion Matrix for SVM Linear.

VII. CONCLUSION AND FUTURE WORK

This paper presented a new dataset, called MASC consisting of over 7000 UI screens manually categorized into 10 distinct classes. The MASC dataset provides a robust foundation for ML model development, serving as a benchmark for classification. Based on this dataset, the paper presented a proposed framework for applying ML algorithms to the classification of mobile app screens. Using the proposed framework, the paper presented a comprehensive study of the classification of mobile app screens using various ML algorithms. Evaluation metrics, including accuracy and F1 score, were used.

The results demonstrated that all the algorithms achieved relatively high accuracy rates demonstrating their effectiveness in classification. Gradient Boosting, XGBoost, SVM Linear, Random Forest, and Multi-Layer Perceptron achieved best accuracy, indicating that they are the most suitable algorithms. Gradient Boosting has the highest accuracy and F1 score, indicating that it has the best overall performance among the algorithms. while SVM RBF has the lowest accuracy and F1 score among the algorithms.

The proposed model for mobile app screens classification is a promising development that can enhance the accuracy and efficiency of mobile app screen classification. However, there are several areas where the model could be further improved, and where its potential applications could be expanded.

LIMITATIONS:

Despite the promising results, the MASC dataset and our approach have some limitations:

- **Manual Classification Bias**: While three independent annotators manually labeled the dataset, no formal measurement of inter-annotator agreement (e.g., Cohen's Kappa) was conducted. This introduces the potential for subjective bias, particularly in ambiguous screens with overlapping functionalities or unconventional layouts. Future iterations may benefit from including such agreement metrics to ensure consistency.
- **Android-Centric Design:** The dataset is based solely on Android interfaces (from the RICO dataset), which may limit the generalizability of the proposed framework to other platforms such as iOS or cross-platform frameworks like Flutter. Although the feature extraction method is conceptually applicable to other systems, empirical validation on different UI ecosystems is needed.
- **Fixed Screen Division:** Our feature extraction relies on a static horizontal screen division (15%-70%-15%) to analyze spatial distribution. While effective in many cases, this approach may not suit all UI layouts, especially those with unconventional element positioning (e.g., floating action buttons, bottom sheets, or split-screen designs). An adaptive or dynamic screen segmentation strategy could better capture such variances.

- **Scalability Challenges and Mitigation**: Expanding the dataset remains a labor-intensive process, as it depends on manual annotation. To address this, future work will explore semi-supervised and active learning approaches, allowing machine learning models to suggest labels that are then verified by human annotators. This could significantly accelerate dataset growth while maintaining labeling accuracy.

FUTURE RESEARCH DIRECTIONS:

To address these limitations and further improve the model, we propose the following areas for future work:

- **Enhanced Feature Extraction**: Develop more sophisticated techniques, possibly incorporating computer vision or deep learning approaches, to better capture complex UI layouts.
- **Dataset Expansion**: Investigate semi-automated classification methods to speed up dataset expansion while maintaining quality. This could include exploring transfer learning techniques to adapt the model to new app categories or UI design trends without extensive retraining.
- **Diverse App Categories**: Expand the dataset to include a wider range of app categories and UI designs, particularly focusing on underrepresented or emerging app types.
- **Hyperparameter Tuning**: Conduct thorough hyperparameter tuning for each algorithm to potentially improve performance further.
- **Large-Scale Validation**: Investigate the performance of the proposed model on larger datasets to validate its effectiveness in real-world scenarios and expand its potential applications.
- **Cross-Platform Extension**: Extend the methodology developed in this research to classify screens for other mobile platforms, such as iOS.

Data availability:

The data that support the findings of this study are publicly available. The MASC dataset can be accessed at https://doi.org/10.5281/zenodo.14783065, and the complete source code is available on https://github.com/Ali-Aahmed/MASC-Dataset.

References

[1] Z. Wang, et al., "A deep learning method for android application classification using semantic features," *Security and Communication Networks*, vol. 2022, Article ID 1289175, pp. 1-16, 2022. https://doi.org/10.1155/2022/1289175.

[2] K. Alharbi, T. Yeh, "Collect, decompile, extract, stats, and diff: Mining design pattern changes in Android apps," Proceedings of the 17th International Conference on Human-Computer Interaction with Mobile Devices and Services, 2015, pp. 515-524. https://doi.org/10.1145/2785830.2785892.

[3] R. Kuber, A. Hastings, and M. Tretter, "Determining the accessibility of mobile screen readers for blind users," UMBC Faculty Collection, 2020.

[4] A. Rodrigues, et al., "Open challenges of blind people using smartphones," *International Journal of Human–Computer Interaction*, vol. 36, issue 17, pp. 1605-1622, 2020. https://doi.org/10.1080/10447318.2020.1768672.

[5] R. Kumar, et al, "Webzeitgeist: design mining the web," *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2013, pp. 3083–3092. https://doi.org/10.1145/2470654.2466420.

[6] F. Behrang, S.P. Reiss, and A. Orso, "GUIfetch: supporting app design and development through GUI search," *Proceedings of the 5th International Conference on Mobile Software Engineering and Systems*, 2018, pp. 236-246. https://doi.org/10.1145/3197231.3197244.

[7] G. Berardi, et al., "Multi-store metadata-based supervised mobile app classification," *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, 2015, pp. 585-588. https://doi.org/10.1145/2695664.2695997.

[8] H. Zhu, et al., "Mobile app classification with enriched contextual information," *IEEE Transactions on Mobile Computing*, vol. 13, issue 7, pp. 1550-1563, 2013. https://doi.org/10.1109/TMC.2013.113.

[9] E. Platzer, and O. Petrovic, "Learning mobile app design from user review analysis," *International Journal of Interactive Mobile Technologies (IJIM)*, vol. 5, issue 3, pp. 43-50, 2011. https://doi.org/10.3991/ijim.v5i3.1673.

[10] B. Deka, et al., "Rico: A mobile app dataset for building data-driven design applications," *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology, UIST'2017*, Canada, 2017, pp. 845-854. https://doi.org/10.1145/3126594.3126651.

[11] T. F. Liu, et al., "Learning design semantics for mobile apps," *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology*, Berlin, 2018, pp. 569-579. https://doi.org/10.1145/3242587.3242650.

[12] A. Rosenfeld, O. Kardashov, and O. Zang, "Automation of Android applications testing using machine learning activities classification," arXiv preprint arXiv:1709.00928, 2017. https://doi.org/10.1145/3197231.3197241.

[13] B. Wang, et al., "Screen2words: Automatic mobile UI summarization with multimodal learning," *Proceedings of the 34th Annual ACM Symposium on User Interface Software and Technology*, 2021, pp. 1-13. https://doi.org/10.1145/3472749.3474765.

[14] H. Wen, et al., "AutoDroid: LLM-powered task automation in Android," *ACM MobiCom '24: Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*, 2024, 543-557. https://doi.org/10.1145/3636534.3649379.

[15] L. Zhang, et al., LlamaTouch: A Faithful and Scalable Testbed for Mobile UI Task Automation, *Proceedings of the UIST'24*, October 13–16, 2024, Pittsburgh, PA, USA, pp. 1-13. https://doi.org/10.1145/3654777.3676382.

[16] A. Shirazi, et al., "Insights into layout patterns of mobile user interfaces by an automatic analysis of Android apps," *Proceedings of the 5th ACM SIGCHI Symposium on Engineering Interactive Computing Systems EICS'13*, 2013, pp. 275-284. https://doi.org/10.1145/2494603.2480308.

[17] B. Deka, Z. Huang, and R. Kumar, "ERICA: Interaction mining mobile apps," Proceedings of the 29th Annual Symposium on User Interface Software and Technology, 2016, pp. 767-776. https://doi.org/10.1145/2984511.2984581.

[18] L. Leiva, A. Hota, and A. Oulasvirta, "Enrico: A dataset for topic modeling of mobile UI designs," Proceedings of the 22nd International Conference on Human-Computer Interaction with Mobile Devices and Services MobileHCI'20, 2020, pp. 1-4. https://doi.org/10.1145/3406324.3410710.

[19] A. Lavanya, et al., "Assessing the performance of Python data visualization libraries: A review," *International Journal of Computer Engineering in Research Trends (IJCERT)*, vol. 10, no. 1, pp. 28–39, 2023. https://doi.org/10.22362/ijcert/2023/v10/i01/v10i0104.

[20] T. Chen, and C. Guestrin, "XGBoost: A scalable tree boosting system," *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 785–794. https://doi.org/10.1145/2939672.2939785.

[21] J. Friedman, "Greedy function approximation: A gradient boosting machine," *The Annals of Statistics*, vol. 29, no. 5, pp. 1189-1232, 2001. https://doi.org/10.1214/aos/1013203451.

[22] L. Breiman, "Random Forests," *Machine Learning*, vol. 45, issue 1, pp. 5-32, 2001. https://doi.org/10.1023/A:1010933404324.

[23] M.-C. Popescu, et al., "Multilayer perceptron and neural networks," WSEAS Transactions on Circuits and Systems, vol. 8, issue 7, pp. 579-588, 2009.

[24] Y. Freund, R. E. Schapire, "A desicion-theoretic generalization of on-line learning and an application to boosting," In: Vitányi, P. (eds) Computational Learning Theory. EuroCOLT 1995. *Lecture Notes in Computer Science*, vol 904, 1995. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-59119-2_166.

[25] J. Peng, K. Lee, and G. Ingersoll, "An introduction to logistic regression analysis and reporting," *Journal of Educational Research*, vol. 96, no. 1, p. 3-14, 2002. https://doi.org/10.1080/00220670209598786.

[26] L. Rokach, O. Maimon, Decision Trees, In: Maimon, O., Rokach, L. (eds) Data Mining and Knowledge Discovery Handbook, 2005, pp. 165-192, Springer, Boston, MA. https://doi.org/10.1007/0-387-25465-X_9.

[27] I. Rish, "An empirical study of the Naive Bayes classifier," *Proceedings of the IJCAI 2001 Workshop on Empirical Methods in Artificial Intelligence,* Seattle, 4 August 2001, pp. 41-46.

[28] C. Cortes, and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, issue 3, pp. 273-297, 1995. https://doi.org/10.1023/A:1022627411411.
[29] D. Valero-Carreras, J. Alcaraz, and M. Landete, "Comparing two SVM models through different metrics based on the confusion matrix," *Computers & Operations Research*, vol. 152, pp. 106131, 2023. https://doi.org/10.1016/j.cor.2022.106131.
[30] J. Li, H. Sun, and J. Li, "Beyond confusion matrix: Learning from multiple annotators with awareness of instance features," *Machine Learning*, vol. 112, issue 3, pp. 1053-1075, 2023. https://doi.org/10.1007/s10994-022-06211-x.
[31] S. Narkhede, "Understanding auc-roc curve," *Towards Data Science*, vol. 26, issue 1, pp. 220-227, 2018.

***MOHEB R. GIRGIS** received his B.Sc. Degree from Mansoura University, Egypt, in 1974, M.Sc. degree from Assiut University, Egypt, in 1980, and Ph.D. degree from the University of Liverpool, England, in 1986. He is a professor of computer science at Minia University, Egypt. His research interests include software engineering, software testing, information retrieval, evolutionary algorithms, image processing, computer networks, and bioinformatics.*

***ALAA M. ZAKI** received his B.Sc. Degree from Minia University, Egypt, in 1999, M.Sc. degree from Mina University, Egypt, in 2009, and Ph.D. degree from Minia University, Egypt, in 2015. He is an associate professor of computer science at Minia University, Egypt. His research interests include Software Engineering, Data Mining, evolutionary algorithms, computer networks, and bioinformatics.*

***ENAS ELGELDAWI** received her B.Sc. Degree from Minia University, Egypt, in 1999, M.Sc. degree from Mina University, Egypt, in 2005, and Ph.D. degree from Minia University, Egypt, in 2015. She is an associate professor of computer science at Minia University, Egypt. Her research interests include Data Mining, evolutionary algorithms, computer networks, and bioinformatics.*

***MOHAMED M. ABDALLAH** received his B.Sc. Degree from Minia University, Egypt, in 2004, M.Sc. degree from Mina University, Egypt, in 2008, and Ph.D. degree from Minia University, Egypt, in 2015. He is an assistant professor of computer science at Minia University, Egypt. His research interests include information retrieval, Data Mining, evolutionary algorithms, computer networks, and bioinformatics.*

***ALI A. AHMED** received his B.Sc. Degree from Minia University, Egypt, in 2015, M.Sc. degree from Minia University, Egypt, in 2021. He is an Assistant lecturer of computer science at Minia University, Egypt. His research interests include Software Engineering, machine learning, deep learning, big data analytics, and mobile applications development testing*