UIClip: A Data-driven Model for Assessing User Interface Design

Jason Wu
jsonwu@cmu.edu
HCI Institute, Carnegie Mellon University
Pittsburgh, PA, USA

Yi-Hao Peng
yihao@cmu.edu
HCI Institute, Carnegie Mellon University
Pittsburgh, PA, USA

Xin Yue Li
xal@cmu.edu
HCI Institute, Carnegie Mellon University
Pittsburgh, PA, USA

Amanda Swearngin
aswearngin@apple.com
Apple
Seattle, WA, USA

Jeffrey P. Bigham
jbigham@apple.com
Apple
Pittsburgh, PA, USA

Jeffrey Nichols
jwnichols@apple.com
Apple
Seattle, WA, USA

ABSTRACT

User interface (UI) design is a difficult yet important task for ensuring the usability, accessibility, and aesthetic qualities of applications. In our paper, we develop a machine-learned model, UIClip, for assessing the design quality and visual relevance of a UI given its screenshot and natural language description. To train UIClip, we used a combination of automated crawling, synthetic augmentation, and human ratings to construct a large-scale dataset of UIs, collated by description and ranked by design quality. Through training on the dataset, UIClip implicitly learns properties of good and bad designs by *i)* assigning a numerical score that represents a UI design's relevance and quality and *ii)* providing design suggestions. In an evaluation that compared the outputs of UIClip and other baselines to UIs rated by 12 human designers, we found that UIClip achieved the highest agreement with ground-truth rankings. Finally, we present three example applications that demonstrate how UIClip can facilitate downstream applications that rely on instantaneous assessment of UI design quality: i) UI code generation, *ii)* UI design tips generation, and *iii)* quality-aware UI example search.

KEYWORDS

UI Modeling; UI Design Assessment; Dataset

ACM Reference Format:
Jason Wu, Yi-Hao Peng, Xin Yue Li, Amanda Swearngin, Jeffrey P. Bigham, and Jeffrey Nichols. 2024. UIClip: A Data-driven Model for Assessing User Interface Design. In *The 37th Annual ACM Symposium on User Interface Software and Technology (UIST '24), October 13–16, 2024, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3654777.3676408

1 INTRODUCTION

What makes a good user interface (UI)? It is hard to comprehensively articulate what separates a good UI design from a bad one, and the task of UI design is challenging even for experts with years of training and practice. Guidelines exist that list some general principles [52, 67], but they are often insufficient or difficult to operationalize, especially for novice designers. Because of this, many application UIs today contain common design problems, which can negatively impact usability, accessibility, and design aesthetics.

The most holistic method of evaluating UIs is *usability testing*, which can uncover UI design flaws, accessibility problems, and software bugs, but it is generally a time-consuming and costly process. Approximate assessments, such as *heuristic evaluation*, rely on experts applying a set of pre-defined principles to rapidly identify potential problems and estimate overall UI quality. However, even these abbreviated strategies can be difficult to employ consistently or in the absence of a knowledgeable expert.

To this end, computational methods have been developed to estimate the quality of UIs, taking into account factors such as visual aesthetics [48], cognitive principles [54], and context [55]. Because of their automated nature, they unlock new opportunities for UI design [70, 73] and evaluation [49]. However, most of these prior computational approaches are limited. Some techniques apply objectives and metrics inspired by cognitive principles [48, 54], such as visual complexity, layout quality, and color harmony to UI designs, but their outputs still require interpretation and cannot, for example, be used to compare the quality of two candidate designs. Other approaches are toolkits that learn user-specific models for generating adaptive interfaces [16–18], and they also cannot be applied to more generalized UI design tasks.

Our paper introduces a novel computational model, *UIClip*, that estimates the design quality of any UI from its screenshot and a textual description. Specifically, UIClip generates a numerical score predicts two aspects of design quality: *i)* the presence of design defects and *ii)* design preferences based on human rankings. UIClip is based on the well-known CLIP vision-language model [58], and it uses a natural language description of the UI coupled with a screenshot to assign a numerical score that estimates design quality. CLIP, by default, is not well-suited for judging UI quality and relevance. Therefore, to train UIClip, we developed a novel technique for synthetically generating a large-scale dataset of UIs ranked by design quality. Our strategy takes existing UIs (e.g., web pages) and intentionally introduces design defects by modifying style and layout attributes. The process created pairs of original and "jittered" interfaces and allowed the models to learn the differentiation between these pairs. We used this method to generate 2.3 million pairs of UIs coupled with their quality-related descriptions. To align our model with real-world design preferences, we collected 1.2K ratings

UIST '24, October 13–16, 2024, Pittsburgh, PA, USA

ACM ISBN 979-8-4007-0628-8/24/10
https://doi.org/10.1145/3654777.3676408

from professional designers on an extra UI set. These ratings were used to refine UIClip and validate the effectiveness of our model.

We benchmarked UIClip with other large vision-language models (LVLM) by evaluating them on a held-out set of UI screens. We assess the models on three tasks, including design quality, improvement suggestions, and design relevance. The results showed that UIClip outperformed all other models in every task, despite being smaller in size. Finally, to demonstrate the utility of UIClip, we present three example applications that use our model to provide different types of computational UI design assistance: *i)* quality-aware UI code generation, *ii)* UI design suggestion generation, and *iii)* quality-aware UI example retrieval.

To summarize, our work makes the following contributions:

(1) A large-scale dataset of UI designs and descriptions comprised of synthetic and human-generated design ratings.
(2) A computational model that scores UI screenshots based on relevance to a textual description and design quality.
(3) Three example applications that demonstrate how UIClip can be used to facilitate downstream applications: *i)* a tool that improves the quality of UI code generated by LLMs, *ii)* a tool that generates design recommendations for a UI screenshot, and *iii)* a UI design search engine.

To facilitate research in this area, we plan to release all the training code, data, and models.

2 RELATED WORK

Our research builds upon existing work in UI design and evaluation by encoding UI design quality into computational models, enabling the models to serve as potential tools for UI design assessment. We review the literature in three relevant areas: UI design tools, UI evaluation, and machine learning-based quality metrics.

2.1 UI Design Tools

Embedding computational capabilities into UI design tools enables machines to computationally assess the design thus empowering designers to ideate, prototype, and iterate their work effectively. Early research like SILK [35] and DENIM [50] introduced quick sketching capabilities, making the design process more agile. Damask [43] refined the creation process with its emphasis on pattern-based design, enhancing UI component reusability. The evolution continued with Smart Templates [51], which provided designers with adaptable frameworks that intelligently adjusted to their needs, simplifying the design process. Sikuli [82] built upon the thread of intelligent design tools by integrating image-based search functionalities, making it easier for designers to find and incorporate UI elements. As the field progressed, tools like Sketchplore [73] and Scout [70] enabled designers to explore a wider array of design alternatives, encouraging creativity. D.note [23] and Swire [27] introduced interactive elements that incorporated user feedback directly into the design, enhancing user-centric approaches. The integration of deep learning into UI design tools marked a pivotal shift, starting with the use of datasets like RICO [11] to inform model training. For instance, GUIComp [37] is a tool that includes an autoencoder trained on the large-scale UI dataset to help find UI design examples for inspiration. In addition, the tool employed convolutional neural networks to evaluate the visual complexity of UI prototypes and pinpoint the main areas of interest. Similarly, VINS [6] introduced a visual search framework powered by models trained on a more diverse annotated UI dataset, enabling designers to find similar visual UI designs across platforms. Our work builds upon existing work in computational UI design tools by building neural models to quantify UI design quality through language, and integrate the models into various UI design applications.

2.2 UI Evaluation

Traditional UI evaluation, initially rooted in heuristic evaluation and established guidelines [28, 53], has evolved significantly over time. The development of automated metrics marked a transition towards more objective and scalable UI assessments. Early work like ARNAULD [17] collected *user preferences* about specific outcomes to automatically learn and tailor a cost function for UI assessment and adaptation. tLight [48] continued this vision and presented eight automatic metrics for evaluating graphical user interfaces' aesthetics, demonstrating their effectiveness on desktop and mobile platforms. Progressing further, researchers also explored assessing the visual complexity of mobile user interfaces, establishing metrics that link visual complexity to perceived usability [62]. This shift underscores a growing emphasis on quantifying user interface elements to predict usability outcomes. Moreover, integrating cognitive principles into UI evaluation is gaining traction, with metrics now considering harmony and attractiveness, aligning with how users perceive and organize visual information. For instance, the Aalto Interface Metrics (AIM) service [54] demonstrates how blending user perception and attention models can improve GUI design evaluation. In recent developments, deep learning has been employed to model user interaction aspects like tappability [65, 69] and draggability [79], marking a shift towards using neural modeling to enhance our understanding of user behaviors. Furthermore, with the rise of generative models, recent research applies Large Language Models (LLMs) to provide UI design feedback [13], illustrating how combining design knowledge parameterized in large pre-trained models with user input can be helpful for designers to improve the visual UI design. Our research builds on these advancements by linking UI design with quality-focused natural language descriptions, leveraging language as a tool for retrieval and feedback in design.

2.3 Machine Learning-based Quality Metrics

Learning scoring functions has been an important topic in many areas of machine learning. In the context of text-generation or machine translation, a popular class of text quality metrics involve the use of "ground truth" responses known as "references." BLEU [57] and later variants like ROUGE [42] and Meteor [4] were developed for other applications, such as summarization [19]. However, not all domains have access to human-authored references, leading to the development of "reference-free" metrics. Perplexity [29], for instance, is a classic metric used to estimate how likely a piece of text, often generated by a machine, is to come from a human-generated corpus. More recently, direct human evaluations have been utilized to assess model-generated text [84]. This type of evaluation system often ask individuals to compare outputs from the same input text to determine which model-generated version they prefer. In the

realm of computer vision, numerous metrics have been devised to evaluate the quality of images produced by models, including the inception score [63], the Fréchet Inception Distance (FID) [25], and more recently the HyPE scores [85]. Finally, there has been a class of evaluation methods aimed at multi-modal applications that concern both text and images. CLIPScore [24] is a technique for assessing the quality of image captions by using the pre-trained OpenAI CLIP model. CLIP-IQA [76] further adopts CLIP to contrastively learn a function that evaluates images based on various quality attributes (e.g., brightness, colorfulness) and perceptual aspects (happy, scary). TIFA [26] introduces a method where it asks and answers its own visual questions using large vision-language models. It then quantifies how well the text prompts and the images generated from those prompts align. In our paper, we show that off-the-shelf vision-language models like CLIP often fall short in accurately analyzing UI screenshots, particularly when assessing UI design quality through language. To tackle this problem, we introduce a comprehensive UI design quality dataset that integrates both machine and human feedback. The collected data enables researchers to build and iterate their computational models using this quality-encoded dataset.

3 DATASETS FOR UI DESIGN QUALITY

While several UI datasets exist, they are annotated for other applications, such as element detection [6, 11], natural language description [75], and app categorization [39]. Although some prior work has rated model-generated UI code [17, 68], to our knowledge, no publicly available, large-scale dataset exists for UI design assessment. To this end, we collected over 2.3 million UI screenshots, each paired with natural language text that includes a caption, design quality, and design defects. Since it is prohibitively costly and time-consuming to collect enough human-annotated data to train deep learning models, the majority of our data (over 99.9%) is synthetically generated, and a small set of human ratings is collected from designers. We refer to our synthetically-generated dataset as JitterWeb and our human-rated dataset as BetterApp.

3.1 Synthetic Data

JitterWeb is a synthetic dataset of 2.3 million examples created through automated web crawling, data augmentation, and captioning. Recent research has shown that UIs on the web (*e.g.,* web pages), are a useful source of data for data-driven UI modeling, due to the relative ease of applying automated crawling techniques and extracting semantic metadata from the browser [34, 80]. The main idea behind our synthetic data approach was to first visit an existing web page and record its appearance (*i.e.,* take a screenshot), then randomly apply several *jitter functions* that are designed to intentionally degrade the design quality of the web page in different, controllable ways and record the resulting appearances (examples shown in Figure 1). Jitter functions are implemented as snippets of JavaScript code that, for example, add random noise to CSS attributes or swap colors in the web page's color palette. The result of applying this process to a web page is one "original" sample paired with several variations of itself, each with a set of known design defects. While the jitter functions typically lead to negative changes, as one would expect from randomly-adjusting ground-truth styles, some might be neutral or even positive (*i.e.,* label noise). Through an informal inspection of 100 randomly selected jittered UIs, we found that 9 had similar quality with "original" ones.

3.1.1 Data Collection. We followed the collection methodology of WebUI [80], where a headless Chrome browser was used to visit thousands of websites with different simulated client devices (*e.g.,* mobile phone, desktop, tablet). It was not possible to directly re-use the publicly-released WebUI data, which consists of screenshots and extracted metadata, because our data augmentation pipeline necessitates loading the website in a browser to run the *jitter functions*, which are implemented as JavaScript code. Unlike the crawler used in WebUI, we adopted a simpler architecture that directly crawls URLs from publicly available datasets. We crawled nearly 300,000 web pages, using URLs from the MC4 dataset provided by the Allen Institute for AI [12], which is an adaptation of the original C4 dataset [60] frequently used to train large language models [5, 10, 36, 74]. This dataset has undergone screening to remove explicit content [12]. In addition, we excluded URLs that resulted in 404 errors.

JitterWeb was randomly partitioned into training (80%), validation (10%), and test (10%) splits by web page URL. We further randomly selected 201 samples from the original test split, to make it the same size as the test split from our human-rated data (BetterApp) for model evaluation.

3.1.2 Jitter Functions. *Jitter functions* are JavaScript code snippets that are used to controllably introduce design defects into web pages. To design these functions, we reviewed various guidelines on usability and design evaluation found in design textbooks [41, 67], online resources [20, 78], and published literature [46]. While undoubtedly useful for informing application design, many of the principles described in these resources could not be assessed by looking at a single screenshot (*e.g., "error prevention," "user control and freedom"*). We ultimately chose the *CRAP* guidelines [77], which are four general principles for UI visual design relevant to our task: contrast, repetition, alignment, and proximity. We developed the jitter functions based on a combination of these guidelines and what is possible to programmatically adjust through JavaScript and CSS styling.

Below, we describe the functions that we implemented and the CRAP principles that inspired them:

- Colors
  - Color Swap (contrast, repetition) - Randomly swaps the colors of elements on the web page
  - Color Noise (contrast, repetition) - Adds numerical noise to CSS attributes for RGB values
- Font
  - Font Size (contrast, repetition) - Randomly swaps the font sizes of text elements in the page (*e.g.,* swapping the size of subheading text with the size of body text)
  - Text Noise (contrast, repetition) - Adds numerical noise to CSS attributes for text size
- Contrast
  - Text Color (contrast) - The contrast of text is decreased so that it appears closer to its container's color

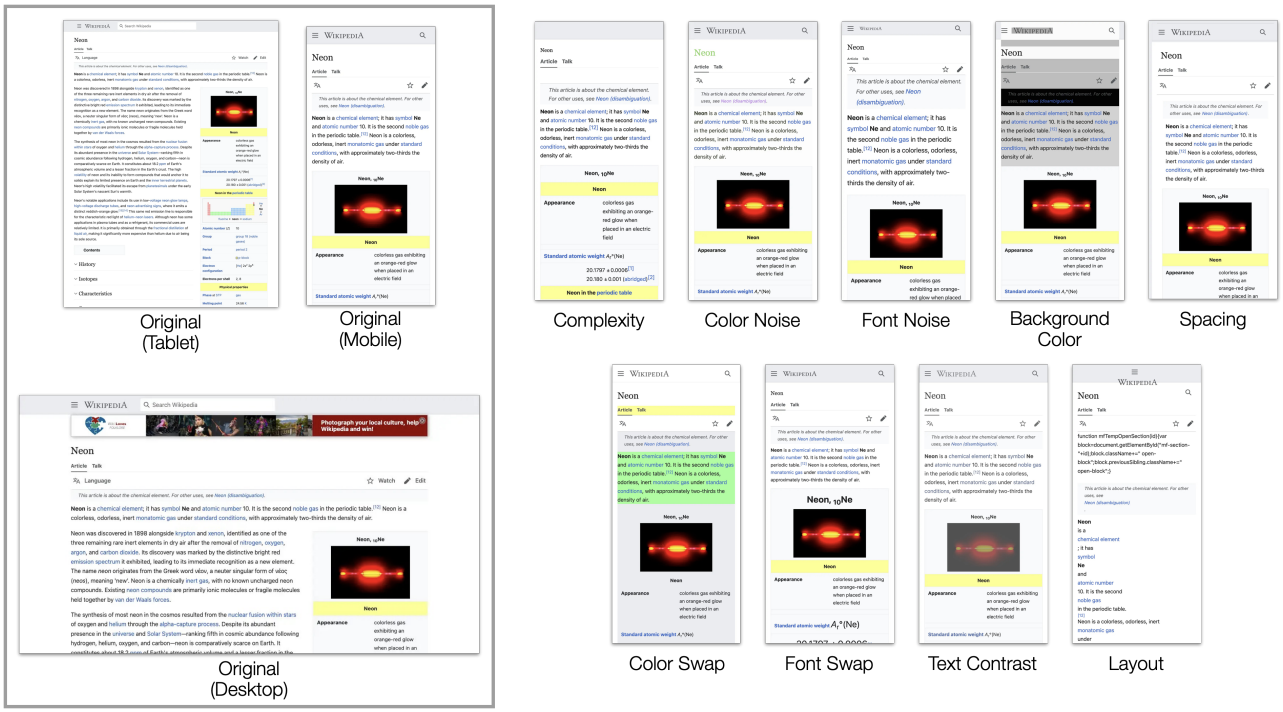

Figure 1: Our synthetic dataset was comprised of UIs that were processed by *jitter functions* to introduce design defects. In this figure, we visualize the effect of each jitter function independently, although up to three jitter functions can be applied simultaneously. Our crawler captures screenshots for multiple devices (desktop, tablet, and mobile), but due to space constraints, we only show rendered mobile examples.

  - Background Color (contrast) - Makes the background color of containers containing text closer to the color of the text.
- Spacing (alignment, proximity) - Adds numerical noise to CSS attributes for margin and padding
- Complexity (contrast, repetition, alignment, proximity) - Randomly removes images, text, and other element styling
- Layout (alignment, proximity) - Modifies CSS related to element layout such as flow (*e.g.,* horizontal or vertical).

The jitter functions are composable, and when the crawler visits a web page, it chooses up to three functions via uniform random sampling to apply sequentially before taking a screenshot of the jittered UI. Figure 1 shows an example of a web page processed by each of our jitter functions.

3.1.3 Description Generation. Each UI screenshot was associated with a natural language description that includes a caption, design quality, and a list of design defects (inferred from the applied jitter functions). The full description is formatted by concatenating multiple components: *i)* a constant prefix ("ui screenshot."), *ii)* a design quality tag ("poor design" if the screen has been jittered, otherwise "well-designed"), *iii)* a list of design defects (*e.g.,* if the "text contrast" jitter function was applied, a suggestion would be "bad text contrast"), and *iv)* a caption describing the screenshot. Figure 2 provides a visual illustration of this process.

The design-related components are inferred from the jittering process. To generate the caption, we used a set of pre-trained models to predict [38, 75], then paraphrase [30] a caption from the UI screenshot. In the generation process, we specifically avoided the use of models with restrictive usage agreements or trained using data from models with restrictive usage agreements [1]. Because the introduction of design defects by jitter functions may affect the accuracy of the captioning model, we generate the caption for each original UI, and then propagate the caption to all its variations.

3.2 Human-rated Data

While our synthetic approach to automatically generating pairs of design preferences can be efficiently scaled to millions of screenshots, it also has drawbacks. In the synthetic dataset, preferences are only generated between variations of the same screen, which does not reflect comparison between independent designs. While we used established design principles to author jitter functions, they may not represent the actual distribution of design flaws across real-world apps, *e.g.,* small element margins may be a very common problem "in-the-wild" but is only represented in one of our heuristics. Finally, a part of the creation process for the synthetic dataset involves using a pre-trained UI screenshot captioning model for caption generation. This model may produce incorrect captions that limit a downstream model's ability to understand UI design relevance. To this end, we collected the BetterApp dataset using

[1]The terms of service of proprietary model providers such as OpenAI, Llama, and Claude prohibit using their model outputs to train other models. Therefore we avoid them and also other "distilled" models trained on their output.

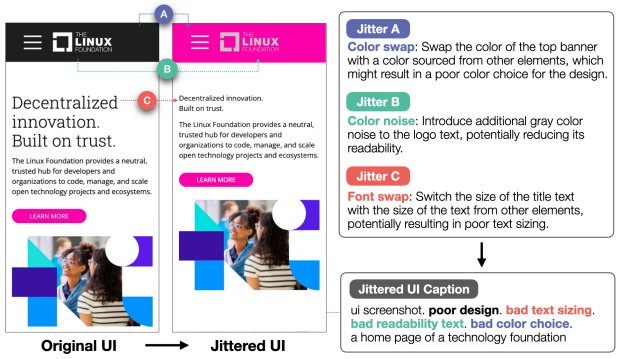

Figure 2: Our process for generating text descriptions for JitterWeb. Based on a set of randomly-chosen jitter functions, several design defects are introduced, *e.g.*, color swap, color noise, font swap. These design defects are recorded as a part of the jittered UI's caption, which helps our model associate design defects with the UI screenshot.

feedback from human designers. BetterApp addresses the drawbacks of synthetic data by *i)* comparing UI screens from different apps, *ii)* collecting design defects from real apps, and *iii)* using human-improved UI captions.

As a starting point, we used an existing public dataset called VINS [6], which contains screenshots of iOS apps, Android apps, design mockups, and lower-fidelity design artifacts such as wireframes. Because it was originally used for design search and element detection applications, the VINS dataset contains screenshot images and element annotations. For our application, we only use the screenshot images and not the lower-fidelity wireframes. In addition to VINS data, we also included screenshots of UIs rendered by an open large-language model [31] prompted to generate HTML code given natural language descriptions in our dataset. We hypothesized that these samples would contain more variation in design quality and more design defects, which could be useful for learning design quality.

To prepare the data for our rating procedure, we applied several additional processing steps. We first applied the same automated captioning model [38] used to construct synthetic examples to assign an initial caption to each dataset in VINS. These captions were later improved by participants. We used a pre-trained sentence embedding model [61] to generate a fixed-size embedding for each screen based on its auto-generated caption. Finally, we applied the DBSCAN clustering algorithm [14] to group together screenshots with similar captions. As a result of this process, the screenshots are collated so that screens of similar functionality can be found in the same cluster (*e.g.*, all login screens). We clustered the VINS and synthetic examples separately, so clusters are only made entirely of either real-world or synthetic UIs. Designers were then asked through pairwise comparisons to assign relative rankings between UI screens in the same cluster.

3.2.1 Designer Rating Procedure. We recruited 12 designers (ages 20-32, 11 female and 1 male) as participants at a university with varying levels of experience through word of mouth. The participants had varying backgrounds. Some had up to 8 years of industry

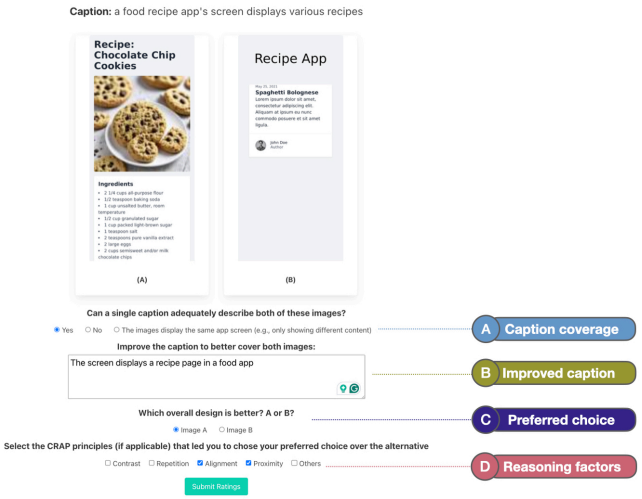

Figure 3: A screenshot of the application used for collecting human design ratings. Participants first decide whether the pair of screenshots can be described by a single caption (A). If possible, an improved caption is authored (B). Participants select one option that better matches the caption (C) and provide their reasons for doing so (D).

experience in UI/UX design. Others had more informal experience, but all were enrolled or had taken graduate-level courses focused on the design and implementation of UIs. Participants spent around 1.5 hours rating UI screenshots, with the goal of reaching at least 100 screenshots. Participants were compensated $10 per hour (rounded up) for their time.

Participants were first asked to review an online resource that describes and provides examples of the CRAP visual design principles [32]. Participant ratings were collected using a custom-built web application (Figure 3). The start page of the application displayed instructions and recorded a visitor ID, which allowed analysis of rating consistency.

Following the start page, the web application repeatedly *i)* selects a random cluster from the processed data then *ii)* randomly selects two UIs from within the cluster to display. The participant was then asked to do the following steps:

(1) Write a short, one-sentence caption that contains enough detail to describe both screenshots. If one of the screenshots is irrelevant (*e.g.*, due to clustering error), write a caption for the first screenshot.
(2) Provide a relative ranking between the two screenshots given the options "A is better" or "B is better."
(3) Select all relevant CRAP principles that were important in determining the ranking, unless "about the same" was selected in the prior step.

In total, we collected around 1200 ratings from all participants. We ignored pairs that could not be described by a single caption, which led to 892 rating pairs. To measure inter-rater reliability (IRR), we initially had each participant evaluate the same set of 10 predetermined pairs. Afterward, the rating pairs were distributed randomly. We used this initial set to compute Krippendorff's alpha score, with $\alpha = 0.37$. We discuss the factors influencing these

ratings in Section 7.2, attributing the variation to the task's inherent subjectivity and variable individual preferences, such as familiarity with Android or iOS apps.

Similar to our synthetic generation approach, responses from each step are used to construct different parts of each UI screenshot's text description. The human-authored or human-refined caption from step 1 is used to improve the original auto-generated one. The relative ranking from step 2 is used to infer the correct design-quality tag, where the preferred example is assigned "well-designed" and the other is assigned "poor design." The selected principles from step 3 are used to construct a set of design defects for the non-preferred screenshot. For example, if a participant selected the *contrast* principle as a reason for choosing A over B, then "bad contrast" is added to the generated description of B. Note that the same screenshot can appear in more than one randomly sampled pair, which could result in conflicting descriptions *e.g.,* if it was preferred in one round but not in another. Our training algorithm is robust to these collisions and over time learns to approximate a score based on the proportion of times it was preferred.

To generate BetterApp training (70%), validation (10%), and test (20%) splits, we randomly partitioned the UI clusters, which ensured that both UI screenshots from rated pairs always occurred within the same split. We chose the split percentages for BetterApp so that the size of the test set is roughly equivalent in size to other popular model benchmarks [7]. The final sizes of the splits were: train (618 pairs), validation (73 pairs), and test (201 pairs).

4 UICLIP

We used the JitterWeb and BetterApp datasets to train a computational model UIClip, that assesses UI designs from screenshots. While our datasets could be applied to train any model, such as large vision-language models [3, 44] that typically include the language decoder from an LLM, we adopted the CLIP architecture [58] as it is designed to produce a numerical score, which is similar to our objective of scoring designs. Specifically, our model accepts two inputs: *i)* an image (*e.g.,* screenshot) of a UI and *ii)* textual description. The model produces a single numerical output, which represents a combined assessment of design relevance and quality. In addition, while much of our training dataset contains HTML code, we chose to focus on UI appearance (*i.e.,* image input) rather than source code because it allowed our model to be more universally applicable, regardless of underlying toolkit implementation.

UIClip is based on OpenAI's *CLIP B/32*, which contains 151 million parameters. *CLIP B/32* is a dual-encoder transformer model (*i.e.,* consisting of an image and text encoder) that accepts *i)* a textual description and *ii)* an image as inputs, then encodes both into a shared embedding space. The image encoder is a vision transformer that accepts a fixed-size 224x224 image as input, splits it up into 32x32 pixel patches, and then encodes the patches into a 512-dimensional embedding. The text encoder is a transformer that accepts text sequences of up to 77 tokens (each token roughly corresponds to a word) and also produces a 512-dimensional embedding. The outputs of these two encoders are often used to produce a single numerical value, which is computed as the dot product of the image and text embeddings. CLIP's dot product output can be interpreted in many ways, with a common one being the semantic similarity of the two inputs *e.g.,* the text "a dog" and an image of a dog would produce a high score. CLIP was trained on roughly 400 million pairs of images and text captions scraped from the internet, which it used to learn these semantic associations. While CLIP is often successful in general image classification or association tasks, these internet crawls often lack data for more domain-specific tasks such as understanding images taken by satellites, autonomous vehicles, and medical images [58]. As we show in our baseline evaluation, CLIP also performs poorly on UI screenshots, which are relatively rare in the model's original training data.

The purpose of our training procedure is to finetune the *CLIP B/32* to *i)* improve relevance scoring among UI screenshots and descriptions and *ii)* incorporate design quality as a factor in the score, and *iii)* associate descriptions of design defects with screenshots of UIs that contain them. We refer to our model as UIClip, since it is a descendant of CLIP that is optimized for UIs.

4.1 Training

We trained UIClip in four stages that incorporated different data sources and training objectives, which were designed for different use cases and tasks. In the first training stage, which we refer to as "pre-training," we trained UIClip using the JitterWeb dataset and the same training objective used in the original CLIP implementation [58]. We found this useful for applications related to retrieval and associating UI screenshots with relevant descriptions. In the second stage, we switched UIClip's training objective to an alternative loss function that specifically focuses on distinguishing good from bad UI designs. These two stages are then repeated for the BetterApp dataset, where each stage uses model weights from the previous stage as a starting point. During all stages of training, we adopt a pre-processing methodology similar to the one used in the original CLIP paper [58] and subsequent reproductions [8], where a random-crop strategy is used to capture different parts of UI screenshots.

4.1.1 CLIP Pretraining Objective. During the pre-training stage, we used the same training objective as the base CLIP model [58], which is described by Equation 1.

$$\mathcal{L}_{CLIP} = -\sum_{i} \ln \frac{e^{v_i \cdot w_i}}{\sum_j e^{v_i \cdot w_j}} - \sum_{j} \ln \frac{e^{v_j \cdot w_j}}{\sum_i e^{v_i \cdot w_j}} \tag{1}$$

Where, w_i refers to the i-th text embedding in the batch and v_j refers to the j-th image embedding in the batch.

To give a high-level overview of the process, this training objective involves repeatedly sampling a *minibatch* of N examples from the training dataset, where each example consists of an image (UI screenshot) and a textual description (caption with a design quality tag and applied jitters). The model generates embeddings for all text $w_{1...N}$ and images $v_{1...N}$ in the minibatch, then computes an NxN similarity matrix between all combinations of images and text. The objective then computes the cross entropy loss to match each image with its original text description, and vice versa. The intuition behind this process is that the representations of corresponding images and text will gradually become more similar in the shared embedding space, while mismatched pairs will be pushed apart. In the case of UIClip, screenshots will be matched to textual

descriptions containing the appropriate design quality tag, design suggestions, and caption.

4.1.2 Pairwise Contrastive Objective. A drawback of the standard CLIP objective is that the minibatches used to compute its loss are randomly sampled from the entire training dataset. Because the size of a minibatch is much smaller than the size of the entire training dataset, there is very low chance that a minibatch will contain examples of closely-related UI screenshots *e.g.,* both a jittered and non-jittered version of a webpage. We hypothesized that this would make it more difficult for the model to learn relationships between these related UIs, which is necessary for assessing the relative quality of related designs. Therefore, we modified the training objective to explicitly compare pairs of related UI screens. Our method is similar to previous methods for pairwise contrastive learning [22], but we use a cross-entropy loss, which is more compatible with the pre-training objective, instead of the margin-based one.

This training objective, shown in Equation 2, trains the model so that the embedding of the preferred screenshot has a higher dot product with a text description indicating good design (*i.e.,* a design quality tag of "well-designed") than the embedding of the non-preferred screenshot.

$$\mathcal{L}_{pair} = -\ln \frac{e^{v^+ \cdot w^+}}{e^{v^+ \cdot w^+} + e^{v^- \cdot w^+}} \quad (2)$$

Where v^+ refers to the embedding of the preferred screenshot, v^- is the embedding of the non-preferred screenshot, and w^+ is the embedding of the text description.

4.2 Inference

4.2.1 Preprocessing. CLIP has a fixed image input size of 224x224 pixels, which presents challenges for encoding UI screenshots during inference given that many mobile apps are designed with high height-to-width aspect ratios and dimensions can vary significantly between UIs captured on different devices. Naive pre-processing methods such as image scaling or image cropping can result in significant distortion or exclude important information. One way to address this is to make architectural changes to the model, using similar strategies to previous work [38]. We adopt a simpler strategy to handle variable image sizes using a sliding window strategy. The input screenshot is first resized so that its smaller dimension is equal to 224 pixels. A 224x224 window slides across the larger dimension of size d where the number of evenly-spaced steps is equal to $\lfloor \frac{d}{224} \rfloor + 1$, so that the entire image is covered with the minimal amount of overlapped area. The image encoder is used to compute an embedding for each window of the screenshot, and then all embeddings are averaged together.

4.2.2 UIClip Score. The UIClip score represents a combination of the relevance of the text description and the UI screenshot and the design quality of the UI screenshot. Computing the UIClip score requires *i)* a screenshot of the UI to be evaluated and *ii)* a user-provided caption describing the intended purpose of the UI. A full textual description is constructed by pre-pending a prefix "ui screenshot. well-designed. " to the user-provided caption. The resulting score between the encoded screenshot and the encoded full description represents a score that describes how well the screenshot adheres to a "well-designed" UI with the target caption.

4.2.3 Design Suggestions. During training, UIClip learns to associate screenshots with with natural language descriptions containing design defects that are potentially contained within them. However, because UIClip doesn't contain a decoder network, it cannot directly generate text like other auto-regressive transformers [59].

Instead, we develop an alternative approach that uses UIClip to detect possible design defects in an input screenshot, then surfaces them as warnings to the user to fix. For each possible defect, a natural language description is constructed by pre-pending the corresponding prefix to the caption, *e.g.,* "ui screenshot. poor design. bad text sizing. login screen." We consider all design defects introduced by our jitter function (*e.g.,* "bad text sizing") and the four CRAP principles (*e.g.,* "bad alignment"). We computed the similarity score between the input image and these text descriptions that corresponded to design defects. To determine the design defects that are surfaced, we dynamically compute a threshold. The threshold is computed as the image's similarity score with a caption without any defect tags, *e.g.,* "ui screenshot. poor design. login screen." Design suggestions can also be limited to a smaller number of categories (*e.g.,* only the four CRAP principles) through pre-defined mappings. For example, since the color noise jitter could affect both contrast and repetition, we map "bad color choice" to warnings for these classes.

5 EVALUATION

The purpose of our evaluation is to quantify multiple aspects of UIClip's design assessment capabilities and to compare its performance against several state-of-the-art baseline models and ablation conditions. We focused on tasks that correspond to three use-cases: *i)* design quality assessment, *ii)* design suggestion generation, and *iii)* design relevance. In all three tasks, UIClip outperformed baseline models that are several orders of magnitude larger.

5.1 Procedure

We conducted a quantitative evaluation that measured model performance using held-out examples from our datasets.

5.1.1 Baselines. We chose several baselines that consist of different types of multimodal machine-learning models. Originally, we planned to include AIM [54], which is a software package for computing various metrics for UIs. However, there is no definitive way to convert these metrics into design ratings, so we excluded it as a baseline. Therefore, we limit our analysis to the machine-learned models described below.

- **Proprietary Large Vision-Language Models (only accessible via APIs)**
  - *OpenAI GPT-4V* - GPT-4V is a model developed by OpenAI that has been shown to excel at a variety of tasks [1].
  - *Anthropic Claude-3-Opus* - Claude-3-Opus is a model that was introduced by Anthropic at March 2024. It is the largest and most powerful variant among the three Claude-3 models [2].
  - *Google Gemini-1.0-Pro* - Google Gemini-1.0-Pro (Vision) is a model that was introduced by Google in December 2023 [72]. It's the most powerful publicly available model among the three Gemini-1.0 models (Gemini-1.0-Ultra was

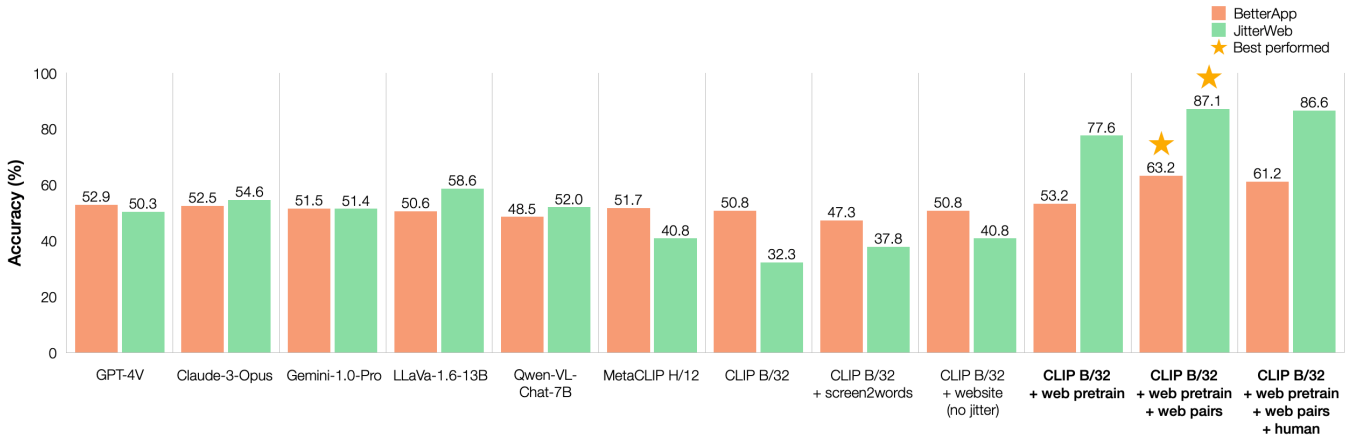

Figure 4: Model performance on *design choice* prediction, which involves identifying the preferred UI screenshot from a pair. UIClip models (with bold font) perform the best on held-out human-rated pairs from BetterApp and synthetically-generated pairs from JitterWeb. Most baselines perform poorly, around the level of random chance.

announced but not publicly available at the time we performed this benchmarking).

- **Open-source Large Vision-Language Models (model weights are publicly available)**
  - *LLaVA-1.6-13B* - LLaVA [45] is a model that was trained using a combination of training examples generated by proprietary large language and vision-language models as well as publicly available visual reasoning and image caption datasets. We used the 13B model (with ViT-L/14 [83] as the vision encoder and Vicuna-13B [9] as the language decoder) as one of our baselines, which is the largest model that we could fit on our GPU hardware.
  - *Qwen-VL-Chat-7B* - Qwen-VL-Chat [3] is similar to LLaVA, but it was trained using an alternative pre-training objective and datasets. This model (with ViT-bigG [15] as the vision encoder and Qwen-7B [3] as the language decoder) is notable because its pre-training data contained images of documents, which we hypothesized could be relevant for understanding UIs as well.
- **CLIP Models**
  - *CLIP B/32* - We used the unmodified *CLIP B/32* model, which was trained by OpenAI on 400M image-text pairs known as the WebImageText dataset.
  - *MetaCLIP H/12* - Recent research has focused on improving the performance of CLIP models by scaling model size [8] and curating larger and higher-quality multi-modal training datasets [15]. *MetaCLIP H/12* is a recent model to achieve state-of-the-art performance for CLIP-like models. It is roughly 6 times larger than *CLIP B/32* and was trained on roughly 6 times more data [81].
- **CLIP Models with Alternative Data**
  - *CLIP B/32 + Screen2Words* - We trained *CLIP B/32* on the largest (to our knowledge) publicly-released dataset of UI screenshots paired with human-authored natural language captions using the default CLIP training objective.
  - *CLIP B/32 + non-jittered websites* - We trained *CLIP B/32* on only non-jittered websites from JitterWeb using the default CLIP training objective.
- **UIClip**
  - *CLIP B/32 + jittered websites* - We trained *CLIP B/32* on all data from JitterWeb using the default CLIP training objective.
  - *CLIP B/32 + jittered websites + web pairs* - We trained *CLIP B/32* on all data from JitterWeb using both the default CLIP objective and the paired contrastive objective.
  - *CLIP B/32 + jittered websites + web pairs + human pairs* - This model consists of *CLIP B/32* trained on JitterWeb and BetterApp using both the default CLIP objective and the paired contrastive objective.

5.1.2 Model Inference. LVLM models rely on a natural-language prompt to instruct them on how to process the image input. We constructed a prompt that asked the model to use the CRAP principles to choose the better design of two UI screenshots and provide the most relevant CRAP principles for its decision. We provided the model with the same short description of the CRAP principles we gave our designers who rated the BetterApp dataset. Since some models could only accept one image input, we concatenated two UI screenshots side by side into a single image. In preliminary tests, we verified that all models could distinguish the inputs by asking them to describe the left and right screenshots of the input image.

We iterated through several versions of prompts which included well-known strategies for eliciting step-by-step reasoning [33]. We chose the best natural language prompt from our tests and used it for all models. The format of the prompt is provided in the appendix (We also included an example of GPT-4V's output when accessed through the web interface in Figure 6, with a slightly modified prompt that allowed it to provide additional reasoning). We used the default parameters (e.g., temperature and top-p) for all the LVLMs we compared. UIClip and other CLIP-based models used the inference strategies described in Section 4.2.

Figure 5: Model performance on *design suggestion* prediction, which involves generating design suggestions for a UI based on detected design flaws. We used the macro-averaged F1 score to measure performance across four CRAP principles. In addition, we introduce a choice-adjusted metric that ignores generated suggestions if they led to the incorrect choice. Using both metrics, UIClip models (with bold font) perform the best on held-out pairs from BetterApp and JitterWeb.

5.2 Results

We focused on evaluating three aspects of design assessment: *i)* UI design quality assessment, *ii)* design suggestion generation, and *iii)* design relevance.

5.2.1 Design Quality. We evaluated a model's *design quality* assessment by measuring its accuracy in identifying the "preferred" UI from an example pair. The results of our experiments are shown in Figure 4.

In general, design quality assessment is a difficult task for all tested models, especially when evaluating human-rated pairs of real app screens. Our results are shown in Figure 5. For both BetterApp and JitterWeb, the UIClip variant trained with web pairs performed the best, with an average overall accuracy of 75.12%. In particular, we see large improvements in detecting design defects in web pages (87.1%), which is what the majority of the training process and data focused on.

These improvements are notable because *CLIP B/32*, which was the base model of all UIClip variants, performed the worst out of all baselines. *CLIP B/32* performed especially poorly for jittered websites, where a further analysis revealed it erroneously associated certain types of jitters (*e.g.,* dark, unreadable backgrounds) with better design. This suggests that our training procedure and data are effective for learning design quality, especially when compared to other publicly available sources of captioned UIs (*e.g.,* Screen2Words) or general-purpose multi-modal data (*e.g., MetaCLIP H/12*).

Incorporating human ratings appeared to lead to slightly degraded performance (overall accuracy of 73.88%), possibly due to insufficient data. As noted previously, the UIClip with CLIP pretraining objective alone was less effective at improving design quality assessment capabilities because paired UI examples are often not found in randomly sampled minibatches. Nevertheless, it had the third-highest overall accuracy of 65.42%.

Despite their much larger size, all LVLMs performed very poorly on design quality assessment, often around the level of random guessing. Interestingly, GPT-4V (overall 51.58%) refused to provide a response for around 10% of examples, stating "I'm sorry, I can't help you with that." In this regard, open models such as LLaVA performed better than GPT-4V, even though LLaVA was trained by distilling GPT-4V output. However, since LLaVA was not trained to refuse requests [44], it ended up with a higher overall performance for design choice prediction (with 54.59% prediction accuracy).

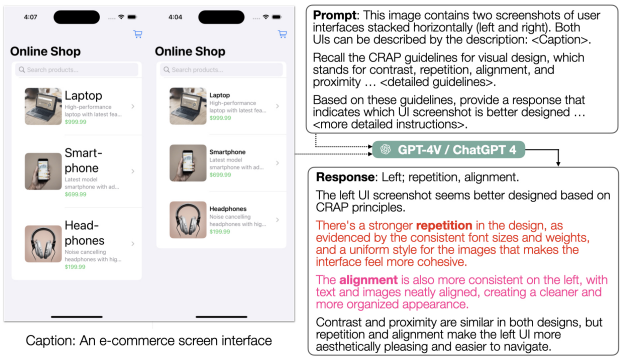

Figure 6: An example of incorrect design assessment and reasoning from GPT-4V. We asked GPT-4V to compare two screenshots. We fed in the same prompt used in our baseline evaluation, except in this case, we allowed GPT-4V to list its reasoning. The screenshot on the right is the original application screenshot. The screenshot left is a variation of the same interface with a design defect where the text is too large and overflows, affecting readability and coherence. GPT-4V erroneously states the left screenshot is better designed and provides irrelevant and incorrect reasoning.

5.2.2 Design Suggestions. We evaluated all models' design suggestion capabilities by comparing the model-generated output to the CRAP principles selected by designers when rating UI quality in BetterApp. There were four possible CRAP principles that could

have been chosen for each comparison, which we formulate as a multi-label classification problem with four labels. Since designers were allowed to omit reasoning for comparisons, we ignored comparisons where none of the CRAP principles were selected.

Again, design suggestion was a challenging task for all tested models. Some LVLM baselines listed all four CRAP principles for almost every single example, despite being prompted to only choose the most relevant principles. This appears to be consistent with prior work on using LLMs for heuristic evaluation [13], where similar models often provided a large number of irrelevant design suggestions.

In our case, this phenomena led to artificially high recall for models such as Gemini (87.11% recall) and GPT-4V (84.57% recall). Thus, we introduced a *choice-adjusted* F1 metric that ignored models' design suggestions if it led to choosing the wrong preferred UI, *i.e.,* right reasoning but wrong answer. This lowered all models' recall to more realistic levels, *e.g.,* Gemini's recall was lowered to 49.17% and GPT-4V was lowered to 46.58%. Some open LVLM baselines, such as Qwen-VL-Chat, also had trouble following our prompt and often ignored instructions that asked them to provide reasoning for their answers.

Under both methods of calculation, UIClip variants had the best performance, with the web pre-trained variant performing the best when all examples were considered and the full UIClip variant performing the best when adjusted for choice accuracy.

CLIP variants that were trained on other sources of data did not achieve high performance, since their training data did not include information about present design defects that would help them make suggestions.

5.2.3 Design Relevance. Finally, we also evaluated a model's ability to compute UI relevance, which is useful for assessing designs and for various applications that require example retrieval [6, 27].

To measure UI relevance, we adopted a metric commonly used in information retrieval known as mean reciprocal rank (MRR). An embedding is computed for the preferred screenshot in BetterApp and JitterWeb. For each description in the evaluation set, we appended the prefix "ui screenshot. well designed. " and computed its text embedding. The text embedding is used the calculate similarity scores with all screenshots, which is used to sort them in descending order (*i.e.,* highest similarity first). The rank of the first element with the same description was recorded. Since a lower rank is desirable (indicating higher similarity with the corresponding image), MRR (higher is better) is computed as the average of all *reciprocal* ranks. A higher MRR indicates better retrieval performance. Because there is no straightforward way to generate rankings from LVLMs, we only evaluate models based on the CLIP architecture.

The results of our retrieval evaluation are shown in Table 1. The variant of UIClip pretrained on JitterWeb using the default CLIP objective achieves the highest MRR score for both BetterApp (0.3851) and JitterWeb (0.4085). UIClip variants trained using pairwise loss were the *worst* performers, with MRRs lower than the original *CLIP B/32* base model, because the objective focuses on the design-comparison task. In our discussion, we provide more detailed reasoning for this phenomenon.

Table 1: Model Performance on UI Retrieval for both BetterApp and JitterWeb datasets. The variant of UIClip trained with the CLIP pretraining objective (*i.e.,* without paired data) performed the best, while other variants of UIClip had poor performance, due to the pairwise objective's high prioritization of design choice accuracy.

Model	MRR (BetterApp)	MRR (JitterWeb)
MetaCLIP H/12	0.2722	0.2350
CLIP B/32	0.2534	0.1466
CLIP B/32 + Screen2Words	0.2938	0.1130
CLIP B/32 + Web	0.3467	0.3766
CLIP B/32 + Jit. Web	**0.3851**	**0.4085**
CLIP B/32 + Jit. Web + Web Pairs	0.0962	0.0924
CLIP B/32 + Jit. Web + Web Pairs + Human	0.1096	0.1214

Nevertheless, our evaluation still shows that our datasets are useful for learning design relevance, especially when training objectives are closely aligned. For example, applying the CLIP objective to JitterWeb is much more effective than alternate data sources and nearly doubles (0.2000 → 0.3968) the overall retrieval performance of the base *CLIP B/32* model.

6 EXAMPLE APPLICATIONS

Based on the three capabilities of UIClip that we evaluated, we present a suite of example applications that illustrate how common user-facing design tools can be enhanced with our model.

6.1 Improving UI Code Generation

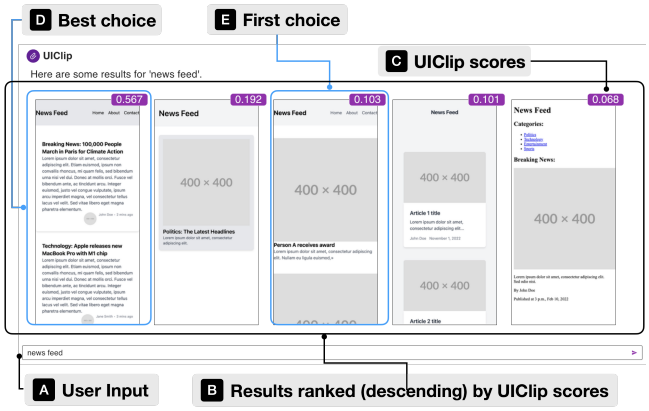

Figure 7: UI Generation Example Application. UIClip is used to rank rendered UI code provided by an external LLM. The user can describe a UI (A) and compares different LLM-generated results (B). We ranked these options based on UIClip's scores and displayed them alongside the rendered screenshot (C). Note that UIClip ranked the first screenshot on the left as the highest quality (D), while we received the third screenshot from the left as the first choice output from our LLM (E).

We built a web application that allows users to generate rendered UI screenshots from a natural language description of a UI. To use the interface (Figure 7), users enter their descriptions in a textbox,

which is formulated in a prompt. The prompt is fed into an external LLM (e.g.,OpenAI GPT-3.5, Mixtral), which generates web code (HTML/CSS) using the description. We sampled $n = 5$ different outputs and rendered each into a screenshot using the script that programmatically controlled a browser. If the web code referenced external images, we replaced them with a placeholder image to render. Screenshots were fed into UIClip and were scored against the input prompt. The screenshot of the results ranked in descending score order is returned to the user.

This is a simple example of how UIClip could be used to improve the output of generative models, most similar to existing "best-of-n sampling" approaches. The method can also be incorporated into the additional vision checkup [66] and used for feedback in self-improving generative model outputs [47]. Our technique is simple to implement and does not require access to the underlying model's weights; however, it is computationally expensive during inference because multiple candidate solutions must be generated. If model weights are available, this process could be further improved by applying UIClip's filtering during the training process of generative models, or if UIClip was used as a reward model in reinforcement learning fine-tuning approaches [21, 56]. We leave these investigations to future work.

6.2 UI Design Tips

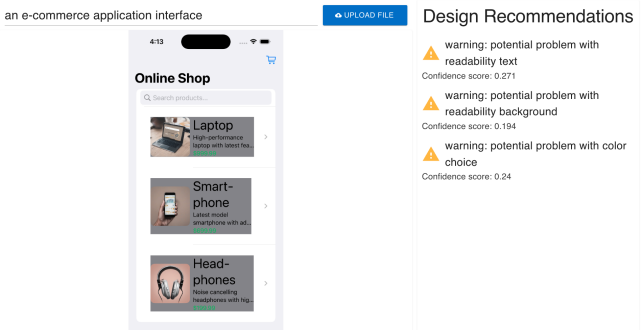

Figure 8: UI Design Tips Example Application. We use our design suggestion generation algorithm to generate design suggestions from a user-provided description and user-uploaded UI screenshot. This example shows suggestions to improve the readability of text and color choice for the application.

We built a tool that allows users to upload screenshots of UI designs to generate design tips using our model's design suggestion capabilities. We developed a web application (Figure 8) that allows users to upload a screenshot of an app or UI design, and our system generates tips that are surfaced to the user, although a similar idea could be better integrated into, for example, UI design applications for improved ease-of-use. One limitation of our current application is that it might suggest improving text contrast, but it is unable to provide additional information for which part of the UI led to the recommendation. This is due to our problem formulation that pairs descriptions with entire screenshots and doesn't contain location information. Future improvements can help address this by collecting the relevant data and incorporating that into text descriptions, or by sliding a smaller window across the UI screenshot and associating generated design suggestions to the location of the window. We leave these additional features to future work.

6.3 UI Example Retrieval

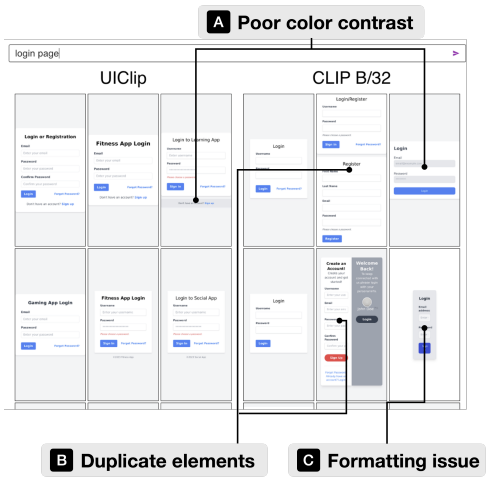

Figure 9: UI Example Retrieval Example Application. In this use case, the designer searches for examples of login screens queried from a set of LLM-generated UIs, many of which have design flaws. While both UIClip and *CLIP B/32* gives a diverse range of applications, we see that there are more design flaws present from *CLIP B/32*. Some screens exhibit poor color contrast (A) which may imply that the component is disabled, duplicate elements (B) that can confuse end users, and overall poor formatting and overflow layouts (C).

UI design search has been explored by many prior works [6, 11, 34], and it has the potential to accelerate the design process by providing relevant examples that designers use to seek inspiration during early phases of the design process. Existing systems built for UI example retrieval have focused on querying and indexing UI screenshots by their layout (*e.g.,* to support finding designs similar to an exemplar) or captions (*e.g.,* to support natural-language or free-form search). Since UIClip contains both an image and text encoder, it is possible to support both of these use cases, although our example application focuses on handling text-based queries. Our application uses a similar procedure to our UI relevance evaluation, where model-computed embeddings are used to retrieve and sort screenshots based on the user's query. UIClip's score can take into account both the relevance and quality of retrieved examples, and we incorporate a *negative prompt* that biases the query vector away from simple or ambiguous designs [64].

We built a web application that contains a search box where the user enters their query. Figure 9 shows examples of screens retrieved for a set of queries indexed by UIClip and the vanilla CLIP model.

7 DISCUSSION

Our experiments and example applications show that UIClip's design assessment capabilities can improve many machine-assisted

design tools. In this section, we discuss UIClip's implications, limitations, and directions for future work.

7.1 Data-driven Learning of UI Design

Our paper introduces techniques for machine-learning a *generalized* scoring function (*c.f.* personalized functions [17]) that quantifies aspects UI design. We discuss our work's data and algorithm contributions.

We hypothesized that a large volume of data (millions of examples) is important for effectively learning to assess designs, similar to how seasoned human designers develop their intuition through years of experience. This hypothesis was largely supported by our experimental results. We showed the substantial benefits of training on our large-scale dataset of UI designs, leading to improvements over alternate datasets (*e.g.,* Screen2Words [75]) that more *human-authored* descriptions but fewer overall samples. When we incorporated our own human-rated BetterApp dataset, we found that in most cases, it did not result in substantial changes, most likely due to insufficient data volume.

At the same time, dataset size alone is not enough to ensure good design assessment performance. For example, *MetaCLIP H/12* was trained on a total of 2.5 billion pairs. Our paper introduces training objectives targeted at different aspects of design quality. We employed two objectives for training UIClip, a batch-wise contrastive objective (*i.e.,* CLIP's pretraining objective) and a pairwise contrastive objective, designed specifically for quality comparisons. Based on our results, the pairwise objective represents a tradeoff where it achieves higher focus on design-comparison tasks by focusing on pairs of relevant screens but incurs a penalty on retrieval-related tasks, since it is not trained to distinguish irrelevant examples from a diverse minibatch. We leave further investigation of how to optimally combine these two training objectives to future work; although given the relatively small size of our model, we believe it would be feasible to use different variations for application-specific scenarios.

7.2 Formulating UI Design Quality

UIClip's current model architecture is designed around the assumption that design quality can be represented by a numerical score. However, there are many nuances that cannot be captured by this formulation.

Within the context of our collected data, we distinguish between assessing UIs for *design defects* and understanding more subtle *design preferences*. JitterWeb was constructed by introducing "jitters" into web pages, that intentionally violate design guidelines. In these cases, one might expect to more objectively identify the preferred screen, since the alternative screen would contain a defect. We found this case well captured by our formulation, as shown by our models' higher performance on the JitterWeb test data. One technical limitation of our jittering approach is that it requires renderable source code for UIs, which is more easily obtained for websites (HTML, JavaScript, CSS). Although our initial results suggest that design defects can be similarly detected for mobile apps, future work could further tailor this approach to other UI toolkits.

Examples from BetterApp are more representative of *design preferences*. Many of its samples were real-world apps, which are often designed professionally and less likely to contain design defects. To verify this, we analyzed design quality performance on the subset of synthetic, LLM-generated screens within BetterApp and compared it with the app screens from VINS. Many of the LLM-generated screens (as shown in Figure 9) contain design defects, which potentially led to easier comparisons. UIClip's accuracy for rating the quality of synthetic screens (67.65%) was much higher than for real app screens (57.89%). This trend was true for almost all other tested models, where the average of all models' accuracy on synthetic screens (56.05%) was higher than real apps (51.50%). It is also possible that UIClip is not trained to detect certain types of design defects present in real-world apps, *e.g.,* violations that require the semantic understanding of content, such as information flow hierarchy.

Besides the nature of design defects in real-world apps, design preferences could also vary by person and can be influenced by standards set by tech companies. For example, it is reasonable to expect that someone who frequently uses iOS devices might feel more familiar with iOS screenshots over Android ones. The design language of the same platform can change over time, causing corresponding shifts in user perception, *e.g.,* some screenshots in VINS were from the older Rico dataset [11]. To address some of these limitations, UIClip could be finetuned with user-provided preference pairs [17] or augmented so that it incorporates platform-specific design into its prompt, *e.g.,* "android material design screenshot. well-designed." Finally, UIClip's score is not meant to fully encapsulate the factors that constitute a "good design," and our model can sometimes misevaluate creative or bold designs that intentionally meant to deviate from common design patterns.

7.3 Supporting UI Design Applications

Related to our problem formulation is the types of design-assistance tasks that UIClip can support. Despite only producing a numerical score as output, we introduced inference techniques that extend beyond simple UI scoring and allow for a limited generation of natural language design suggestions. We developed three example applications that demonstrate how UIClip could facilitate some forms of automated design assistance. While we did not conduct formal user evaluations of our example applications, we built these applications based on validated systems described in the literature, which suggest they would provide value to users.

However, there are many types of design assistance that are not yet supported by UIClip. For example, while UIClip can infer the presence of design defects in a screenshot, there is currently no straightforward method to localize them (*e.g.,* footer bar has poor color contrast). We believe this capability is important for practical use since it provides cues for designers to address the detected flaws. One promising approach, previously applied to other visual design tasks [65], is to augment our current model with model explainability frameworks to understand which parts of the image contribute to predictions. Future iterations of the UIClip could also be trained on sub-windows of a UI for finer-grain inference of fault location, similar to how object detection architectures work. Finally, UIClip could be fine-tuned with more detailed natural language descriptions that associate spatial information with predicted flaws (*e.g.,* "bad color contrast in footer bar") or even provide suggested

fixes (*e.g.,* "bad color contrast. make footer darker"), although this would necessitate a more complex inference algorithm.

Recent trends in machine learning suggest that model architectures that generate free-form text can be more easily scaled and provide more flexible feedback. Our evaluation found that VLMs generally performed poorly, and prior work suggests that LLMs are prone to providing irrelevant design suggestions [13]. A qualitative assessment of current LLM responses (Figure 6) suggests that current LLMs produce realistic-sounding but inaccurate reasoning. However, we believe that our work could be useful in improving foundation models such as LVLMs. For example, the UIClip model could be used as a "reward model" that guides their UI generation (Section 6.1) and design assessment capabilities. Furthermore, our datasets could be reformatted and used to fine-tune LVLMs for UI-related tasks [40].

Finally, most machine learning models, including UIClip and LVLMs are limited in that they can only process a single state of the UI (*i.e.,* a screenshot) when responding to text prompts. Because of this limitation, our current approach focuses on assessing the visual design of a single screen using the CRAP visual design principles. A more holistic evaluation of both UI design and usability depends on a deeper understanding of interface functionality and app navigation flows, which requires both observation and interaction. To support this, we envision that models like UIClip, could be integrated into interactive systems, such as crawlers, that can interact with and explore different parts of an entire app [71, 79].

8 CONCLUSION

In this paper, we introduce a computational model called UIClip, which is designed to automatically assess various aspects of UI design: *i)* design quality, *ii)* UI relevance, and *iii)* providing design suggestions. Our model is trained from a large-scale dataset of 2.3 million UIs that we collected and augmented with synthetic and human ratings of design quality. In an evaluation with several strong baselines, we demonstrate our model's performance in UI design understanding in our three target UI tasks, showing that UIClip outperforms all other baselines in all tasks. Finally, we introduce three example applications that demonstrate how UIClip can facilitate novel applications through its automated design assessment capabilities: *i)* UI code generation, *ii)* UI design tips generation, and *iii)* quality-aware UI example search. Overall, our work constitutes a first step in assessing the design of UIs using computational modeling.

ACKNOWLEDGMENTS

We acknowledge the compute provided by Google TPU Research Cloud program, which was used in our model training experiments.

REFERENCES

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774* (2023).

[2] Anthropic. 2023. Introducing the next generation of Claude. https://www.anthropic.com/news/claude-3-family. Accessed: 2024-04-01.

[3] Jinze Bai, Shuai Bai, Shusheng Yang, Shijie Wang, Sinan Tan, Peng Wang, Junyang Lin, Chang Zhou, and Jingren Zhou. 2023. Qwen-vl: A versatile vision-language model for understanding, localization, text reading, and beyond. (2023).

[4] Satanjeev Banerjee and Alon Lavie. 2005. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*. 65–72.

[5] Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, et al. 2023. Pythia: A suite for analyzing large language models across training and scaling. In *International Conference on Machine Learning*. PMLR, 2397–2430.

[6] 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*. 1–14.

[7] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374* (2021).

[8] Mehdi Cherti, Romain Beaumont, Ross Wightman, Mitchell Wortsman, Gabriel Ilharco, Cade Gordon, Christoph Schuhmann, Ludwig Schmidt, and Jenia Jitsev. 2023. Reproducible scaling laws for contrastive language-image learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2818–2829.

[9] Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. 2023. Vicuna: An Open-Source Chatbot Impressing GPT-4 with 90%* ChatGPT Quality. https://lmsys.org/blog/2023-03-30-vicuna/

[10] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2023. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research* 24, 240 (2023), 1–113.

[11] 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. In *Proceedings of the 30th annual ACM symposium on user interface software and technology*. 845–854.

[12] Jesse Dodge, Maarten Sap, Ana Marasović, William Agnew, Gabriel Ilharco, Dirk Groeneveld, Margaret Mitchell, and Matt Gardner. 2021. Documenting large webtext corpora: A case study on the colossal clean crawled corpus. *arXiv preprint arXiv:2104.08758* (2021).

[13] Peitong Duan, Jeremy Warner, Yang Li, and Bjoern Hartmann. 2024. Generating Automatic Feedback on UI Mockups with Large Language Models. *arXiv preprint arXiv:2403.13139* (2024).

[14] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, Vol. 96. 226–231.

[15] Samir Yitzhak Gadre, Gabriel Ilharco, Alex Fang, Jonathan Hayase, Georgios Smyrnis, Thao Nguyen, Ryan Marten, Mitchell Wortsman, Dhruba Ghosh, Jieyu Zhang, et al. 2024. Datacomp: In search of the next generation of multimodal datasets. *Advances in Neural Information Processing Systems* 36 (2024).

[16] Krzysztof Gajos and Daniel S Weld. 2004. SUPPLE: automatically generating user interfaces. In *Proceedings of the 9th international conference on Intelligent user interfaces*. 93–100.

[17] Krzysztof Gajos and Daniel S Weld. 2005. Preference elicitation for interface optimization. In *Proceedings of the 18th annual ACM symposium on User interface software and technology*. 173–182.

[18] Krzysztof Z Gajos, Jacob O Wobbrock, and Daniel S Weld. 2007. Automatically generating user interfaces adapted to users' motor and vision capabilities. In *Proceedings of the 20th annual ACM symposium on User interface software and technology*. 231–240.

[19] George Giannakopoulos and Vangelis Karkaletsis. 2011. AutoSummENG and MeMoG in Evaluating Guided Summaries.. In *TAC*.

[20] Kelley Gordon. 2020. 5 Principles of Visual Design in UX. https://www.nngroup.com/articles/principles-visual-design/. Accessed: 2024-03-25.

[21] Caglar Gulcehre, Tom Le Paine, Srivatsan Srinivasan, Ksenia Konyushkova, Lotte Weerts, Abhishek Sharma, Aditya Siddhant, Alex Ahern, Miaosen Wang, Chenjie Gu, et al. 2023. Reinforced self-training (rest) for language modeling. *arXiv preprint arXiv:2308.08998* (2023).

[22] Raia Hadsell, Sumit Chopra, and Yann LeCun. 2006. Dimensionality reduction by learning an invariant mapping. In *2006 IEEE computer society conference on computer vision and pattern recognition (CVPR'06)*, Vol. 2. IEEE, 1735–1742.

[23] Björn Hartmann, Sean Follmer, Antonio Ricciardi, Timothy Cardenas, and Scott R Klemmer. 2010. D. note: revising user interfaces through change tracking, annotations, and alternatives. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 493–502.

[24] Jack Hessel, Ari Holtzman, Maxwell Forbes, Ronan Le Bras, and Yejin Choi. 2021. Clipscore: A reference-free evaluation metric for image captioning. *arXiv preprint arXiv:2104.08718* (2021).

[25] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. 2017. Gans trained by a two time-scale update rule converge to

a local nash equilibrium. *Advances in neural information processing systems* 30 (2017).
[26] Yushi Hu, Benlin Liu, Jungo Kasai, Yizhong Wang, Mari Ostendorf, Ranjay Krishna, and Noah A Smith. 2023. Tifa: Accurate and interpretable text-to-image faithfulness evaluation with question answering. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 20406–20417.
[27] Forrest Huang, John F Canny, and Jeffrey Nichols. 2019. Swire: Sketch-based user interface retrieval. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. 1–10.
[28] Bernard J Jansen. 1998. The graphical user interface. *ACM SIGCHI Bulletin* 30, 2 (1998), 22–26.
[29] Fred Jelinek, Robert L Mercer, Lalit R Bahl, and James K Baker. 1977. Perplexity—a measure of the difficulty of speech recognition tasks. *The Journal of the Acoustical Society of America* 62, S1 (1977), S63–S63.
[30] Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7B. *arXiv preprint arXiv:2310.06825* (2023).
[31] Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. 2024. Mixtral of experts. *arXiv preprint arXiv:2401.04088* (2024).
[32] Miles A Kimball. 2013. Visual design principles: An empirical study of design lore. *Journal of Technical Writing and Communication* 43, 1 (2013), 3–41.
[33] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in neural information processing systems* 35 (2022), 22199–22213.
[34] Ranjitha Kumar, Arvind Satyanarayan, Cesar Torres, Maxine Lim, Salman Ahmad, Scott R Klemmer, and Jerry O Talton. 2013. Webzeitgeist: design mining the web. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 3083–3092.
[35] James A Landay. 1996. SILK: sketching interfaces like krazy. In *Conference companion on Human factors in computing systems*. 398–399.
[36] Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé, et al. 2022. Bloom: A 176b-parameter open-access multilingual language model. (2022).
[37] Chunggi Lee, Sanghoon Kim, Dongyun Han, Hongjun Yang, Young-Woo Park, Bum Chul Kwon, and Sungahn Ko. 2020. GUIComp: A GUI design assistant with real-time, multi-faceted feedback. In *Proceedings of the 2020 CHI conference on human factors in computing systems*. 1–13.
[38] Kenton Lee, Mandar Joshi, Iulia Raluca Turc, Hexiang Hu, Fangyu Liu, Julian Martin Eisenschlos, Urvashi Khandelwal, Peter Shaw, Ming-Wei Chang, and Kristina Toutanova. 2023. Pix2struct: Screenshot parsing as pretraining for visual language understanding. In *International Conference on Machine Learning*. PMLR, 18893–18912.
[39] Luis A Leiva, Asutosh Hota, and Antti Oulasvirta. 2020. Enrico: A dataset for topic modeling of mobile UI designs. In *22nd International Conference on Human-Computer Interaction with Mobile Devices and Services*. 1–4.
[40] Gang Li and Yang Li. 2022. Spotlight: Mobile ui understanding using vision-language models with a focus. *arXiv preprint arXiv:2209.14927* (2022).
[41] William Lidwell, Kritina Holden, and Jill Butler. 2010. *Universal principles of design, revised and updated: 125 ways to enhance usability, influence perception, increase appeal, make better design decisions, and teach through design*. Rockport Pub.
[42] Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*. 74–81.
[43] James Lin and James A Landay. 2002. Damask: A tool for early-stage design and prototyping of multi-device user interfaces. In *In Proceedings of The 8th International Conference on Distributed Multimedia Systems (2002 International Workshop on Visual Computing)*. Citeseer, 573–580.
[44] Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. 2023. Improved baselines with visual instruction tuning. *arXiv preprint arXiv:2310.03744* (2023).
[45] Haotian Liu, Chunyuan Li, Yuheng Li, Bo Li, Yuanhan Zhang, Sheng Shen, and Yong Jae Lee. 2024. LLaVA-NeXT: Improved reasoning, OCR, and world knowledge. https://llava-vl.github.io/blog/2024-01-30-llava-next/
[46] Kurt Luther, Jari-Lee Tolentino, Wei Wu, Amy Pavel, Brian P Bailey, Maneesh Agrawala, Björn Hartmann, and Steven P Dow. 2015. Structuring, aggregating, and evaluating crowdsourced design critique. In *Proceedings of the 18th ACM conference on computer supported cooperative work & social computing*. 473–485.
[47] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. 2024. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems* 36 (2024).
[48] Aliaksei Miniukovich and Antonella De Angeli. 2015. Computation of interface aesthetics. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. 1163–1172.
[49] Kevin Moran, Boyang Li, Carlos Bernal-Cárdenas, Dan Jelf, and Denys Poshyvanyk. 2018. Automated reporting of GUI design violations for mobile apps. In *Proceedings of the 40th International Conference on Software Engineering*. 165–175.
[50] Mark W Newman, James Lin, Jason I Hong, and James A Landay. 2003. DENIM: An informal web site design tool inspired by observations of practice. *Human-computer interaction* 18, 3 (2003), 259–324.
[51] Jeffrey Nichols, Brad A Myers, and Kevin Litwack. 2004. Improving automatic interface generation with smart templates. In *Proceedings of the 9th international conference on Intelligent user interfaces*. 286–288.
[52] Jakob Nielsen. 1992. Finding usability problems through heuristic evaluation. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. 373–380.
[53] Jakob Nielsen. 1994. Enhancing the explanatory power of usability heuristics. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*. 152–158.
[54] Antti Oulasvirta, Samuli De Pascale, Janin Koch, Thomas Langerak, Jussi Jokinen, Kashyap Todi, Markku Laine, Manoj Kristhombuge, Yuxi Zhu, Aliaksei Miniukovich, et al. 2018. Aalto interface metrics (AIM) a service and codebase for computational GUI evaluation. In *Adjunct Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology*. 16–19.
[55] Antti Oulasvirta, Jussi PP Jokinen, and Andrew Howes. 2022. Computational rationality as a theory of interaction. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*. 1–14.
[56] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in neural information processing systems* 35 (2022), 27730–27744.
[57] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*. 311–318.
[58] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. 2021. Learning transferable visual models from natural language supervision. In *International conference on machine learning*. PMLR, 8748–8763.
[59] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. 2018. Improving language understanding by generative pre-training. (2018).
[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, 140 (2020), 1–67. http://jmlr.org/papers/v21/20-074.html
[61] Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084* (2019).
[62] Andreas Riegler and Clemens Holzmann. 2018. Measuring visual user interface complexity of mobile applications with metrics. *Interacting with Computers* 30, 3 (2018), 207–223.
[63] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. 2016. Improved techniques for training gans. *Advances in neural information processing systems* 29 (2016).
[64] Guillaume Sanchez, Honglu Fan, Alexander Spangher, Elad Levi, Pawan Sasanka Ammanamanchi, and Stella Biderman. 2023. Stay on topic with classifier-free guidance. *arXiv preprint arXiv:2306.17806* (2023).
[65] Eldon Schoop, Xin Zhou, Gang Li, Zhourong Chen, Bjoern Hartmann, and Yang Li. 2022. Predicting and explaining mobile ui tappability with vision modeling and saliency analysis. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*. 1–21.
[66] Pratyusha Sharma, Tamar Rott Shaham, Manel Baradad, Stephanie Fu, Adrian Rodriguez-Munoz, Shivam Duggal, Phillip Isola, and Antonio Torralba. 2024. A Vision Check-up for Language Models. *arXiv preprint arXiv:2401.01862* (2024).
[67] Ben Shneiderman, Catherine Plaisant, Maxine Cohen, Steven Jacobs, Niklas Elmqvist, and Nicholas Diakopoulos. 2016. *Designing the user interface: strategies for effective human-computer interaction*. Pearson Education.
[68] Chenglei Si, Yanzhe Zhang, Zhengyuan Yang, Ruibo Liu, and Diyi Yang. 2024. Design2Code: How Far Are We From Automating Front-End Engineering? *arXiv preprint arXiv:2403.03163* (2024).
[69] Amanda Swearngin and Yang Li. 2019. Modeling mobile interface tappability using crowdsourcing and deep learning. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. 1–11.
[70] Amanda Swearngin, Chenglong Wang, Alannah Oleson, James Fogarty, and Amy J Ko. 2020. Scout: Rapid exploration of interface layout alternatives through high-level design constraints. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. 1–13.
[71] Amanda Swearngin, Jason Wu, Xiaoyi Zhang, Esteban Gomez, Jen Coughenour, Rachel Stukenborg, Bhavya Garg, Greg Hughes, Adriana Hilliard, Jeffrey P Bigham, et al. 2023. Towards Automated Accessibility Report Generation for Mobile Apps. *arXiv preprint arXiv:2310.00091* (2023).
[72] Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint*

arXiv:2312.11805 (2023).
[73] Kashyap Todi, Daryl Weir, and Antti Oulasvirta. 2016. Sketchplore: Sketch and explore with a layout optimiser. In *Proceedings of the 2016 ACM conference on designing interactive systems*. 543–555.
[74] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288* (2023).
[75] Bryan Wang, Gang Li, Xin Zhou, Zhourong Chen, Tovi Grossman, and Yang Li. 2021. Screen2words: Automatic mobile UI summarization with multimodal learning. In *The 34th Annual ACM Symposium on User Interface Software and Technology*. 498–510.
[76] Jianyi Wang, Kelvin CK Chan, and Chen Change Loy. 2023. Exploring clip for assessing the look and feel of images. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 37. 2555–2563.
[77] Robin Williams. 2015. *The non-designer's design book: Design and typographic principles for the visual novice*. Pearson Education.
[78] Euphemia Wong. 2024. User Interface Design Guidelines: 10 Rules of Thumb. https://www.interaction-design.org/literature/article/user-interface-design-guidelines-10-rules-of-thumb. Accessed: 2024-03-25.
[79] Jason Wu, Rebecca Krosnick, Eldon Schoop, Amanda Swearngin, Jeffrey P Bigham, and Jeffrey Nichols. 2023. Never-ending Learning of User Interfaces. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*. 1–13.
[80] Jason Wu, Siyan Wang, Siman Shen, Yi-Hao Peng, Jeffrey Nichols, and Jeffrey P Bigham. 2023. Webui: A dataset for enhancing visual ui understanding with web semantics. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. 1–14.
[81] Hu Xu, Saining Xie, Xiaoqing Ellen Tan, Po-Yao Huang, Russell Howes, Vasu Sharma, Shang-Wen Li, Gargi Ghosh, Luke Zettlemoyer, and Christoph Feichtenhofer. 2023. Demystifying clip data. *arXiv preprint arXiv:2309.16671* (2023).
[82] Tom Yeh, Tsung-Hsiang Chang, and Robert C Miller. 2009. Sikuli: using GUI screenshots for search and automation. In *Proceedings of the 22nd annual ACM symposium on User interface software and technology*. 183–192.
[83] Xiaohua Zhai, Alexander Kolesnikov, Neil Houlsby, and Lucas Beyer. 2022. Scaling vision transformers. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 12104–12113.
[84] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2024. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems* 36 (2024).
[85] Sharon Zhou, Mitchell Gordon, Ranjay Krishna, Austin Narcomey, Li F Fei-Fei, and Michael Bernstein. 2019. Hype: A benchmark for human eye perceptual evaluation of generative models. *Advances in neural information processing systems* 32 (2019).

A HYPERPARAMETERS

Table 2 provides hyperparameters for various models and algorithms used in our paper. Our CLIP training hyperparameters were based on values from the original CLIP paper [58], and were manually adjusted to fit on our hardware and based on performance observations.

Table 2: Hyperparameters of models and algorithms used in our paper.

Algorithm	Hyperparam.	Value
UIClip (JitterWeb pretraining)	Batch Size	128
	Epochs	1
	Learning Rate	$5e-7$
	Weight Decay	0.2
	Adam Beta 1	0.9
	Adam Beta 2	0.98
	Adam Epsilon	$1e-6$
UIClip (JitterWeb pairwise, BetterApp)	Batch Size	256
	Epochs	1
	Learning Rate	$5e-7$
	Weight Decay	0.2
	Adam Beta 1	0.9
	Adam Beta 2	0.98
	Adam Epsilon	$1e-6$
BetterApp DBSCAN	Epsilon	0.1
	Min samples	5
	Metric	Cosine Similarity

B LARGE VISION-LANGUAGE MODEL PROMPT

Below, we provide the prompt that was used to evaluate UI screenshots in our quantitative study. The prompt below is used to evaluate a screenshot of an e-commerce application and contains the same description of CRAP guidelines that we gave to human raters.

```
This image contains two screenshots of user
interfaces stacked horizontally (left and right).
Both UIs can be described by the description:

An e-commerce application interface

Recall the CRAP guidelines for visual design,
which stands for contrast, repetition, alignment,
and proximity.

The C.R.A.P principles, coined by Robin Patricia
Williams in her non-designers' design book, are a
set of guidelines aimed at improving the visual
appeal and effectiveness of graphic designs. These
 principles are essential for creating visually
appealing and user-friendly designs. CRAP stands
for:
Contrast: This principle suggests that elements
that are not the same should be very different so
that they stand out. Using contrast can attract
the viewer's attention and help organize
information. It can be applied through variations
in color, size, typeface, and other visual
elements.
Repetition: Repetition involves repeating some
aspect of the design throughout the entire piece.
This can include the consistent use of colors,
fonts, and logos, which helps to create a cohesive
 and harmonious look. Repetition strengthens a
design by tying together individual elements and
can enhance the overall sense of unity.
Alignment: Every element should have a visual
connection with something else on the page. This
doesn't mean that elements always need to be in a
straight line, but rather that they should be
visually connected in a way that makes the entire
design appear well organized. Proper alignment
eliminates disorder, connects elements, and
creates a visually logical structure.
```

Proximity: Items that relate to each other should be grouped together, which helps in organizing information and reducing clutter. By effectively grouping related elements, the design becomes easier to comprehend, and relationships between elements become clearer to the viewer. Proximity can also help in creating focal points in a design.

Based on these guidelines, provide a response that indicates which UI screenshot is better designed. The first part of your response should contain one of two choices: 'left', 'right.' The second part of your response should contain a comma-separated list of which CRAP principles (if any) are most relevant to your choice. Do not provide explanations, and separate the first and second part of your response with a new line.