GUing: A Mobile GUI Search Engine using a Vision-Language Model

Jialiang Wei, Anne-Lise Courbis, Thomas Lambolais, Binbin Xu, Pierre Louis Bernard, Gérard Dray, Walid Maalej

GUing: A Mobile GUI Search Engine using a Vision-Language Model

JIALIANG WEI, EuroMov Digital Health in Motion, Univ Montpellier, IMT Mines Ales, France
ANNE-LISE COURBIS, EuroMov Digital Health in Motion, Univ Montpellier, IMT Mines Ales, France
THOMAS LAMBOLAIS, EuroMov Digital Health in Motion, Univ Montpellier, IMT Mines Ales, France
BINBIN XU, EuroMov Digital Health in Motion, Univ Montpellier, IMT Mines Ales, France
PIERRE LOUIS BERNARD, EuroMov Digital Health in Motion, Univ Montpellier, IMT Mines Ales, France
GÉRARD DRAY, EuroMov Digital Health in Motion, Univ Montpellier, IMT Mines Ales, France
WALID MAALEJ, University of Hamburg, Germany

Graphical User Interfaces (GUIs) are central to app development projects. App developers may use the GUIs of other apps as a means of requirements refinement and rapid prototyping or as a source of inspiration for designing and improving their own apps. Recent research has thus suggested retrieving relevant GUI designs that match a certain text query from screenshot datasets acquired through crowdsourced or automated exploration of GUIs. However, such text-to-GUI retrieval approaches only leverage the textual information of the GUI elements, neglecting visual information such as icons or background images. In addition, retrieved screenshots are not steered by app developers and lack app features that require particular input data.

To overcome these limitations, this paper proposes GUing, a GUI search engine based on a vision-language model called GUIClip, which we trained specifically for the problem of designing app GUIs. For this, we first collected from Google Play app introduction images which display the most representative screenshots and are often captioned (i.e. labelled) by app vendors. Then, we developed an automated pipeline to classify, crop, and extract the captions from these images. This resulted in a large dataset which we share with this paper: including 303k app screenshots, out of which 135k have captions. We used this dataset to train a novel vision-language model, which is, to the best of our knowledge, the first of its kind for GUI retrieval. We evaluated our approach on various datasets from related work and in a manual experiment. The results demonstrate that our model outperforms previous approaches in text-to-GUI retrieval achieving a Recall@10 of up to 0.69 and a HIT@10 of 0.91. We also explored the performance of GUIClip for other GUI tasks including GUI classification and sketch-to-GUI retrieval with encouraging results.

CCS Concepts: • **Human-centered computing** → **Graphical user interfaces**; • **Software and its engineering** → **Designing software**.

Additional Key Words and Phrases: Vision-Language Model, GUI Prototyping, Information Retrieval, Requirements Engineering

ACM Reference Format:

Jialiang Wei, Anne-Lise Courbis, Thomas Lambolais, Binbin Xu, Pierre Louis Bernard, Gérard Dray, and Walid Maalej. 2024. GUing: A Mobile GUI Search Engine using a Vision-Language Model. 1, 1 (October 2024), 29 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Authors' addresses: Jialiang Wei, jialiang.wei@mines-ales.fr, EuroMov Digital Health in Motion, Univ Montpellier, IMT Mines Ales, Ales, France; Anne-Lise Courbis, anne-lise.courbis@mines-ales.fr, EuroMov Digital Health in Motion, Univ Montpellier, IMT Mines Ales, Ales, France; Thomas Lambolais, thomas.lambolais@mines-ales.fr, EuroMov Digital Health in Motion, Univ Montpellier, IMT Mines Ales, Ales, France; Binbin Xu, binbin.xu@mines-ales.fr, EuroMov Digital Health in Motion, Univ Montpellier, IMT Mines Ales, Ales, France; Pierre Louis Bernard, pierre-louis.bernard@umontpellier.fr, EuroMov Digital Health in Motion, Univ Montpellier, IMT Mines Ales, Montpellier, France; Gérard Dray, gerard.dray@mines-ales.fr, EuroMov Digital Health in Motion, Univ Montpellier, IMT Mines Ales, Ales, France; Walid Maalej, walid.maalej@uni-hamburg.de, University of Hamburg, Hamburg, Germany.

Manuscript submitted to ACM

1 INTRODUCTION

The Graphical User Interfaces (GUI) of mobile apps serve as the primary means of interaction between users and their devices. A well-designed GUI streamlines the navigation process, facilitates the accomplishment of user tasks, and improves the overall user experience — contributing towards a higher user engagement and retention [13, 28]. Furthermore, a modern, attractive, and user-friendly GUI can potentially differentiate an app from its market counterparts, amplifying its likelihood of success in the highly competitive app market [46, 53]. To design a good GUI, developers often create multiple prototypes with varying fidelity: from low-fidelity sketches for brainstorming to high-fidelity GUIs for testing and optimisation. GUI prototypes are used in interviews or workshops to discuss, refine, and validate requirements, helping reduce misunderstanding and ultimately saving resources.

In this context, app developers often explore other GUIs of related apps as a way for rapid prototyping and a source of inspiration to design and improve their own apps. Numerous approaches have thus been suggested to retrieve relevant GUIs from existing apps. Researchers have proposed GUI retrieval approaches that accept sketches [29, 48, 49], wireframes [12, 16, 40], or screenshots [8, 37] as input to locate similar or fitting designs. While certainly useful, these approaches require a preliminary graphical prototype, which might not be available or might be too restrictive in early ideation phases. Therefore, when only general ideas and textual requirements are available, text-to-GUI retrieval approaches would be particularly useful.

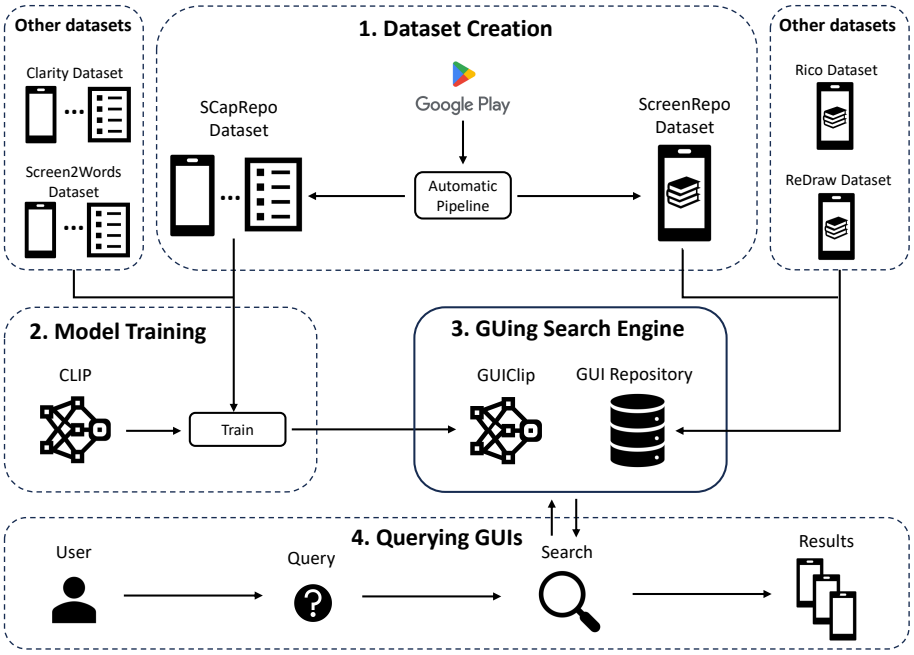
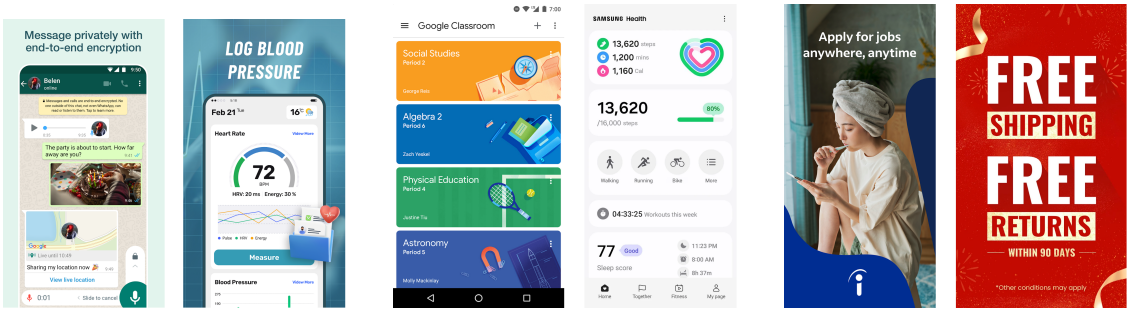


Fig. 1. Overview of our approach: including the creation of the datasets, the training of the vision-language model GUIClip, the development of the GUI search engine, and the process of querying on the engine.

Recent GUI search engines, such as GUIGLE [6] and RaWi [31], enable their users to search GUI datasets using text queries. Both engines use app metadata and GUI-related text for query matching. GUIGLE adopts basic keyword matching, while RaWi calculates semantic similarity based on a BERT model [17]. However, these text-based retrieval

approaches solely use textual information available within the GUI such as text blocks or button labels, neglecting key visual information such as images, layouts, and backgrounds. Furthermore, the underlying GUI datasets (namely Rico [16] and ReDraw [52]) are created by crowdsourced or automated exploration of app screens at runtime. However, the access to certain screens may require authorisation or initial configurations, which is often skipped in automated exploration. Even with a crowdsourced exploration, some app features may not be captured, particularly those requiring substantial or specific input data, such as dashboard pages with charts. Thus, screenshots crawled at runtime may fail to comprehensively capture essential features of the app.

In this paper, we propose a novel GUI search engine based on a vision-language model trained with app introduction images from Google Play as shown on Figure 1. Recent vision-language models, such as CLIP [59], BLIP [36], and BLIP-2 [35], are trained on large-scale image-caption data using contrastive learning. These models have the ability to transform images and text into a *multimodal* embedding, ensuring that semantically similar images and texts are mapped closely in the embedding space. Thus, by computing the similarity between the images and the textual query, these models can be used for text-to-image retrieval tasks. As prerequisite a large screenshot-caption dataset is needed. Since, currently available datasets (Screen2Words [71] and Clarity [54]) are inadequate for training a vision-language model (as our evaluation shows in Section 5), we have created a new large screenshot-caption dataset.



(a) Surrounded Screenshots (b) Screenshots (c) Irrelevant

Fig. 2. Examples of app introduction images from Google Play app store.

Google Play is one of the largest mobile app stores with thousands of apps from diverse domains. This makes it a rich source of inspiration for requirements elicitation and app design [23, 43]. Particularly, the app introduction images on Google Play are a gold mine for GUI retrieval, as they are carefully selected and described by developers to represent the important app features. Figure 2 shows examples of these images. A large portion can be categorised as *surrounded screenshots*, each displaying a screenshot and a succinct caption that describes the screen (Figure 2 (a)). In order to extract the screenshots and respective captions, we developed an automated **pipeline**, optimised for these images. The pipeline first classifies images into three categories as shown in Figure 2. Then, from the *surrounded screenshots* it crops the screenshot areas and extracts the captions (like "LOG BLOOD PRESSURE" in Figure 2 (a)).

By applying the pipeline on the introduction images of approximately 117k apps, we created a comprehensive dataset comprising 303k screenshots, of which 135k have captions. We refer to the collection of 303k screenshots as the **ScreenRepo** and the subset containing captions as **SCapRepo**. We then used SCapRepo together with the Screen2Words and Clarity datasets to fine-tune the CLIP model and create a vision-language model specific to the GUI domain. We call the new model **GUIClip**. Our evaluation results demonstrate a promising performance of GUIClip

for text-to-image GUI retrieval, outperforming both text-only approaches and the CLIP model. Based on GUIClip, we developed **GUing**, a search engine that accepts textual queries as input to retrieve relevant GUI images from our dataset as well as the Rico and ReDraw datasets. The engine can easily be extended with additional screenshots. Our manual evaluation shows that the search engine recommends highly relevant GUIs, which can serve as a valuable source of design inspiration and a tool for rapid prototyping and requirements refinement. Two additional experiments suggest that GUIClip is also beneficial for other GUI-related tasks, such as sketch-to-GUI retrieval and GUI classification. The paper provides the following contributions, with source code and datasets publicly available for research purposes https://github.com/Jl-wei/guing:

- A vision-language model named GUIClip for a range of GUI-related tasks, including text-to-GUI retrieval, sketch-to-GUI retrieval, and GUI classification available at https://huggingface.co/Jl-wei/guiclip-vit-base-patch32.
- A GUI search engine that can achieve high performance with textual queries.
- Two large GUI datasets containing 303k screenshots, 135k of which include captions.
- An extensible pipeline for automatically extracting screenshots and captions from app introduction images.

2 DATASET CREATION

Created two datasets: (1) *Google Play Screenshot Caption (SCapRepo)* including 135k screenshot-caption pairs for the training of our vision-language model and (2) Google Play Screenshot Repository (ScreenRepo) including 303k screenshots for the search engine repository. Figure 3 depicts the datasets creation pipeline. In the initial step, we collected app introduction images from Google Play. Subsequently, we developed an image classifier to categorise the images into *screenshots*, *surrounded screenshots* and *irrelevant*. The area in a *surrounded screenshot* containing a screenshots was precisely cropped from the surrounding by applying object detection. Additionally, we extracted the captions from the *surrounded screenshots* by applying Optical Character Recognition (OCR). The images classified as *screenshots* together with the cropped screenshots constitute the ScreenRepo, while the screenshot-caption pairs extracted from *surrounded screenshots* constitute the SCapRepo. The following describes each step in detail.

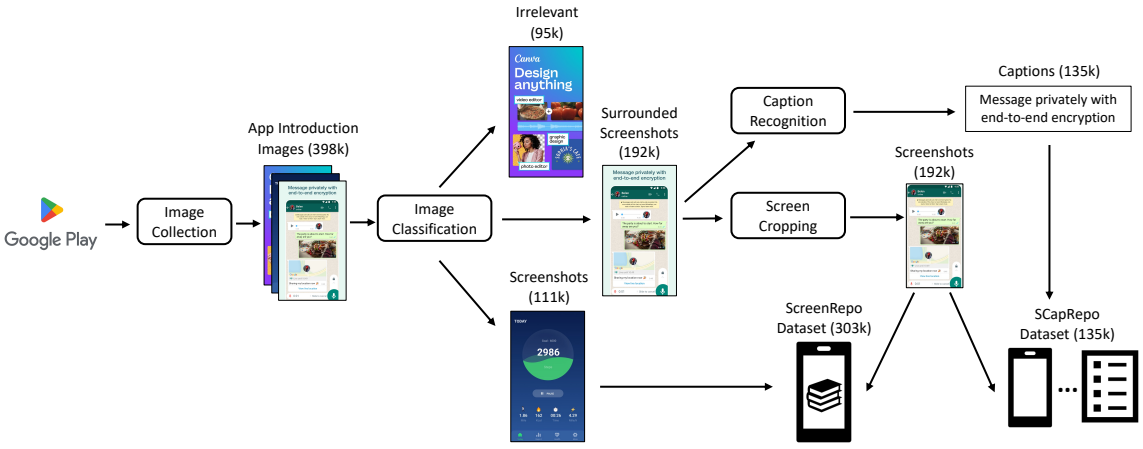


Fig. 3. Overview of dataset creation pipeline.

2.1 Image Collection

Given the ID of an app, we can download the app introduction images via Google Play Scraper [55]. In order to obtain the app IDs, we initially gathered top ranked apps in each category from AppBrain [2], creating a seed list. An app on Google Play often suggests similar apps and the app developer who may also offer additional apps. This scenario can be conceptually treated as a graph, where apps represent nodes and the relationships (such as app similarity and common developer) represent edges. Starting from the seed list, we performed a breadth-first search on this graph, collecting a total of 117,283 apps (by October 2023). All apps were collected from the US Google Play in English language.

The stylistic differences between the introduction images of games and non-game apps make the former unsuitable as reference for general app design. Consequently, we excluded games from our analysis, leaving a dataset of 85,631 non-game apps. This includes a total of 874,130 images. Some of these images specifically demonstrates landscape views, Wear OS, or Android TV [24]. To exclude those from our dataset, we applied a filtering process based on aspect ratio, leading to 553,755 precisely filtered images. Additionally, some images could be duplicates despite having different filenames. To remove duplicates, we calculated the SHA-1 hash of each individual image, retaining only one image with the same hash. This yielded a final count of 398,511 unique app introduction images.

2.2 Image Classification

For subsequent analysis, we classified the images obtained from the prior phase into three categories as depicted in Figure 3. A subset of these images show the complete display of an app user interface: those are *screenshots*. In some images, only a segment represents a captured screenshot. Those are categorised as *surrounded screenshots*. The remaining images either do not show a screenshot or only a partial or a slanted screenshot. Those are classified as *irrelevant*. As a manual classification of around 400k images is unreasonable, we trained an image classifier to automatically classify these images into *screenshots*, *surrounded screenshot*, or *irrelevant*.

2.2.1 Classification Model. We used the Vision Transformer (ViT) [18] as classifier due to its top performance (as will be discussed in subsequent sections). The ViT architecture essentially constitutes the encoder of transformer as described by Vaswani et al. [70]. It has demonstrated superior performance in image classification tasks. In the area of natural language processing, the dominant approach is to pre-train the model on a large text corpus followed by fine-tuning on a more targeted dataset [27]. Similarly, the ViT benefits from a pre-training phase using a comprehensive collection of images, allowing the model to capture inherent image features. Consequently, when deployed on novel tasks, the ViT requires a reduced amount of task-specific training due to the knowledge acquired during its pre-training.

2.2.2 Creation of Training Data. To train and fine-tune the classification model, a labelled dataset is required. For this, we initially sampled 5,000 random images from the collected filtered dataset (see Section 2.1). After eliminating duplicates, 3,615 unique images remained. To annotate these images, we created an annotation guide with examples to clarify the definition of *screenshots*, *surrounded screenshot*, and *irrelevant*. Two of the authors then annotated these images independently. For the entire labelled dataset, only 7 images had conflicting labels, primarily due to disagreements between the *surrounded screenshot* and *irrelevant* categories. These conflicts arose because the images in question did not contain a full screenshot but rather a UI component, such as a widget. After discussion, we reached a consensus to label images featuring a larger UI component (as thus potentially inspiring for a screen design) as *surrounded screenshot*, and the others as *irrelevant*.

Table 1. Accuracy of the image classification (mean values for 10-fold cross-validation).

Class	Precision	Recall	F1
Surrounded Screenshot	0.974	0.978	0.976
Screenshot	0.967	0.970	0.968
Irrelevant	0.937	0.926	0.931
Weighted Average	**0.964**	**0.964**	**0.964**

2.2.3 Classification Performance. We evaluated the performance of the classifier using the labelled app introduction images from the previous step, which we split into a 80:20 ratio for training and testing, respectively. The goal of training a machine learning (ML) model is to minimise the loss function, which quantifies the difference between the model's predicted output and the actual target values. We trained the classification model using mini-batch gradient descent, with a batch size[1] of 64, AdamW [41] optimiser[2], and an initial learning rate[3] of $2e^{-5}$. The classifier was trained in 5 epochs on a machine with a NVIDIA Tesla T4 GPU with 16 GB VRAM. To enhance the robustness of our results, we performed a 10-fold cross-validation by repetitively splitting the dataset into distinct training and testing subsets ten times in a randomised manner. We then computed the mean values of the *precision, recall*, and *F1 score* across all runs. The results presented in Table 1 show a top performance of our classifier, with an average F1 score of 0.964.

2.2.4 Applying the Classifier. Subsequent to the evaluation, we trained a ViT model on all of the 3,615 images. This model was then used to classify all of the filtered app introduction images in the dataset. To increase the reliability and integrity of our repository, we calibrated the classifier's threshold at a high confidence level of 0.9, which means that only images with a classification probability exceeding 0.9 are included in the respective category. Our empirical investigation reveals that, at the 0.9 threshold, the precision scores for the categories *screenshots* and *surrounded screenshots* surpass 0.99. Finally, our repository had 111,847 images classified as *screenshots* and 191,993 as *surrounded screenshots*.

2.3 Screen Cropping

The *surrounded screenshots* are the images that contain screenshots with additional visual frames. To automatically crop the screenshot areas from these *surrounded screenshots*, we trained an object detection model. This model is capable of localising the screenshot within the larger image and subsequently generating a bounding box. A bounding box is defined as a rectangular delineation that encases the area of interest: in this case, the screenshot. Given the bounding boxes inside the *surrounded screenshots*, we can easily crop the screenshot area using an image processing library.

2.3.1 Object Detection Model. We use DETR (DEtection TRansformer) [9] as screenshot detector. DETR is an end-to-end object detector, composed of a Convolutional Neural Network (CNN) backbone and a encoder-decoder transformer [70]. Atop the transformer decoder, dual output heads are appended: a linear layer dedicated to categorise class labels and a multi-layer perceptron (MLP) tasked with the generation of bounding boxes for the object location. The model uses so-called object queries to detect objects in an image. Each object query is designed to search specifically for one particular object of interest in the given image. We set the number of queries to 1, as we are interested in the largest screenshot within a *surrounded screenshot*.

[1]Batch size refers to the number of data samples processed simultaneously during a single training step in ML.
[2]Optimisers, like AdamW, are strategies used to minimise the loss function.
[3]Learning rate is a parameter in ML algorithms, controlling the training step size while moving toward a minimum of a loss function.

Table 2. Accuracy of the screen localisation (mean values for 10-fold cross-validation).

Precision	IoU=0.50:0.95	0.919
	IoU=0.50	0.981
	IoU=0.75	0.971
Recall	IoU=0.50:0.95	0.947

2.3.2 Creation of Training Data. We created a separate dataset to fine-tune the detection model for screenshot cropping. From the dataset created in Section 2.2.2, we collected all the *surrounded screenshots* and labelled the screenshot areas with a bounding box. The annotation was performed with Prodigy [21]. For each image, an author drew one bounding boxes that cover the screenshot. The results were reviewed by another author. The majority of the *surrounded screenshots* feature only a single screenshot. In instances where a single image contains multiple screenshots, annotation was selectively applied only to the most prominent (largest) screenshot. This resulted in a dataset comprising 1,768 annotated images, each including a delineated bounding box that specifies the screenshot area.

2.3.3 Detection Performance. Intersection over Union (IoU) is a standard metric for object detection tasks. It evaluates the extent of overlap between the predicted bounding box and the ground truth bounding box. If $IoU = 0$, it means that there is no overlap between the boxes, while $IoU = 1$ means that the overlap is perfect.

$$IoU = \frac{AreaOfOverlap}{AreaOfUnion}$$

We followed the evaluation protocol of the Common Objects in Context (COCO) [39], which is a common object detection benchmark. As stated in the protocol, we quantitatively evaluated the object detection performance by calculating precision and recall metrics at various IoU thresholds. Specifically, we computed the precision values at three IoU levels: 0.50, 0.75, and an average over the range from 0.50 to 0.95. Similarly, recall was determined over the IoU range of 0.50 to 0.95.

We used 80% of the data for training and the remaining 20% for evaluation. The detector was trained on the training set using mini-batch gradient descent, with a batch size of 16 and AdamW optimiser with an initial learning rate of $1e^{-5}$ and ten training epochs. Table 2 shows the results for a 10-fold cross-validation, indicating a top performance.

2.3.4 Applying the Detector. We trained a DETR model with all of the 1,768 labelled images, and applied this model to the 191,993 images categorised as *surrounded screenshot*. DETR predicted a bounding box that localise the screenshot for each mage. We then cropped these images according to the bounding box. The screenshots cropped from the *surrounded screenshot* as well as the app introduction images previously classified as *screenshots* represent the ScreenRepo dataset.

2.4 Caption Recognition

The majority of the images, classified as *surrounded screenshots*, include captions that provide a succinct descriptions of the screenshots. Typically, these captions are positioned either above or below the image. In order to accurately extract this textual information, we employed Optical Character Recognition (OCR), a commonly used technology for recognising text displayed within images.

2.4.1 Applying OCR. We used PaddleOCR [34], which is an industrially robust OCR model renowned for its accuracy and swift performance. The application of PaddleOCR is adequate since the textual content within the images under

analysis is markedly legible and usually not hand-written. PaddleOCR processes an input image and outputs a collection of identified elements. Each element is comprised of a bounding box delineating the text area, its recognised text, and an associated confidence level reflecting the recognition certainty. We applied PaddleOCR on all of the 191,993 *surrounded screenshots*. For each *surrounded screenshot*, text bounding boxes that exhibit spatial overlap with the screenshot's bounding box, as delineated in Section 2.3.4, were excluded. The remaining text bounding boxes were subsequently aggregated to form a coherent caption for the corresponding image.

2.4.2 Post-processing. Some of the *surrounded screenshots* may not contain captions. These images were removed from our analysis. The captions extracted are not exclusively in English. They encompass a variety of languages including French, German, and Arabic, among others. As we focus on English in this work, captions in languages other than English were excluded. To identify the language of each caption, we employed Lingua [66], an effective language detection tool. Furthermore, to correct any spelling error within the captions due to potential OCR errors, we utilised Autocorrect [65], a Python-based spelling corrector.

It is noteworthy that in some instances, multiple *surrounded screenshots* of an app may feature identical captions. This suggests that the caption is either overly generic or simply a logo of the app, and thus may not adequately convey the screenshot content. To address this redundancy, we conducted a filtering process whereby, for each app, only the first *surrounded screenshot* from the set containing identical captions was retained. We observed that the first *surrounded screenshot* displayed by an app on Google Play typically showcases the most representative feature. The post-processing finally left 135,357 screenshot-caption pairs, collectively referred to as the SCapRepo dataset.

3 GUI SEARCH ENGINE

We introduce GUing, an advanced search engine based on a vision-language model that leverages textual queries to retrieve relevant screenshots from a large GUI repository which consists of ScreenRepo, Rico [16] and ReDraw [52]. The architecture of GUing is illustrated in Figure 4. GUing is based on the GUIClip model. The engine uses the image encoder and text encoder modules of GUIClip to embed screenshots and textual queries within a unified latent space. This enables an efficient search for the screenshots by calculating the cosine similarity between the text query embedding and the screenshot embedding. In the following, we introduce the details of GUIClip and GUing.

3.1 Constructing the GUIClip Model

GUIClip, our vision-language model, serves as the fundamental component of the GUI search engine, integrating the image and text modalities. Building on CLIP [59], GUIClip offers enhanced capabilities in multimodal representation learning for the GUI domain. CLIP (Contrastive Language–Image Pre-training) is a significant foundation model in multimodal representation learning trained on a large-scale image-caption dataset. In this work, we construct a GUI-specific CLIP model by training a vision-language model on a large-scale dataset of screenshot-caption pairs.

3.1.1 Training Data and Pre-processing. To train GUIClip learn the visual representation of mobile screenshots, we combined three datasets:

- Screen2Words dataset [71] is a subset of Rico. It contains 112,085 textual summaries for 22,417 unique screenshots. The screenshot summaries are written by professional annotators. For each screenshot, five summaries are created by five different annotators.

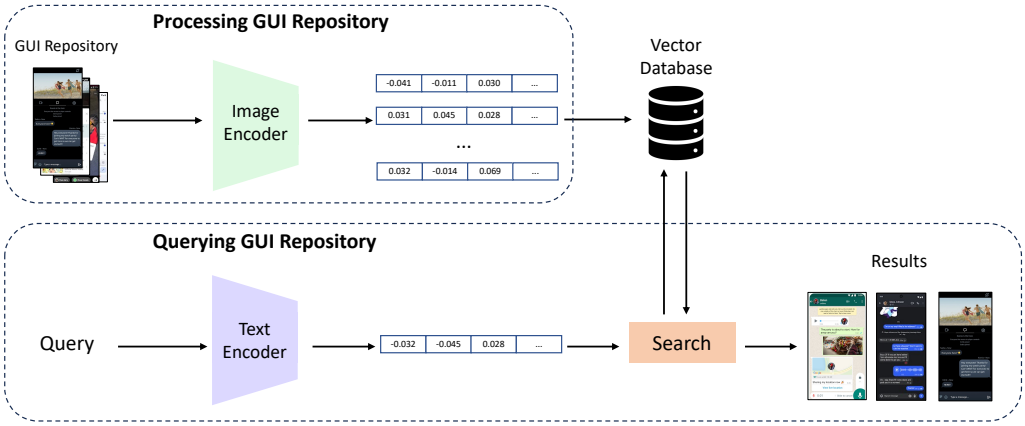


Fig. 4. Overview of GUing, our GUI search engine for text-to-GUI retrieval.

- Clarity dataset [54] consists of 45,998 descriptions for 10,204 screenshots from popular Android apps. Crowd workers summarised each screenshot in different granularity, with one high-level caption and up to four low-level detailed descriptions.
- Google Play Screenshot Caption (SCapRepo) dataset, where textual descriptions are created by developers and app vendors. The dataset includes 135k screenshot-caption pairs (see Section 2). The screenshots are cropped from app introduction images on Google Play. The corresponding captions briefly describe the screenshots.

The captions were first tokenised with the CLIP tokeniser [56]. Captions longer than 77 tokens were truncated to 77, captions shorter were padded to 77. To process the screenshot images, we resized the resolution to 224×224. Furthermore, we proportionately rescaled the pixel value of the images from a range of 0-255 to 0-1. This new value was then normalised using the parameters (means and standard deviations) featured in the official CLIP documentation [56].

3.1.2 Model Architecture. GUIClip consists of two components, an image encoder and a text encoder. The image encoder, which adopts a Vision Transformer (ViT) model [18], ingests image data and generates its corresponding image embedding. The text encoder is a transformer encoder [70] which accepts text as input to produce an associated text embedding. Employing contrastive learning techniques, the image and text embeddings are mapped into a multimodal embedding space. In this shared space, images and texts presenting semantic similarity are mapped close to each other.

Contrastive learning on image-text pairs constructs a bridge between visual perception and natural language. Given a batch of N (image, text) pairs, the training objective is to identify the authentic pairings from $N \times N$ potential (image, text) combinations within the batch [59]. To achieve this objective, the model formulates a multimodal embedding space, concurrently training an image encoder and text encoder to maximise the cosine similarity of the image and text embeddings derived from the N authentic pairs in the batch. Simultaneously, it minimises the cosine similarity of the embeddings from the remaining $N^2 - N$ incorrect pairings.

3.1.3 Training Details. In order to adapt the CLIP model to GUIClip, rather than training the CLIP model from scratch where the weights of image encoder and text encoder are randomly initialised, our training is based on the checkpoint of "openai/clip-vit-base-patch32" [57]. This checkpoint has been pre-trained on a 400 million image-text pairs. This method allows us to harness the benefits inherent to pre-trained models and, as a result, substantially diminishes the

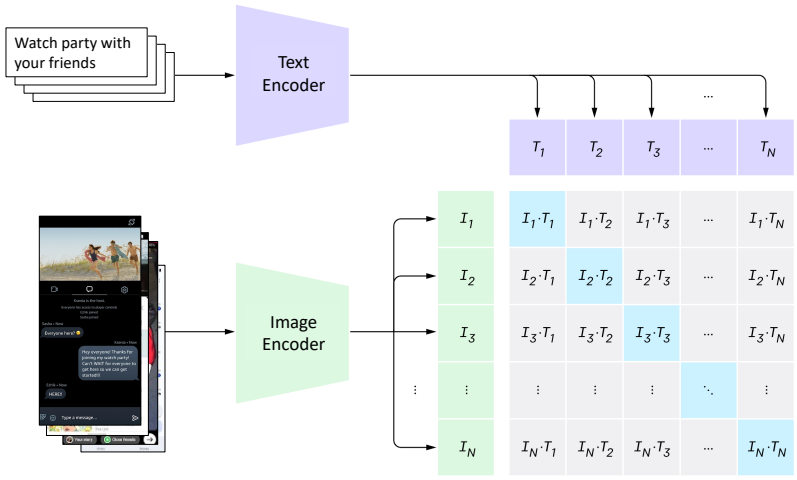


Fig. 5. Overview of contrastive learning to train our vision-language model GUIClip (adapted from Radford et al. [59]).

time required for training. The model was trained on all of the collected screenshot-caption pairs for 5 epochs. We applied mini-batch gradient descent with a batch size of 128 and AdamW optimiser with an initial learning rate of $5e^{-5}$.

3.2 Searching the GUI Repository

3.2.1 Processing the GUI Repository. The GUI repository is composed of 383k screenshots from ScreenRepo (303k), Rico (66k) and ReDraw (14k). These screenshots are stored in image formats such as .jpeg and cannot be directly queried. Therefore, we embedded all the screenshots in the repository and saved them in a vector database.

The screenshots are first processed (i.e. resized, rescaled, and normalised) with the same parameters discussed in Section 3.1.1. The processed screenshots subsequently serve as input for the image encoder of GUIClip. The image encoder accepts an image as an input and returns a 512-dimensional embedding. The entire GUI repository, comprising 383k screenshots, is processed via the image encoder, consequently generating 383k 512-dimensional embeddings. The mapping between the screenshot ids and their embeddings are saved in a vector database for the retrieval.

3.2.2 Querying the GUI Repository. In the querying phase, the textual query is encoded by GUIClip's text encoder, resulting in a 512-dimensional embedding. Subsequently, this query embedding is utilised to identify the top-k most similar screenshot embeddings from the vector database. We use cosine similarity as the measure of similarity. Consequently, the query results consist of the top-k most similar screenshots.

3.2.3 Accelerating the Searching. Given the large scale of the GUI repository, employing a brute force approach to compare the query embedding with all screenshot embeddings is inefficient. To tackle this problem, we applied approximate nearest neighbour search [3] to accelerate the querying. To this end, we established n Voronoi cells within the embedding space, with each screenshot embedding falling into one of these designated cells. During a search, the query embedding is initially compared with the centroid embeddings of each cell. This method results in a considerable reduction of necessary comparisons when contrasted with the total number of screenshot embeddings. The k cells with centroids closest to the query embedding are identified, whereupon the screenshot embeddings within these cells are thereafter compared with the query embedding in order to pinpoint similar screenshot embeddings. Our

implementation is based on Faiss [19], a library for efficient similarity search. We set the number of cells n to 3000, and the number of cells that are visited to perform a search k to 1000.

4 EVALUATION DESIGN

For the empirical evaluation of our search engine as well as our vision-language model, we focus on answering the following research questions:

- **RQ1: How does GUIClip perform in text-to-GUI retrieval tasks compared to state-of-the-art approaches?** As the CLIP model [59] was trained on 400 millions of image-caption pairs from the internet, it can also be applied on text-to-GUI retrieval task. An alternative method involves leveraging only the textual information: i.e. the text displayed on the GUI images, computing its semantic similarity with the input textual query. Thus, the question is whether the performance our proposed GUIClip model surpasses the CLIP model and text-only approaches.
- **RQ2: How relevant are the retrieved screenshots for queries representing app features?** As a search engine, GUing processes textual queries as input and searches for relevant images within the GUI repository. It is thus important to evaluate how effective is GUing in delivering useful results that actually align with user expectations.
- **RQ3: Does GUIClip also show a high performance in other GUI-related tasks?** The CLIP model, as a foundation model, has been extensively applied across various vision-language tasks, consistently delivering promising results [79]. As GUIClip is a fine-tuned version specifically with GUI images, it has the potential to perform well for other GUI-related tasks. Particularly, the question is whether GUIClip can also outperform CLIP in other GUI-related tasks.

To answer these research questions, we designed four experiments. The first two experiments evaluate our approach on text-to-GUI retrieval: the first experiment (Exp1) addresses RQ1 by benchmarking three screenshot-caption datasets, while the second (Exp2) addresses RQ2 through a manual evaluation of search engines. The final two experiments focus on RQ3, assessing the performance of GUIClip on more GUI-related tasks: the third experiment (Exp3) focusing on GUI classification and the fourth (Exp4) on sketch-to-GUI retrieval.

4.1 Exp1: Evaluation of GUIClip for Text-to-GUI Retrieval

In the first experiment, the text-to-GUI retrieval performance of our GUIClip model is evaluated together with baseline models on three distinct datasets. During the evaluation, every screenshot and its associated captions in test dataset are embedded using the models under evaluation. For every single caption, all screenshots within the test dataset are ranked by the cosine similarity of the caption embedding and screenshot embedding. A comparative analysis of the performance of GUIClip against various baselines provide answers to RQ1.

4.1.1 Experimental Data. From each of the screenshot datasets described in Section 3.1.1 (SCapRepo, Screen2Words, and Clarity), we randomly selected 1000 screenshots for validation, 1000 screenshots for test, and the rest for model training, as shown in Table 3. Similar to the approach suggested by Wang et al. [71], we split the data to *not* share the screenshots from the same app across different splits. That is, all the apps and screenshots in the test and validation set were completely unseen during the training. This arrangement allows us to assess how well our model generalises to previously unseen screenshots from unseen apps during the test phase.

Table 3. Statistics of the evaluation datasets.

Split	Dataset	#Apps	#Screenshots	#Captions
Training	SCapRepo	32720	133477	133477
	Screen2Words	5684	20379	101895
	Clarity	3346	8204	36964
Validation	SCapRepo	246	1000	1000
	Screen2Words	281	1000	5000
	Clarity	413	1000	4475
Test	SCapRepo	245	1000	1000
	Screen2Words	286	1000	5000
	Clarity	416	1000	4559

4.1.2 GUIClip Training Details. We trained the "openai/clip-vit-base-patch32" [57] checkpoint on the training set for a total of 5 epochs. During the training, a mini-batch gradient descent approach was employed with a batch size of 128. To optimise the learning process, the AdamW optimiser was utilised with an initial learning rate set to $5e^{-5}$.

4.1.3 Baselines. As highlighted by Bernal-Cardenas et al. [6] and by Kolthoff et al. [31], existing approaches perform their queries based on GUI metadata, which is absent in screenshots collected by processing app introduction images from Google Play. Consequently, we implemented a text-only retrieval approach by ourselves. In addition, we compared our model with (a) the original CLIP model without further fine-tuning and (b) with the CLIP model fine-tuned *without* our SCapRepo data. This results in three baselines:

- *OCR + BGE (Text Only)*: As discussed in the Section 2.4.1, PaddleOCR [34] is an industrial grade OCR library. BGE [76] is a state-of-the-art text embedding model, trained in three steps: pre-training with plain texts, contrastive learning on text pair dataset, and task-specific fine-tuning. First, the screenshots are embedded by using OCR + BGE in three steps: 1) texts displayed on the GUI images are extracted with PaddleOCR; 2) the extracted text is then concatenated into a sentence, separated by semicolons; 3) the sentence is then embedded by BGE. Finally, the captions are directly embedded using the BGE model.
- *CLIP*: The screenshots are embedded by the image encoder of CLIP and the captions are embedded by the text encoder of CLIP. The "openai/clip-vit-base-patch32" checkpoint is used for this evaluation.
- *GUIClip-CS*: This is a fine-tuned model from CLIP using the Clarity and Screen2Words dataset. The training process is exactly the same as for GUIClip, except that our SCapRepo dataset is excluded during the training. The image encoder of GUIClip-CS embeds the screenshots, while its text encoder embeds the captions.

4.1.4 Evaluation Metric. The recall@k is commonly used for evaluating text-to-image retrieval performance [35, 36, 59, 64]. Recall@k is defined as the percentage of captions whose corresponding screenshots fall into its top-k most similar screenshots. For our evaluation, we used recall@1, recall@3, recall@5, recall@10, recall@50 and recall@100.

4.2 Exp2: Manual Evaluation of GUing

A GUI search engine accepts textual queries as input and identifies the corresponding images within the GUI repository as output. Exp1 focuses on evaluating the performance of retrieval models on the test datasets. Thereby, the main limitation is the use of the screenshot-caption pairs as ground truth, assuming a direct one-to-one correspondence

between a single caption (search query) and a relevant screenshot. However, such assumption contradicts real-world scenarios where one query can pertain to multiple screenshots.

To address this limitation, our second experiment aims to evaluate different search engines composed of various retrieval models and GUI repositories, through a manual assessment. Specifically, we measure the proportion and ranks of images retrieved by the GUI search engines that are relevant to the query entered by the user. Through a comparative analysis of the results usefulness between GUing and baseline approaches, we are then able to address RQ2.

4.2.1 Baselines. To answer RQ2, we compared our search engine to two baselines: RaWi and GUing-CS.

- *RaWi.* RaWi [31] is a data-driven GUI prototyping approach that retrieves screenshots for reuse from Rico dataset based on natural language searches. The search engine of RaWi is a BERT-based [17] Learning-To-Rank (LTR) model that is trained on GUI relevance data collected from crowd-workers. As Rico dataset provides the metadata of the screenshots, RaWi creates the textual representation of the screenshots by assimilating elements such as the activity names, UI component text, resource ids, and icon ids. This integrated textual representation is then compared with the user text query, forming the basis for the GUI ranking.
- *GUing-CS.* The architecture of GUing-CS replicates that of GUing. However, the retrieval model of GUing-CS is GUIClip-CS, which only uses the Clarity and Screen2Words datasets for training. That is, we explicitly remove our SCapRepo dataset from the training process. Furthermore, the GUI repository for GUing-CS is comprised exclusively of the Rico and ReDraw datasets, without the ScreenRepo dataset.

4.2.2 Evaluation Method. To compare the search engines, we developed a tool as shown on Figure 6. The tool accepts textual queries as input and produces the top-10 screenshots from the three search engines: GUing, GUing-CS, and RaWi. The tool mixes and randomly shuffles the resulting 30 screenshots to prevent possible biases by the evaluators. The origin of each search results is unclear and cannot be guessed by the evaluator. As shown on Figure 6, GUIs might be displayed multiple times in the result due to their retrieval by more than one search engine.

During the assessment, the evaluators are asked to search for provided text queries, select all screenshots they consider relevant and confirm their selections via the submission button. A screenshot is relevant to a query if it either displays a UI implementation of the feature delineated in the query or if it can serve as a reference or inspiration for the development of that feature.

The evaluation queries were derived from articles by AttractGroup and BuildFire, which are two companies specialised in mobile app development services and have substantial knowledge in this field. AttractGroup's article enumerates 138 mobile app features to be considered during mobile app development, consisting of 45 general features and 93 domain specific features spanning across 11 domains (such as E-learning, mHealth, FinTech, etc.) [69]. BuildFire's article proposes 50 innovative app ideas for 2024, forming a good source of inspiration for prospective entrepreneurs [30]. From these two resources, we collected a total of 188 mobile app features. These were then divided into two sets: one set consisting of the 45 general and 50 innovative features, and another set comprising 93 domain-specific features.

Four evaluators took part in this experiment, each holding a master degree in computer science and having a minimum of five years of software development experience. Two of them performed the evaluation with the first set of queries, the other two used the second set of queries. This ensured that each query was evaluated independently by two evaluators. The evaluators performed the search and submitted for their respective results as described above.

4.2.3 Evaluation Metric. To measure the performance of the GUI search engine, we applied the common information retrieval metrics Precision@k, Mean Reciprocal Rank, and HITS@k (similar to Kolthoff et al. [31]):

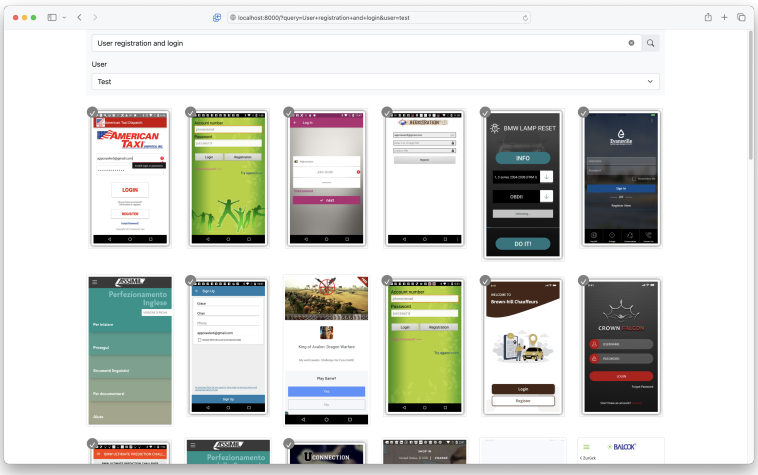


Fig. 6. Interface of the evaluation tool. It displays repeated GUIs due to their retrieval by more than one search engine.

- *Precision at k (P@k)* quantifies the proportion of retrieved screenshots considered relevant, denoted as $|R_k|$, with respect to the top-k retrieved GUIs, denoted as k.

$$P@k = \frac{|R_k|}{k} \tag{1}$$

- *Mean Reciprocal Rank (MRR)* is a measure for evaluating any process that produces an ordered list of possible responses to a sample of queries. $rank_i$ signifies the rank of the highest ranked relevant screenshot of the i-th query. Q stands for the total number of queries.

$$MRR = \frac{1}{Q}\sum_{i=1}^{|Q|}\frac{1}{rank_i} \tag{2}$$

- *HITS@k* is a binary measure that validates the presence of at least one relevant GUI among the top-k ranked screenshots. It has the value 1 if there is at least one relevant screenshot within the top-k, otherwise, it is 0.

$$HITS@k = \begin{cases} 1 & |R_k| > 1 \\ 0 & |R_k| = 0 \end{cases} \tag{3}$$

4.3 Exp3: Evaluation of GUIClip for GUI Classification

GUI classification aims to categorise a screenshot as a whole under a specific label. It can be a fundamental activity for other GUI-related tasks, like GUI captioning [33], the tagging of user screenshots (e.g. submitted in support requests) with certain requirements, components, or devices [44, 45, 61], as well as an early prototyping by a larger group and multiple screens [58]. In this section, we explore the performance of GUIClip on GUI classification under zero-shot and linear-probe setting using the Enrico dataset [32].

4.3.1 Model Architecture. Like Radford et al. [59], we evaluated the performance of GUIClip for GUI classification under zero-shot and linear-probe settings as shown on Figure 7. Under both settings, the image encoder is frozen.

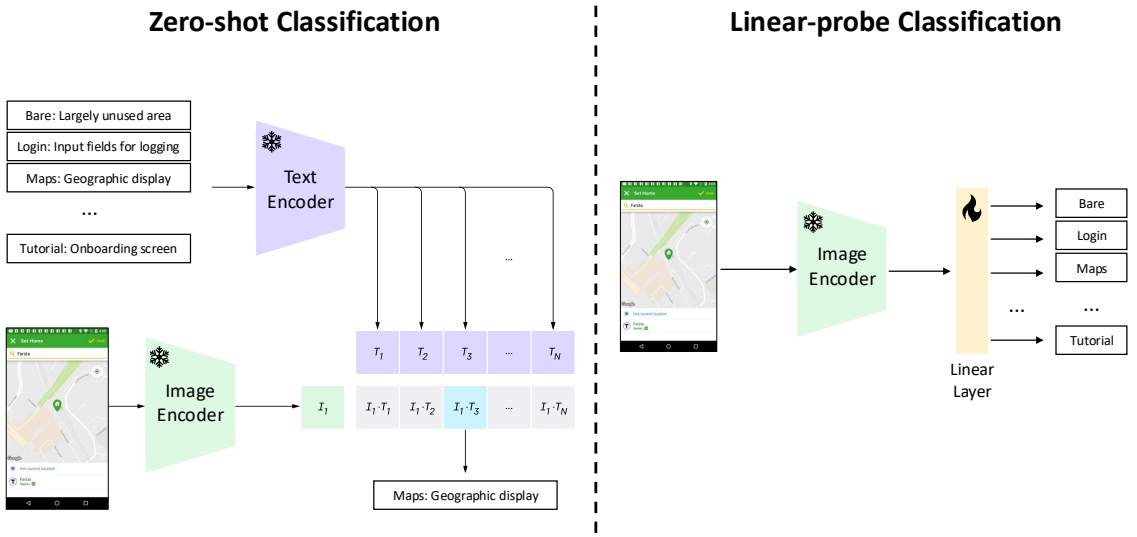


Fig. 7. Overview of zero-shot and linear-probe classification on Enrico dataset with GUIClip (left side adapted from Radfort et al. [59]).

- *Zero-shot*. In the zero-shot setting, all classification labels are embedded to text embedding with the text encoder. During the classification process, the screenshot image is encoded by the image encoder, and the resulting screenshot embedding is compared to all text label embeddings. The label exhibiting the highest similarity to the GUI image embedding is subsequently declared as the predicted outcome.
- *Linear-probe*. In the linear-probe setting, we added a linear layer of shape $m \times k$ to the output of GUIClip image encoder. m refers to the output dimension of image encoder, while k is the number of classification labels.

4.3.2 Baselines. We compared GUIClip performance with two baselines for zero-shot and linear-probe classification:

- *CLIP*. We also evaluated the performance of CLIP on GUI classification. The setting of zero-shot and linear-probe evaluation of CLIP is identical to the setting of GUIClip. Their only difference is the parameters' weights of the two models.
- *OCR + BGE*. Rather than encoding the screenshot image, this method focuses on the textual information presented on the screenshots. As described on the Section 4.1.3, the texts displayed on screenshots are extracted with PaddleOCR [34], then concatenated into sentence and embedded by BGE [76].

4.3.3 Dataset. We used the Enrico dataset to evaluate the model's performance for GUI classification. Leiva et al. [32] used a random sample of 10k screenshots from Rico dataset to create Enrico (shorthand of Enhanced Rico): a human annotated dataset comprising 1460 screenshots and 20 design topics, including camera, chat, media player, etc. In the linear-probe setting, the Enrico dataset was divided into an 80:20 ratio to establish a training set and a test set. For the zero-shot setting, all available data was employed solely for testing.

4.3.4 Training Details of Linear-probe Classification. For the training and evaluation of linear-probe classifier, we split the dataset into training and test set in a ratio of 80:20 in a stratified manner. During training, the image encoder is frozen, the output embedding is served as input of the final linear layer. We trained the model for 100 epochs

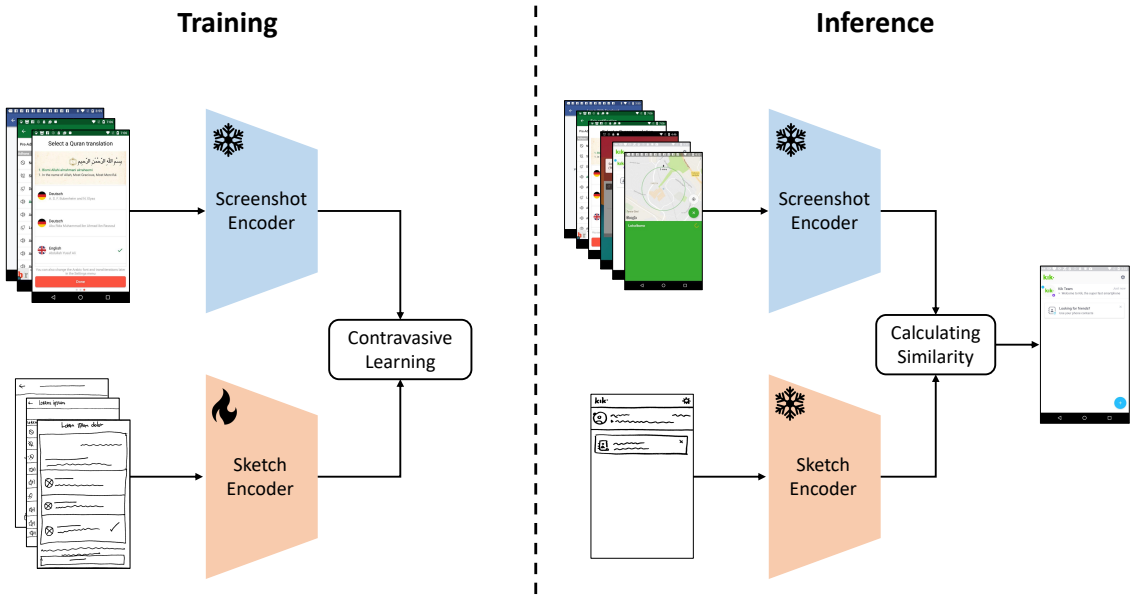


Fig. 8. Overview of the training and inference of Dual GUIClip ViT model for sketch-to-GUI retrieval.

using AdamW optimiser with an initial learning rate $1e^{-5}$, batch size 128. We conducted a 10-fold cross-validation, which involved randomly splitting the training and testing sets ten times, and subsequently computing the average performance spanning across these runs.

4.3.5 Evaluation Metric. The evaluation of the classifiers is performed based on precision, recall, and the F1 score. Given the relatively small size of the Enrico dataset, especially when compared to the 20 available classification labels, we did not calculate the precision, recall, and F1 score for each individual label. In contrast, we calculate their weighted average.

4.4 Exp4: Evaluation of GUIClip for Sketch-to-GUI Retrieval

Instead of searching for a textual query, it is also possible to perform the GUI search using a UI sketch. Based on GUIClip, we built a Dual GUIClip ViT model for sketch-to-GUI retrieval. An evaluation on the Swire dataset [29] shows that our model outperform Swire and CLIP on sketch-to-GUI retrieval, with a recall@10 of 0.685.

4.4.1 Model Architecture. In order to adapt GUIClip for sketch-to-GUI retrieval, we employed its image encoder. The left side of Figure 8 illustrates the architecture of the model, which encompasses a screenshot encoder and a sketch encoder. Both of these components are initialised with the image encoder of GUIClip. During the training phase, the weights of the screenshot encoder are held constant, whilst training is solely conducted on the sketch encoder. During the inference phase, which is illustrated at the right side of the Figure 8, both encoders are frozen. The embedding of the query sketch is compared with the embeddings of each screenshot. The top-k screenshots whose embedding is the most similar with the sketch embedding are retrieved.

4.4.2 Dataset. The dataset employed in this study was procured from Swire [29], which encompasses 3802 sketches corresponding to 2201 screenshots from 167 apps within the Rico dataset. This dataset was compiled by four designers, who contributed to sketching 505, 1017, 1272, and 1008 screenshots respectively. However, it was discovered that certain images from this dataset were not present within the Rico dataset. Hence after appropriate cleaning, a refined dataset featuring a total of 3551 sketches was obtained. The split between the training and test sets within Swire was not explicitly delineated in their work. Consequently, for the purposes of our study, we adopted 486 sketches from one designer as the test set, while the sketches from three other designers constituted the training set. This ensured that the model learns to generalise across sketches developed by various designers.

4.4.3 Baselines. We compared GUIClip's performance in sketch-to-GUI retrieval with the following baselines:

- *Swire*. Swire establishes itself as the pioneer sketch-to-GUI retrieval model [29]. This model comprises two distinct VGG-A networks [63] that independently compute the embeddings for screenshots and sketches. The retrieval process is subsequently conducted by measuring the similarity between these embeddings. As the implementation of Swire is not publicly accessible and as we were unable to get it from the authors, we decided to recreate it according to the details in the original paper.
- *Dual CLIP ViT*. The architecture of the Dual CLIP ViT is identical to that of GUIClip, except for their respective parameters' weights. The weights of the image encoder of Dual CLIP ViT are derived directly from the CLIP model.

4.4.4 Training Details of Fine-tuning. Under zero-shot setting, the Dual GUIClip ViT and Dual CLIP ViT have not been trained. While under fine-tuning setting, the Dual GUIClip ViT and Dual CLIP ViT models have been trained with contrastive learning on the training set for 10 epochs. We used AdamW optimiser with an initial learning rate $1e^{-5}$, batch size 256.

For the training of Swire, we tried to use the same configurations as specified in the original paper, including a learning rate of 0.01 and a batch size of 32. However, we noticed a significant decline in performance with these settings, wherein the top-10 recall stood at a mere 0.02. To address this, we decided to use a batch size of 64 and a learning rate of $1e^{-5}$. As the original article does not specify the number of training epochs employed, we explored different values, ranging from 10 to 100 epochs in increments of 10. Our experiments indicates that 100 epochs produce the best results.

4.4.5 Evaluation Metric. Similar to the evaluation of text-to-GUI retrieval presented on the Section 4.1.4, we use recall@1, recall@3, recall@5 and recall@10.

5 EVALUATION RESULTS

5.1 RQ1: Performance of GUIClip in Text-to-GUI Retrieval

Table 4 shows the evaluation results for the text-to-GUI retrieval task using GUIClip (last row) and the baseline models on three distinct datasets. Compared to the retrieval by chance, which has a recall@10 of 0.01, the CLIP model proves to be more effective in text-to-GUI retrieval, with a recall@10 exceeding 0.2 across the three datasets. This implies that, given a caption, the CLIP model has a 20 percent probability that the corresponding screenshot lies within the top 10 retrieved screenshots. Nevertheless, the GUIClip model consistently outperforms the CLIP model in almost all metrics and for the three datasets. This indicates that our fine-tuned CLIP model, GUIClip, is more adept at bridging the semantic gap between the caption and the screenshot.

Table 4. Text-to-GUI retrieval performance on test set (Experiment 1).

Dataset	Model	Recall@1	Recall@3	Recall@5	Recall@10	Recall@50	Recall@100
	(Chance)	0.001	0.003	0.005	0.010	0.050	0.100
SCapRepo	OCR + BGE	0.164	0.291	0.355	0.441	0.624	0.705
	CLIP	0.127	0.242	0.299	0.392	0.627	0.723
	GUIClip-CS	0.066	0.132	0.164	0.244	0.479	0.603
	GUIClip	**0.377**	**0.526**	**0.593**	**0.687**	**0.857**	**0.908**
Screen2Words	OCR + BGE	0.130	0.229	0.287	0.361	0.559	0.650
	CLIP	0.097	0.171	0.220	0.298	0.508	0.615
	GUIClip-CS	0.195	0.349	0.429	0.532	0.800	0.887
	GUIClip	**0.216**	**0.371**	**0.449**	**0.566**	**0.825**	**0.903**
Clarity	OCR + BGE	**0.152**	**0.242**	**0.281**	**0.338**	0.500	0.585
	CLIP	0.080	0.138	0.166	0.219	0.373	0.464
	GUIClip-CS	0.073	0.146	0.198	0.278	0.515	0.648
	GUIClip	0.075	0.150	0.207	0.287	**0.533**	**0.667**

GUIClip-CS were trained merely on the Screen2Words and Clarity datasets. Notably, GUIClip delivers better performance than GUIClip-CS on the Screen2Words and Clarity datasets, registering a performance improvement ranging between 0.1 to 0.3 across various metrics. This performance distinction is further accentuated in the SCapRepo dataset, where GUIClip-CS exhibits an inferior performance, even when compared to the CLIP model. This outcome is hypothesised to derive from caption style variances. Specifically, the captions associated with the Screen2Words and Clarity datasets are significantly lengthy and detailed, which may possibly impede GUIClip-CS's capacity to generalise effectively on the SCapRepo dataset. On a holistic level, the results of GUIClip suggest that fine-tuning on our SCapRepo dataset contributes substantially to the performance of CLIP models for the text-to-GUI retrieval task. This dataset enhances the performance of the CLIP model not only on the SCapRepo dataset, but also leads to performance gains on other screenshot-caption datasets.

As for the text-only approach, when analysing the SCapRepo and Screen2Words datasets, it becomes evident that GUIClip surpasses the performance of OCR+BGE. Interestingly, when the comparison is carried out on the Clarity datasets, GUIClip exhibits a deterioration in comparison to text-only approach for recall@1, Recall@3, recall@5 and recall@10. This discrepancy can presumably be attributed to the variability in caption styles. The majority of captions in the SCapRepo dataset are app features such as "Track your health trends" or "Custom colour themes". And the captions found within the Screen2Words dataset generally encompass high-level descriptions of the screenshot, such as "screen showing settings options" and a "display of news articles in a news app". In contrast, the Clarity dataset captions offer substantially more details about the screenshots, like "at the bottom left there is a skip button for users to skip the introduction", and "in the middle of the screen a popup is displayed with a label called set date". These detailed descriptions can easily be matched with the text displayed on the screenshots, thereby improving the performance of text-only approaches. Nevertheless, GUIClip continues to surpass OCR+BGE in both recall@50 and recall@100.

Overall, our findings underscore that GUIClip outperforms the baseline models in text-to-GUI retrieval, particularly when the goal is to search for descriptions of an overall functionality and features implemented in the screens rather than verbose, documentation-like descriptions of the screen's UI elements.

Table 5. Relevance of the retrieved screenshots by three GUI search engines (Experiment 2 / Manual assessments).

Search Engine	MRR	P@k				HIT@k			
		P@1	P@3	P@5	P@10	HIT@1	HIT@3	HIT@5	HIT@10
RaWi	0.410	0.291	0.264	0.254	0.214	0.291	0.487	0.594	0.701
GUing-CS	0.255	0.134	0.138	0.139	0.152	0.134	0.291	0.401	0.615
GUing	**0.510**	**0.372**	**0.352**	**0.339**	**0.343**	**0.372**	**0.666**	**0.773**	**0.914**

5.2 RQ2: Relevance of Search Results

Table 5 shows the results of the manual evaluation for GUing and the baseline search engines. The results show that GUing consistently outperforms RaWi across all evaluation metrics. P@1 and HIT@1 show that there is 0.372 probability that GUing will return a relevant screenshot as the first result, a considerable improvement over RaWi's rate of about 0.291. The P@10 value, indicative of the proportion of relevance results of the top-10 results returned by GUing, stands at 0.343, compared with RaWi's rate of 0.214. The HIT@10 evaluation metric shows that, in more than 91% of cases, GUing can return at least one pertinent screenshot among the top-10 results, far exceeding the percentage for RaWi, which is recorded at 70%. This is particularly interesting for GUI ideation tasks, as it seems reasonable for a designer to check 10 screen suggestions for identifying at least one relevant screen (similar to screening the first result page of a google search). Finally, the MRR metric reveals that the ranking of the first relevant result returned by GUing is superior to that of RaWi.

The results also show that GUing significantly outperforms GUing-CS too. This can be quantified by notable gaps in the following metrics: a gap exceeding 0.2 for P@k, approximately 0.3 for HIT@k, and 0.25 in terms of MRR. The architectures of the search engines GUing and GUing-CS are identical, except the training set for the vision-language models and the repository used for searches. GUing-CS does not incorporate our SCapRepo and ScreenRepo datasets for its model training and search operation. The large gap between GUing and GUing-CS demonstrates the importance of our SCapRepo and ScreenRepo datasets.

To understand the difference between the screenshots obtained using GUing and Rawi, we carried out additional analyses on each search engine with a few queries. For example, the screenshots returned by GUing and RaWi for the query "sleeping track" are shown on Figure 9 and 10. Upon examination, it became apparent that there are considerable stylistic differences between the screenshots returned by the two search engines, mainly due to three factors:

- The shape of the screenshots returned by GUing is less neat. A large portion of screenshots within the ScreenRepo dataset are derived from the *surrounded screenshots*, in which the screenshot area varies in shape across diverse images. However, these screenshots are still useful to inspire GUI design.
- GUing returns a wider variety of screenshots than RaWi. As shown on Figure 10, the presence of empty or settings pages among the screenshots returned by RaWi is thought to be an inherent limitation of the text-only retrieval approach, as they barely offer any contribution to sparking creativity in GUI design. Text-only retrieval approaches compare the similarity of the query with text or metadata found on each screenshot. As setting pages typically contain much text, they will likely be matched to many queries. Empty pages are likely a result of the Rico data collection method, which involves the GUI exploration of apps. When performing crowdsourced or automated exploration, the app often lack initial data, such as recorded sleep events.

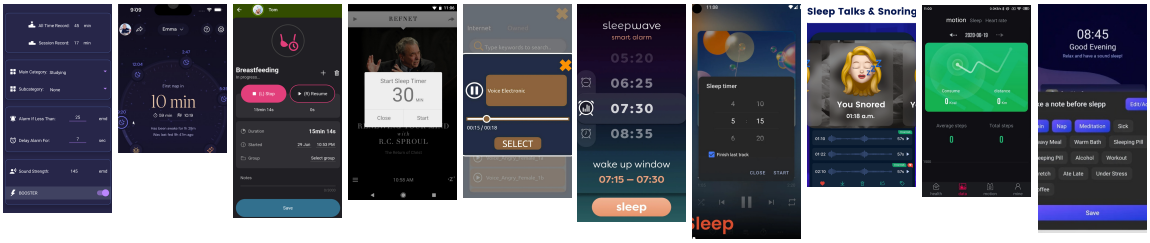


Fig. 9. Top-10 screenshots returned by GUing for the query “sleep tracking”.

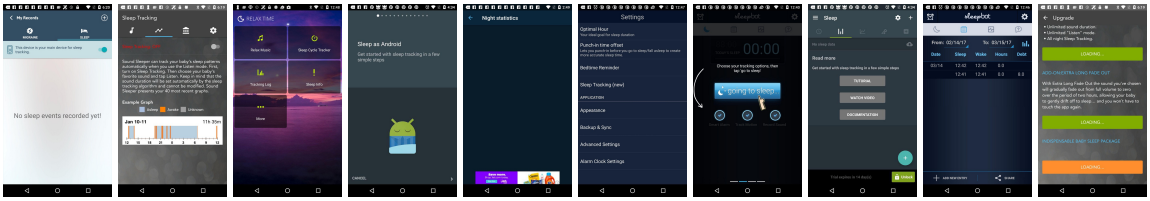


Fig. 10. Top-10 screenshots returned by RaWi for the query “sleep tracking”.

- The modernity of the screenshots returned by GUing vastly exceeds those resulting from RaWi. This discrepancy can be attributed to the collection periods of the underlying datasets. The ScreenRepo dataset was collected recently, whereas the Rico dataset was created in 2017. ScreenRepo can easily be updated by processing introduction images of new apps when added to the Google Play store.

Overall, our findings demonstrate that GUing consistently outperforms the baseline approaches in text-to-GUI retrieval task in term of relevance of the retrieved screens.

5.3 RQ3: Performance of GUIClip in Other GUI-related Tasks

5.3.1 GUI Classification. As shown in Table 6, vision-language models, specifically CLIP and GUIClip, clearly outperform OCR+BGE as expected. This reaffirms the assertion that mere analysis of text displayed on screenshots is insufficient for substantial GUI understanding, making the vision information crucial. Notably, GUIClip achieves superior results compared to CLIP, and records an F1 score that exceeds that of CLIP by 20 percent in both settings. This highlights the advantage of our GUIClip model for GUI classification tasks over the baseline models, given its enhanced knowledge about the GUI domain accumulated during the pre-training phase. Particularly, the Linear-probe led to encouraging precision and recall of over 60% (given the large number of classes in the classification task and the straight application of the model).

Table 6. Classification accuracy on Enrico dataset.

	Zero-shot			Linear-probe		
	Precision	Recall	F1	Precision	Recall	F1
OCR + BGE	0.210	0.094	0.081	0.038	0.186	0.059
CLIP	0.317	0.165	0.138	0.402	0.466	0.395
GUIClip	**0.415**	**0.347**	**0.334**	**0.613**	**0.640**	**0.600**

The efficacy of vision-language models, as measured under the linear-probe setting, surpasses that determined under the zero-shot setting. This superior performance can be explained by the additional knowledge gained during the training phase from the Enrico dataset. However, a contrary pattern was found concerning OCR+BGE, which showed diminished performance following linear-probe training. We hypothesise that this is because the text-only approach is inadequate to extract visual information from the screenshots, which makes the training stage ineffective.

5.3.2 Sketch-to-GUI Retrieval. Table 7 presents the evaluation outcomes for GUIClip and the baseline models on the Swire dataset within both zero-shot and fine-tuning settings. Our observation revealed that even under zero-shot setting, the Dual GUIClip ViT has a recall@10 score of 0.224, which indicates its potential for sketch-to-GUI retrieval. In contrast, the Dual CLIP ViT was not as effective, as evidenced by its low recall@10 score of 0.053.

Table 7. Retrieval accuracy on the test set of Swire dataset.

	Zero-shot				Fine-tuned			
	Recall@1	Recall@3	Recall@5	Recall@10	Recall@1	Recall@3	Recall@5	Recall@10
Swire	-	-	-	-	0.027	0.062	0.117	0.214
Dual CLIP ViT	0.019	0.029	0.037	0.053	0.177	0.296	0.422	0.533
Dual GUIClip ViT	**0.062**	**0.117**	**0.156**	**0.224**	**0.368**	**0.580**	**0.685**	**0.772**

Fine-tuning can significantly improve the models' performance. After fine-tuning, the recall@10 score of the Dual CLIP ViT increased to 0.553, while the Dual GUIClip ViT exhibited a remarkable improvement, reaching 0.772. These results substantiate the superior efficacy of GUIClip in comparison to CLIP for sketch-to-GUI retrieval.

The performance of our Swire implementation, with a recall@1 of 0.027 and a recall@10 of 0.214, is not congruent with the metrics reported in the originating study by Huang et al. [29]. The researchers reported a significantly higher recall values of 0.159 and 0.609, at ranks 1 and 10 respectively. This discrepancy could potentially be attributed to two factors: 1) Our evaluation used 486 pairings for testing, in contrast to the 276 pairings used in the original research. The size of the test set can dramatically influence retrieval performance outcomes; 2) The implementation details could also lead to differing results. It is important to note, however, that the original Swire model's implementation is not available for replication. As reported in Section 4.4, we tried our best to not disadvantage Swire in our evaluation as the original implementation was not accessible. Nonetheless, even if we use the results from Huang et al. [29], our Dual GUIClip ViT model still demonstrates superior performance, with a score of 0.368 for recall@1 and 0.772 for recall@10.

6 RELATED WORK

6.1 Mobile GUI Retrieval

GUI retrieval refers to the search of existing GUI designs (usually images) with queries of various format.

By sketch image: UI sketches are often used at the early stage of the design. They can also be used as a query. As detailed in Section 4.4, Huang et al. introduced Swire [29], which comprises two separate VGG-A networks [63] to compute the embeddings for screenshots and sketches, respectively. The retrieval process is subsequently performed by measuring the similarity of these embeddings. Unlike Swire that treats a sketch image as a unit, Mohian et al. [48, 49] performed object detection on the sketches to identify their UI components, which are then used to find the screenshots in Rico dataset with the best matching types and locations of the UI components.

By wireframe: A wireframe is an image that represents the skeletal layout of a screenshot. Given a wireframe image, one can retrieve the corresponding screenshots. Deka et al. [16] and Liu et al. [40] trained an autoencoder [5] for UI layout similarity, which supports query-by-example search over UIs. Chen et al. [12] performed wireframe-based GUI retrieval by encoding the visual semantics of UI designs using a large database of UI design wireframes.

By screenshot image: Using a screenshot from an app, it is possible to retrieve similar screenshots from a GUI repository. Screen2vec [37] extracts the UI components, layout, and app description to generate a screen embedding. Querying similar screenshots is then done by comparing the similarity of screen embeddings. VINS [8] creates screen embeddings by combining both image embeddings and UI components embeddings. Instead of querying screenshots, GUIFetch [4] searches for apps in public repositories that include similar screens and transitions between them.

By text: Using textual queries for GUI retrieval is practical as often only a textual description or keywords are available at the search time. GUIGLE [6] has indexed a collection of GUIs along with their metadata from the ReDraw dataset [52]. GUIGLE leverages various information sources such as text displayed on the screen, names of UI components, and app name to retrieve relevant screens. On the other hand, as presented in Section 4.2, RaWi [31] uses a BERT-based [17] ranking model to retrieve GUIs from the Rico dataset [16]. This retrieval process involves matching the GUI text (text displayed on the screen, GUI activity name, and GUI component identifier) with the textual query in order to identify the most relevant images. Unlike the preceding studies that focus on retrieving entire screenshot, Gallery D.C. [10, 22] allows users to search UI components, like a button or a checkbox. It provides a GUI gallery that allows users to search UI components with different filters, including component type, size, colour, app category, and displayed text.

All GUI retrieval approaches suggested so far either use the visual information or the textual information available about the the mobile GUIs for matching the query with corresponding data in the repository. In contract, our approach combines both modalities by training a text-vision model based on screen-caption pairs. This bridges between the visual perception and the natural language and boosts the retrieval performance as our results show.

6.2 Vision Language Models for Mobile GUI

Vision-language models combines both the vision and language modalities. They are trained on image-text pairs and can be applied to a range of tasks [20, 79]. Until this work, the main use case of vision-language models in the mobile GUI domain is the GUI captioning. This refers to the task of generating a high-level summary of a screenshot to describe its contents and functionalities. The Screen2Words model [71], built on the foundations of the Transformer Encoder-Decoder architecture [70], uses multimodal data, inclusive of the screenshot image, view hierarchy, and app description, to yield the screen caption. Contrarily, Spotlight, a vision-only approach employed for multiple tasks including widget captioning and screen captioning, possess architecture rooted in Vision Transformer [18] and the T5 model [60].

While some studies pertaining to GUI-related tasks may appear to employ vision-language models, they do not in reality. For instance, XUI [33] model does not fully address the language modality. It generates the screen caption with template-based natural language generation (NLG) engine. As discussed above, GUIGLE [6] and RaWi [31] introduced text-based GUI retrieval approaches by using the textual content exhibited in the screenshots or the metadata, instead of employing the vision information for the search. This may result in the omission of crucial vision information of the UI screen. To the best of our knowledge, no existing research employs vision-language models for text-to-GUI retrieval.

Recently, two related vision-language models, UIClip [75] and Ferret-UI [77], have been presented in parallel to our work. UIClip, like our model GUIClip, is a fine-tuned version the CLIP model. However, their purposes differ significantly. While GUIClip is designed for GUI retrieval (as app vendors describe what is implemented on the corresponding screen),

UIClip is trained to assess the design quality and visual relevance of a UI. Ferret-UI, on the other hand, is a multimodal foundation model that goes beyond vision-language models by supporting more input formats, such as points, boxes, and scribbles on screenshots. This model is capable of performing various tasks (e.g., widget classification, icon recognition, OCR) with flexible input formats, as well as grounding tasks (e.g., finding widgets, icons, or text, and listing widgets on a screen).

6.3 Mobile GUI Datasets

Large-scale mobile GUI datasets comprise a vast collection of screenshots and serve as a valuable resource for GUI retrieval. Rico [16, 40] is one of the largest datasets in the literature. Deka et al. [16] combined crowdsourcing and automation to mine design and interaction data from Android apps at runtime. Their dataset exposes visual, textual, structural, and interactive design properties of more than 66k unique UI screens appx. 9.3k Android apps. Several other datasets [32, 68, 78] are built on Rico including the Screen2Words dataset [71] used in this paper. As Rico was created seven years ago, some of its UIs might be outdated for contemporary apps as our evaluation suggests.

Chen et al. [12] developed an extensive dataset that comprises almost 55k screenshots from 7,748 Android apps. The screenshots were obtained through automated GUI exploration [11]. Moran et al. [52] collected the ReDraw dataset in a fully automated manner by mining and executing the top-250 Android apps in each category on Google Play, excluding games. This yielded a total of 14,382 unique screenshots and 191k labelled GUI components. Clarity [54] is an annotated subset of ReDraw consisting of 45,998 descriptions for 10,204 screenshots of popular apps. The descriptions were created by crowd workers using Amazon Mechanical Turk. Instead of focusing on entire screens, Li et al. [38] released a dataset containing appx. 162,859 phrases created by human workers for annotating 61k UI components across 21,750 unique screens.

Generally, more and more crowdsourced or automated GUI exploration methods are used to (semi)automatically explore UIs of iOS apps [74] or Android apps [14, 15, 80]. While certainly efficient and scalable, automated UI exploration rather targets UI testing than GUI design inspiration. GUI datasets gathered through crowdsourced or automated exploration may omit important app features since the access to certain UIs may necessitate app authorisation or initial configurations [14]. In addition, these datasets are not curated by actual app designers and vendors. Recent studies showed that app vendors and app users (or the crowd) often use different vocabularies to describe app features [25, 26]. The dataset introduced in our work ScreenRepo and SCapRepo aims at closing this gap. While we also used automated techniques for the data collection and cleaning, the screen-caption pairs in SCapRep are curated by various app vendors, describing what a screen is supposed to do and what problem for the user does it solve.

7 DISCUSSION

We discuss the implications of our work for research and practice together with potential threats to validity.

7.1 Implications for Software Practitioners

Our work has several potential implications for software practitioners, particularly for app designers and requirements engineers. Our search engine primarily enables app designers and developers to get inspirations on how their screens can look like, for instance in an early project phase, when only a list of app features is available [73]. Particularly in small development teams, which often lack resources and expertise for GUI design and usability engineering [7], this can provide a substantial support. Despite the general trend with Generative AI for more code generation and automation, we think that it remains crucial in professional app development to have the designer (or developer) in the

loop [72]. GUing is such a tool that uses AI to speed-up design while ensuring the crucial role of human creativity and oversight.

GUing can also serve as a rapid prototyping tool. It can, e.g., be used during requirements interviews and in workshops, to retrieve screenshots based on identified requirements, speeding up their refinement and validation. This by itself, increases stakeholder engagement and helps mitigate development risks. Additionally, retrieved screenshots serve as a visual aid for illustrating and documenting the requirements and reducing potential ambiguity [51].

Given the significant impact of GUI on user experience and retention [13], and as GUI designs and user preferences evolve rapidly, it is essential for app designers to continuously observe and learn from trends and best practices. GUing leverages app introduction images curated by actual app vendors and experts. Our pipeline for image processing and model training allows for a seamless addition of new images to the GUing repository, ensuring a high quality and up-to-dateness. This helps designers learn the latest UI design trends. When augmented with download, rating, and feedback trends, retrieved screens can help not only get inspiration but also estimate user trends [46].

7.2 Implication for Software Researchers

There has been extensive research on app store mining over the last decade, with the majority of studies focusing on app reviews and descriptions [1, 43, 47]. Less work have delved into the analysis of other developer information shared in the stores, such as app introduction images [10, 22]. Our work highlights the potential of this information, particularly as it is curated, usually exhibits a high quality (as created by professionals), and captures various modalities (image, text, videos, binaries, etc.). We think that researchers should focus more on investigating this information and combining it with crowd-data for building multimodal foundation models for software engineering.

From the app introduction images, we developed and shared the SCapRepo dataset and subsequently the vision-language model GUIClip. The evaluation results confirms the power for these models to learn and combine modalities, not only for screenshot retrieval but also at least for GUI classification and sketch-to-GUI retrieval. We think that GUIClip and similar foundation models tuned on software design data have the potential to be adapted for other GUI-related tasks, including GUI captioning, GUI generation, or GUI testing. As a foundational model, CLIP model itself is the basis to numerous models [79], such as the image captioning model ClipCap [50] and the image generation model Stable Diffusion [62]. By replacing the CLIP model with tuned, domain specific models such as GUIClip, it is possible to enhance these derivative models' capabilities for specific tasks. Researchers should investigate this in details, not only with regard to model performance (or accuracy) but also with regard to quality of output, risk of prediction faults, as well as design and engineering workflows to be adapted and integrated [42, 72].

7.3 Threats to Validity

This section discusses potential threats to the validity of our work.

7.3.1 Internal and Construct Validly. As for every study that includes manual evaluation, the process of evaluating the usefulness of the search results may have included inherent biases. Particularly, the evaluation outcomes may have been influenced by the evaluators' preferences towards specific search engines. In an attempt to mitigate this potential observer bias, we incorporated a number of steps. We created an evaluation tool which mixes and shuffles the resulting screenshots originating from the three distinct search engines. As a result, the evaluator cannot discern the origin of a specific screenshot, thereby curbing potential bias among diverse search engines. Moreover, the evaluation process was carried out by four evaluators, each with a minimum of five years' experience in software development. They

conducted a careful evaluation based on a evaluation guide (e.g. stating what is a relevant GUI and what is not) and using a uniform evaluation tool. Finally, each query was independently assessed by at least two evaluators to minimise potential mistakes.

7.3.2 External Validity. We discuss the threats to external validity for the evaluation of GUing and GUIClip.

GUing: Query Selection for Search Engine Evaluation. The selection of queries is crucial for the evaluation, since the search performance may vary depending on the query. For example, a GUI search engine that exhibits satisfactory results for the health and fitness domain may not necessarily produce the same outcome for the finance or education domains. It is thus imperative to consider varying domains and apps in the evaluation. The study of Kolthoff et al. [31] provides a gold standard of queries and corresponding GUIs. However, this dataset has already been exhausted for the training of RaWi, which makes it unsuitable for our comparative study. Given the lack of alternative query datasets, we searched for articles published by companies specialised in mobile app development. This decision could potentially introduce a selection bias. The app features extracted from the articles span ten distinct domains [69], plus innovative features from multiple additional domains [30]. Therefore, we think that these sets are representative enough for the purpose of our evaluation. Nevertheless, reproducing our study with additional queries, scenarios, and evaluators would certainly further strengthen the generalisability of its results.

GUIClip: Bias of Datasets for Model Evaluation. Enrico contains 1460 screenshots for 20 categories, while Swire contains 3551 sketch-screenshot pairs. Compared with the large scale of SCapRepo, these datasets used to evaluate GUIClip for other GUI-related tasks are relatively small. This may introduce potential bias into evaluation. To address this, we manually reviewed these two datasets and found that they encompass diverse domains. This suggests that GUIClip is effective for handling various GUIs across multiple domains. However, to strengthen the external validity of the findings, further evaluation on additional datasets is necessary, ideally with a broader range of domains, apps, and designs.

8 CONCLUSION AND FUTURE WORK

In this paper, we presented GUing, a GUI search engine, built upon GUIClip, a novel vision-language foundation model for the domain of mobile apps. The GUIClip model was trained using SCapRepo, a comprehensive dataset comprised of 135k screenshot-caption pairs created by actual app experts. In addition, we presented ScreenRepo, a dataset that serves as a large repository for GUing. SCapRepo and ScreenRepo were created through an automated pipeline, enabling an easy update, e.g. when new screen designs emerge.

Our evaluation, including a benchmarking on various datasets and a manual assessment of search results, indicates that GUing outperforms state-of-the-art approaches for GUI retrieval. Moreover, the evaluation highlights the critical role that the SCapRepo and ScreenRepo datasets play in achieving the top performance of our vision-language model: not only for GUI retrieval but also for GUI classification and sketch-to-GUI retrieval in the mobile app domain. From here, there are several follow-up future directions:

- *Increasing and diversifying the screenshot dataset.* The performance of a vision-language model is correlated with the size of its training data. According to Statista [67], Google Play includes approximately 2.43 million apps as of 2023. Our training dataset includes merely 117k apps. Mining additional app introduction images will thus likely improve the performance of GUing and relevance of the results. Moreover, other platforms like Apple iOS, Samsung Galaxy Watch, or Microsoft HoloLens also have stores with similar app introduction images. Learning and searching those screenshots can improve the engine and inspire the design of corresponding apps.

- *Hybrid search methods.* The inherent limitations of natural language can pose difficulties in precisely describing the desired GUIs through text. To enhance the retrieval process, it might be beneficial to support a more diverse range of search methods and data. For instance, sketch-to-GUI retrieval would enable users to find GUI designs (or related follow up screens). Additionally, GUI-to-GUI retrieval would enable users to retrieve other GUIs that exhibit a resemblance (or other dependencies) to the original interfaces.
- *Evaluations with practitioners and integration into the development workflows.* Although our empirical evaluation show the efficacy of GUing, the reality of software development can be more complex and nuanced. Thus, additional empirical studies with app developers, UI designers, and requirements engineers would likely lead to comprehensive feedback on real usage, scalability, further search parameters, as well as how GUing and GUIClip can be ideally integrated into existing app development workflows and tools [42].

REFERENCES

[1] Afnan A. Al-Subaihin, Federica Sarro, Sue Black, Licia Capra, and Mark Harman. 2021. App Store Effects on Software Engineering Practices. *IEEE Transactions on Software Engineering* 47, 2 (2021), 300–319. https://doi.org/10.1109/TSE.2019.2891715

[2] AppBrain. 2024. Google Play Ranking in the United States (Oct 2023). https://www.appbrain.com/stats/google-play-rankings. Accessed: 2023-10-01.

[3] Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu. 1998. An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. *J. ACM* 45, 6 (1998), 891–923. https://doi.org/10.1145/293347.293348

[4] Farnaz Behrang, Steven P. Reiss, and Alessandro Orso. 2018. GUIfetch: Supporting app design and development through GUI search. *Proceedings - International Conference on Software Engineering* (2018), 236–246. https://doi.org/10.1145/3197231.3197244

[5] Yoshua Bengio. 2009. *Learning deep architectures for AI.* Vol. 2. 1–27 pages. https://doi.org/10.1561/2200000006

[6] Carlos Bernal-Cardenas, Kevin Moran, Michele Tufano, Zichang Liu, Linyong Nan, Zhehan Shi, and Denys Poshyvanyk. 2019. Guigle: A GUI search engine for android apps. *Proceedings - 2019 IEEE/ACM 41st International Conference on Software Engineering: Companion, ICSE-Companion 2019* (2019), 71–74. https://doi.org/10.1109/ICSE-Companion.2019.00041 arXiv:1901.00891

[7] Nis Bornoe and Jan Stage. 2013. Supporting usability engineering in small software development organizations. In *Proceedings of the The 36th Information Systems Research Conference in Scandinavia (IRIS 36).* 1–12.

[8] Sara Bunian, Kai Li, Chaima Jemmali, Casper Harteveld, Yun Fu, and Magy Seif Seif El-Nasr. 2021. VINS: Visual Search for Mobile User Interface Design. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems (CHI '21).* Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3411764.3445762

[9] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. 2020. End-to-End Object Detection with Transformers. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 12346 LNCS (2020), 213–229. https://doi.org/10.1007/978-3-030-58452-8_13 arXiv:2005.12872

[10] Chunyang Chen, Sidong Feng, Zhenchang Xing, Linda Liu, Shengdong Zhao, and Jinshui Wang. 2019. Gallery D.C.: Design search and knowledge discovery through auto-created GUI component gallery. *Proceedings of the ACM on Human-Computer Interaction* 3, CSCW (2019). https://doi.org/10.1145/3359282

[11] Chunyang Chen, Ting Su, Guozhu Meng, Zhenchang Xing, and Yang Liu. 2018. From UI design image to GUI skeleton: A neural machine translator to bootstrap mobile GUI implementation. In *Proceedings - International Conference on Software Engineering*, Vol. 6. 665–676. https://doi.org/10.1145/3180155.3180240

[12] Jieshan Chen, Chunyang Chen, Zhenchang Xing, Xin Xia, Liming Zhu, John Grundy, and Jinshui Wang. 2020. Wireframe-based UI Design Search through Image Autoencoder. *ACM Transactions on Software Engineering and Methodology* 29, 3 (2020). https://doi.org/10.1145/3391613 arXiv:2103.07085

[13] Qiuyuan Chen, Chunyang Chen, Safwat Hassan, Zhengchang Xing, Xin Xia, and Ahmed E. Hassan. 2021. How Should I Improve the UI of My App?: A Study of User Reviews of Popular Apps in the Google Play. *ACM Transactions on Software Engineering and Methodology* 30, 3 (2021), 1–37. https://doi.org/10.1145/3447808

[14] Sen Chen, Lingling Fan, Chunyang Chen, and Yang Liu. 2023. Automatically Distilling Storyboard With Rich Features for Android Apps. *IEEE Transactions on Software Engineering* 49, 2 (2023), 667–683. https://doi.org/10.1109/TSE.2022.3159548 arXiv:2203.06420

[15] Sen Chen, Lingling Fan, Chunyang Chen, Ting Su, Wenhe Li, Yang Liu, and Lihua Xu. 2019. StoryDroid: Automated Generation of Storyboard for Android Apps. In *Proceedings - International Conference on Software Engineering*, Vol. 2019-May. IEEE, 596–607. https://doi.org/10.1109/ICSE.2019.00070 arXiv:1902.00476

[16] Biplab Deka, Zifeng Huang, Chad Franzen, Joshua Hibschman, Daniel Afergan, Yang Li, Jeffrey Nichols, and Ranjitha Kumar. 2017. Rico: A mobile app dataset for building data-driven design applications. *UIST 2017 - Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology* (2017), 845–854. https://doi.org/10.1145/3126594.3126651

[17] Jacob Devlin, Ming Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL HLT 2019 - 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference*, Vol. 1. 4171–4186. arXiv:1810.04805 https://arxiv.org/abs/1810.04805

[18] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An Image Is Worth 16X16 Words: Transformers for Image Recognition At Scale. In *ICLR 2021 - 9th International Conference on Learning Representations*. arXiv:2010.11929

[19] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2024. The Faiss library. (2024). arXiv:2401.08281 [cs.LG]

[20] Yifan Du, Zikang Liu, Junyi Li, and Wayne Xin Zhao. 2022. A Survey of Vision-Language Pre-Trained Models. *IJCAI International Joint Conference on Artificial Intelligence* (2022), 5436–5443. https://doi.org/10.24963/ijcai.2022/762 arXiv:2202.10936

[21] Explosion. 2024. Prodigy - An annotation tool for AI, Machine Learning and NLP. https://prodi.gy/. Accessed: 2024-3-10.

[22] Sidong Feng, Chunyang Chen, and Zhenchang Xing. 2022. Gallery D.C.: Auto-created GUI Component Gallery for Design Search and Knowledge Discovery. In *Proceedings - International Conference on Software Engineering*, Vol. 1. Association for Computing Machinery, 80–84. https://doi.org/10.1109/ICSE-Companion55297.2022.9793764 arXiv:2204.06700

[23] Alessio Ferrari and Paola Spoletini. 2023. Strategies, Benefits and Challenges of App Store-inspired Requirements Elicitation. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 1290–1302. https://doi.org/10.1109/ICSE48619.2023.00114

[24] Google. 2024. Add preview assets to showcase your app - Play Console Help. https://support.google.com/googleplay/android-developer/answer/9866151?hl=en&sjid=20643806677574592 5-EU. Accessed: 2024-3-10.

[25] Marlo Haering, Muneera Bano, Didar Zowghi, Matthew Kearney, and Walid Maalej. 2021. Automating the Evaluation of Education Apps with App Store Data. *IEEE Transactions on Learning Technologies* 14, 1 (2021), 16–27. https://doi.org/10.1109/TLT.2021.3055121

[26] Marlo Haering, Christoph Stanik, and Walid Maalej. 2021. Automatically Matching Bug Reports With Related App Reviews. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. 970–981. https://doi.org/10.1109/ICSE43902.2021.00092

[27] Xu Han, Zhengyan Zhang, Ning Ding, Yuxian Gu, Xiao Liu, Yuqi Huo, Jiezhong Qiu, Yuan Yao, Ao Zhang, Liang Zhang, Wentao Han, Minlie Huang, Qin Jin, Yanyan Lan, Yang Liu, Zhiyuan Liu, Zhiwu Lu, Xipeng Qiu, Ruihua Song, Jie Tang, Ji Rong Wen, Jinhui Yuan, Wayne Xin Zhao, and Jun Zhu. 2021. Pre-trained models: Past, present and future. *AI Open* 2, August 2021 (2021), 225–250. https://doi.org/10.1016/j.aiopen.2021.08.002

[28] Safwat Hassan, Cor Paul Bezemer, and Ahmed E. Hassan. 2020. Studying Bad Updates of Top Free-to-Download Apps in the Google Play Store. *IEEE Transactions on Software Engineering* 46, 7 (2020), 773–793. https://doi.org/10.1109/TSE.2018.2869395

[29] Forrest Huang, John F. Canny, and Jefrey Nichols. 2019. Swire: Sketch-based User Interface Retrieval. *Conference on Human Factors in Computing Systems - Proceedings* (2019), 1–10. https://doi.org/10.1145/3290605.3300334

[30] Nic Hughart. 2023. 50 Best App Ideas For 2024. https://buildfire.com/best-app-ideas. Accessed: 2024-3-10.

[31] Kristian Kolthoff, Christian Bartelt, and Simone Paolo Ponzetto. 2023. Data-driven prototyping via natural-language-based GUI retrieval. *Automated Software Engineering* 30, 1 (2023), 13. https://doi.org/10.1007/s10515-023-00377-x

[32] Luis A. Leiva, Asutosh Hota, and Antti Oulasvirta. 2020. Enrico: A Dataset for Topic Modeling of Mobile UI Designs. *Extended Abstracts - 22nd International Conference on Human-Computer Interaction with Mobile Devices and Services: Expanding the Horizon of Mobile Interaction, MobileHCI 2020* (2020). https://doi.org/10.1145/3406324.3410710

[33] Luis A. Leiva, Asutosh Hota, and Antti Oulasvirta. 2023. Describing UI Screenshots in Natural Language. *ACM Transactions on Intelligent Systems and Technology* 14, 1 (2023), 1–28. https://doi.org/10.1145/3564702

[34] Chenxia Li, Weiwei Liu, Ruoyu Guo, Xiaoting Yin, Kaitao Jiang, Yongkun Du, Yuning Du, Lingfeng Zhu, Baohua Lai, Xiaoguang Hu, Dianhai Yu, and Yanjun Ma. 2022. PP-OCRv3: More Attempts for the Improvement of Ultra Lightweight OCR System. (2022). arXiv:2206.03001 http://arxiv.org/abs/2206.03001

[35] Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. 2023. BLIP-2: Bootstrapping Language-Image Pre-training with Frozen Image Encoders and Large Language Models. (2023). arXiv:2301.12597 http://arxiv.org/abs/2301.12597

[36] Junnan Li, Dongxu Li, Caiming Xiong, and Steven Hoi. 2022. BLIP: Bootstrapping Language-Image Pre-training for Unified Vision-Language Understanding and Generation. *Proceedings of Machine Learning Research* 162, 2 (2022), 12888–12900. arXiv:2201.12086

[37] Toby Jia Jun Li, Lindsay Popowski, Tom M. Mitchell, and Brad A. Myers. 2021. Screen2vec: Semantic embedding of GUI screens and GUI components. *Conference on Human Factors in Computing Systems - Proceedings* (2021). https://doi.org/10.1145/3411764.3445049 arXiv:2101.11103

[38] Yang Li, Gang Li, Luheng He, Jingjie Zheng, Hong Li, and Zhiwei Guan. 2020. Widget captioning: Generating natural language description for mobile user interface elements. *EMNLP 2020 - 2020 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference* 2015 (2020), 5495–5510. https://doi.org/10.18653/v1/2020.emnlp-main.443 arXiv:2010.04295

[39] Tsung Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. 2014. Microsoft COCO: Common objects in context. In *Computer Vision–ECCV 2014: 13th European Conference*. https://doi.org/10.1007/978-3-319-10602-1_48 arXiv:1405.0312

[40] Thomas F. Liu, Mark Craft, Jason Situ, Ersin Yumer, Radomir Mech, and Ranjitha Kumar. 2018. Learning design semantics for mobile apps. *UIST 2018 - Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology* (2018), 569–579. https://doi.org/10.1145/3242587.3242650

[41] Ilya Loshchilov and Frank Hutter. 2019. Decoupled Weight Decay Regularization. https://doi.org/10.48550/arXiv.1711.05101 arXiv:1711.05101 [cs, math].

[42] Walid Maalej. 2009. Task-First or Context-First? Tool Integration Revisited. In *2009 IEEE/ACM International Conference on Automated Software Engineering*. 344–355. https://doi.org/10.1109/ASE.2009.36 ISSN: 1938-4300.

[43] Walid Maalej, Volodymyr Biryuk, Jialiang Wei, and Fabian Panse. 2024. On the Automated Processing of User Feedback. https://doi.org/10.48550/arXiv.2407.15519 arXiv:2407.15519 [cs].

[44] Walid Maalej, Hans-Jörg Happel, and Asarnusch Rashid. 2009. When users become collaborators: towards continuous and context-aware user input. In *Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications (OOPSLA '09)*. Association for Computing Machinery, New York, NY, USA, 981–990. https://doi.org/10.1145/1639950.1640068

[45] Daniel Martens and Walid Maalej. 2019. Extracting and Analyzing Context Information in User-Support Conversations on Twitter. In *2019 IEEE 27th International Requirements Engineering Conference (RE)*. 131–141. https://doi.org/10.1109/RE.2019.00024

[46] Daniel Martens and Walid Maalej. 2019. Release Early, Release Often, and Watch Your Users' Emotions: Lessons From Emotional Patterns. *IEEE Software* 36, 5 (Sept. 2019), 32–37. https://doi.org/10.1109/MS.2019.2923603 Conference Name: IEEE Software.

[47] William Martin, Federica Sarro, Yue Jia, Yuanyuan Zhang, and Mark Harman. 2017. A Survey of App Store Analysis for Software Engineering. *IEEE Transactions on Software Engineering* 43, 9 (2017), 817–847. https://doi.org/10.1109/TSE.2016.2630689

[48] Soumik Mohian and Christoph Csallner. 2022. PSDoodle: Fast App Screen Search via Partial Screen Doodle. *Proceedings - 9th IEEE/ACM International Conference on Mobile Software Engineering and Systems, MOBILESoft 2022* January (2022), 89–99. https://doi.org/10.1145/3524613.3527816

[49] Soumik Mohian and Christoph Csallner. 2023. Searching Mobile App Screens via Text + Doodle. (2023). arXiv:2305.06165 http://arxiv.org/abs/2305.06165

[50] Ron Mokady, Amir Hertz, and Amit H Bermano. 2021. ClipCap : CLIP Prefix for Image Captioning. (2021). arXiv:2111.09734 https://arxiv.org/abs/2111.09734

[51] Lloyd Montgomery, Davide Fucci, Abir Bouraffa, Lisa Scholz, and Walid Maalej. 2022. Empirical research on requirements quality: a systematic mapping study. *Requirements Engineering* 27, 2 (June 2022), 183–209. https://doi.org/10.1007/s00766-021-00367-z

[52] Kevin Moran, Carlos Bernal-Cardenas, Michael Curcio, Richard Bonett, and Denys Poshyvanyk. 2020. Machine Learning-Based Prototyping of Graphical User Interfaces for Mobile Apps. *IEEE Transactions on Software Engineering* 46, 2 (2020), 196–221. https://doi.org/10.1109/TSE.2018.2844788 arXiv:1802.02312

[53] Kevin Moran, Boyang Li, Carlos Bernal-Cárdenas, Dan Jelf, and Denys Poshyvanyk. 2018. Automated reporting of GUI design violations for mobile apps. In *40th International Conference on Software Engineering*. 165–175. https://doi.org/10.1145/3180155.3180246 arXiv:1802.04732

[54] Kevin Moran, Ali Yachnes, George Purnell, Junayed Mahmud, Michele Tufano, Carlos Bernal Cardenas, Denys Poshyvanyk, and Zach H'Doubler. 2022. An Empirical Investigation into the Use of Image Captioning for Automated Software Documentation. *Proceedings - 2022 IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2022* (2022), 514–525. https://doi.org/10.1109/SANER53432.2022.00069 arXiv:2301.01224

[55] Facundo Olano. 2015. Google Play Scraper. https://github.com/facundoolano/google-play-scraper. Accessed: 2024-3-10.

[56] OpenAI. 2021. CLIP. https://github.com/openai/CLIP/blob/main/clip/clip.py. Accessed: 2024-3-10.

[57] OpenAI. 2021. Model Card openai/clip-vit-base-patch32 on HuggingFace. https://huggingface.co/openai/clip-vit-base-patch32. Accessed: 2024-3-10.

[58] Yen Dieu Pham, Davide Fucci, and Walid Maalej. 2018. A first implementation of a design thinking workshop during a mobile app development course project. In *Proceedings of the 2nd International Workshop on Software Engineering Education for Millennials (SEEM '18)*. Association for Computing Machinery, New York, NY, USA, 56–63. https://doi.org/10.1145/3194779.3194785

[59] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. 2021. Learning Transferable Visual Models From Natural Language Supervision. In *Proceedings of the 38th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 139)*, Marina Meila and Tong Zhang (Eds.). PMLR, 8748–8763. arXiv:2103.00020 http://arxiv.org/abs/2103.00020https://proceedings.mlr.press/v139/radford21a.html

[60] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research* 21 (2020), 1–67. arXiv:1910.10683

[61] Tobias Roehm, Nigar Gurbanova, Bernd Bruegge, Christophe Joubert, and Walid Maalej. 2013. Monitoring user interactions for supporting failure reproduction. In *2013 21st International Conference on Program Comprehension (ICPC)*. 73–82. https://doi.org/10.1109/ICPC.2013.6613835 ISSN: 1092-8138.

[62] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Bjorn Ommer. 2022. High-Resolution Image Synthesis with Latent Diffusion Models. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* 2022-June (2022), 10674–10685. https://doi.org/10.1109/CVPR52688.2022.01042 arXiv:2112.10752

[63] Karen Simonyan and Andrew Zisserman. 2015. Very deep convolutional networks for large-scale image recognition. In *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*. 1–14. arXiv:1409.1556

[64] Amanpreet Singh, Ronghang Hu, Vedanuj Goswami, Guillaume Couairon, Wojciech Galuba, Marcus Rohrbach, and Douwe Kiela. 2022. FLAVA: A Foundational Language And Vision Alignment Model. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* 2022-June (2022), 15617–15629. https://doi.org/10.1109/CVPR52688.2022.01519 arXiv:2112.04482

[65] Filip Sondej. 2020. Autocorrect. https://github.com/filyp/autocorrect. Accessed: 2024-3-10.

[66] Peter M. Stahl. 2022. Lingua. https://github.com/pemistahl/lingua-py. Accessed: 2024-3-10.

[67] Statista. 2024. Number of available applications in the Google Play Store from December 2009 to December 2023. https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/. Accessed: 2024-5-10.

[68] Srinivas Sunkara, Maria Wang, Lijuan Liu, Gilles Baechler, Yu Chung Hsiao, Jindong Chen, Abhanshu Sharma, and James Stout. 2022. Towards Better Semantic Understanding of Mobile Interfaces. *Proceedings - International Conference on Computational Linguistics, COLING* 29, 1 (2022), 5636–5650. arXiv:2210.02663

[69] Vladimir Terekhov. 2023. 138 Features to Consider While Developing a Mobile App. https://attractgroup.com/blog/most-comprehensive-list-of-mobile-app-features-while-developing-a-mobile-application. Accessed: 2024-3-10.

[70] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in Neural Information Processing Systems* 2017-Decem, Nips (2017), 5999–6009. arXiv:1706.03762

[71] Bryan Wang, Gang Li, Xin Zhou, Zhourong Chen, Tovi Grossman, and Yang Li. 2021. Screen2Words: Automatic Mobile UI Summarization with Multimodal Learning. In *UIST 2021 - Proceedings of the 34th Annual ACM Symposium on User Interface Software and Technology*. 498–510. https://doi.org/10.1145/3472749.3474765 arXiv:2108.03353

[72] Jialiang Wei, Anne-Lise Courbis, Thomas Lambolais, Gérard Dray, and Walid Maalej. 2024. On AI-Inspired UI-Design. http://arxiv.org/abs/2406.13631 arXiv:2406.13631 [cs].

[73] Jialiang Wei, Anne-Lise Courbis, Thomas Lambolais, Binbin Xu, Pierre Louis Bernard, Gérard Dray, and Walid Maalej. 2024. Getting Inspiration for Feature Elicitation: App Store- vs. LLM-based Approach. https://doi.org/10.48550/arXiv.2408.17404 arXiv:2408.17404 [cs].

[74] Jason Wu, Rebecca Krosnick, Eldon Schoop, Amanda Swearngin, Jeffrey P. Bigham, and Jeffrey Nichols. 2023. Never-ending Learning of User Interfaces. (2023). https://doi.org/10.1145/3586183.3606824 arXiv:2308.08726

[75] Jason Wu, Yi-Hao Peng, Amanda Li, Amanda Swearngin, Jeffrey P. Bigham, and Jeffrey Nichols. 2024. UIClip: A Data-driven Model for Assessing User Interface Design. https://doi.org/10.48550/arXiv.2404.12500 arXiv:2404.12500 [cs].

[76] Shitao Xiao, Zheng Liu, Peitian Zhang, and Niklas Muennighoff. 2023. C-Pack: Packaged Resources To Advance General Chinese Embedding. (2023). arXiv:2309.07597 http://arxiv.org/abs/2309.07597

[77] Keen You, Haotian Zhang, Eldon Schoop, Floris Weers, Amanda Swearngin, Jeffrey Nichols, Yinfei Yang, and Zhe Gan. 2024. Ferret-UI: Grounded Mobile UI Understanding with Multimodal LLMs. https://doi.org/10.48550/arXiv.2404.05719 arXiv:2404.05719 [cs].

[78] Alaa Zaki and Mohamed Abdallah. 2023. MASC : A Dataset for the Development and Classification of Mobile Applications Screens. (2023), 1–15. https://doi.org/10.21203/rs.3.rs-3786876/v1

[79] Jingyi Zhang, Jiaxing Huang, Sheng Jin, and Shijian Lu. 2024. Vision-Language Models for Vision Tasks: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 46, 8 (Aug. 2024), 5625–5644. https://doi.org/10.1109/TPAMI.2024.3369699 Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence.

[80] Xiangyu Zhang, Lingling Fan, Sen Chen, Yucheng Su, and Boyuan Li. 2023. Scene-Driven Exploration and GUI Modeling for Android Apps. (2023). arXiv:2308.10228 http://arxiv.org/abs/2308.10228