electronics

MDPI

Article

UISGPT: Automated Mobile UI Design Smell Detection with Large Language Models

Bo Yang * and Shanping Li

College of Computer Science and Technology, Zhejiang University, Hangzhou 310012, China; shan@zju.edu.cn
* Correspondence: imyb@zju.edu.cn

Abstract: Manual inspection and remediation of guideline violations (UI design smells) is a knowledge-intensive, time-consuming, and context-related task that requires a high level of expertise. This paper proposes `UISGPT`, a novel end-to-end approach for automatically detecting user interface (UI) design smells and explaining each violation of specific design guidelines in natural language. To avoid hallucinations in large language models (LLMs) and achieve interpretable results, UISGPT uses few-shot learning and least-to-most prompting strategies to formalize design guidelines. To prevent the model from exceeding the input window size and for the enhancement of the logic in responses, UISGPT divides design smell detection into the following three subtasks: design guideline formalization, UI component information extraction, and guideline validation. The experimental results show that UISGPT performs effectively in automatically detecting design violations (F1 score of 0.729). In comparison to the latest LLM methods, the design smell reports generated by UISGPT have higher contextual consistency and user ratings.

Keywords: android GUI testing; large language model; prompt engineering

Citation: Yang, B.; Li, S. UISGPT: Automated Mobile UI Design Smell Detection with Large Language Models . *Electronics* **2024**, *13*, 3127. https://doi.org/10.3390/electronics13163127

Academic Editor: George Angelos Papadopoulos

Received: 3 June 2024
Revised: 25 July 2024
Accepted: 6 August 2024
Published: 7 August 2024

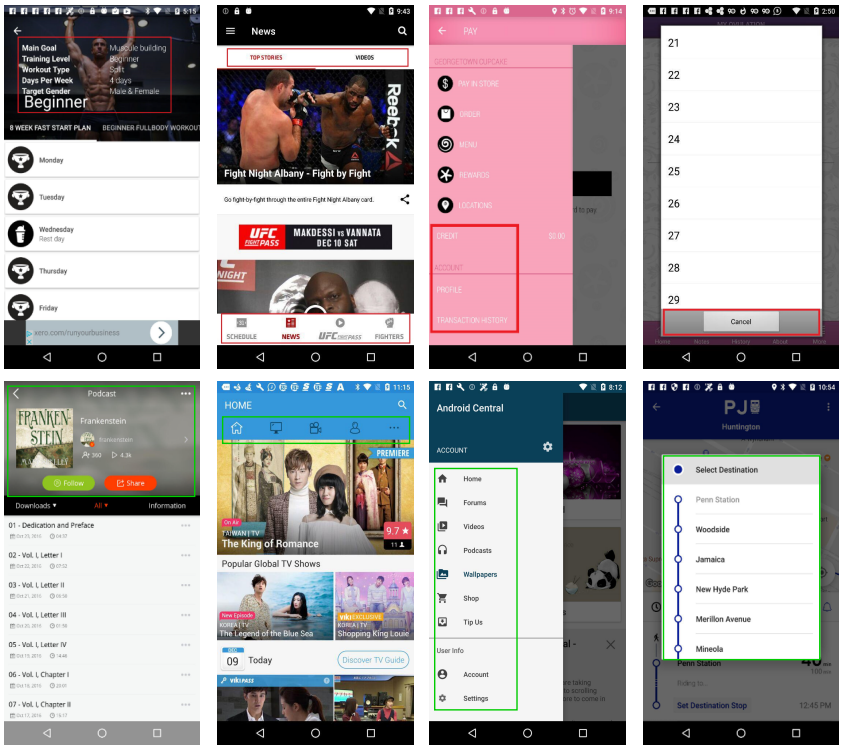
1. Introduction

A graphical user interface (GUI) is a pivotal feature of mobile applications that is essential for user retention and the successful promotion of software [1]. Nevertheless, developing a good GUI can be challenging, even for experienced development teams. For one thing, designers must adhere to numerous design principles [2] to ensure that the final prototype meets standards of consistency, aesthetics, and effectiveness. [3]. Furthermore, some high-level design principles cannot serve as actionable tactics for independent developers.

User interface (UI) design smells in mobile app GUIs are inevitable, revealing violations of design principles and ultimately affecting user experience [4–6]. The concept of design smells is derived from code smells [7]. Figure 1 shows common design smells (highlighted in red boxes). For example, in Figure 1a, text overlaps with the background image, making the header title unclear. In Figure 1b, using both tabs and bottom navigation can confuse users about which tab controls the current content. In Figure 1c, some items in the navigation drawer use icons, while others do not; icons should be used consistently or not at all. In Figure 1d, action buttons in simple dialogs are redundant, since users can make selections and close the dialog by clicking outside it. These design smells violate the design guidelines of the Google Material Design system [8].

Existing methods have significant limitations that hinder their practical application and widespread adoption. First, rule-based heuristic algorithms [4,9] rely on manual creation by experts, with high development and maintenance costs. Second, to ensure that UI design meets specific design guidelines, it is necessary to check multiple sources of information, including component type, position, color, and text content. Learning-based algorithms (e.g., KNN outlier detection [10]) have greater limitations on the number of detectable guidelines. There are only a few studies that focus on detecting design smells, as discussed in Section 2.

Large Language Model (LLM)-enhanced GUI testing methods have significantly reduced the demand for tester expertise and have improved performance. Liu et al. [11] employed an LLM to intelligently generate test cases for semantic text input. Feng et al. [12] examined an LLM's capabilities in wire-framing UI design using a simple language description. Therefore, this paper aims to investigate the potential of LLMs in detecting UI design smells for mobile applications.



(a) Illegible buttons/text (b) Tab and bottom navigation together (c) Icons lost in navigation drawer (d) No button in simple dialog

Figure 1. Comparison of user interface (UI) design smells (**top row**) with good UI practices (**bottom row**), with issues highlighted in red boxes.

Using LLMs for GUI design smell detection faces several challenges. First, LLMs primarily deal with textual information and have a limited context window [13], implying relative weakness in handling lengthy contexts [14]. To address this problem, this paper develops a UI component information extraction module. It can extract style information (coordinates, sizes, colors, etc.) and semantic information (text content, semantic labels, and component types) from the view hierarchy files of UIs. The module also converts UI component information into HTML format, since HTML format is one of the most widely processed data types during the LLM pre-training phase. Secondly, LLMs may potentially introduce "hallucination" phenomena, that is, generate false information that does not correspond to reality [15]. This includes but is not limited to fabricating non-existent design guidelines or incorrect design smells.

To overcome this challenge, this paper attempts to use LLMs for design smell detection. This research proposes `UISGPT`, a novel approach to automatically detect design smells and explain each violation of specific design guidelines in natural language. Since LLMs only take text tokens as input, this paper uses HTML syntax as domain-specific language to convert UI screens into text. This research employed advanced prompt engineering techniques to gain a precise understanding of GUIs without extensive model training. Furthermore, UISGPT leverages least-to-most prompting to elicit UI design knowledge and logical reasoning from LLMs to enhance output consistency and coherence.

The evaluations demonstrate the effectiveness of UISGPT in automatically detecting design violations (the F1 score was improved from 0.456 to 0.729 compared with the

latest LLM method). In the ablation experiments, UISGPT significantly outperformed other prompting methods across GPT-4 [16], PaLM 2 [17], LLaMA 2 [18], and GPT-4 with Vision [16,19]. A user study involving 12 front-end developers confirmed the usefulness and consistency of the generated reports.

In this paper, we make the following contributions:

- To the best of our knowledge, this study is the first to employ LLMs in UI design smell detection with scattered UI design guidelines.
- This paper proposes UISGPT, an end-to-end system that automatically completes the formalization of design guidelines, the extraction of UI component information, and the generation of interpretable smell detection reports after validation.
- This paper conducts experiments on various prompting strategies and different LLMs using real-world datasets. The results reveal that UISGPT's prompting strategies achieve the best performance on GPT-4, with an F1 score of 0.729. Compared to two state-of-the-art LLM-based UI modeling methods, UISGPT not only exceeds their performance but is also confirmed by user studies to excel in usefulness, content consistency, and conciseness.

2. Related Work

This section discusses (1) design violation detection and (2) LLMs for GUI testing.

2.1. Design Smell Detection

To ensure the correctness of the GUI display and to reduce the expenses of manual inspection, researchers have explored automated GUI testing. Alegroth et al. [20] divided GUI automated testing technologies into three chronological generations. Third-generation testing methods [10,21], known as visual GUI testing (VGT), refer to the utilization of computer vision as the core technology for analyzing and interacting with the GUI. However, these methods focus on functional software defects and display issues [22,23]. Early studies, such as that by Issa et al. [24], indicated that visual smells accounted for 16–33% of reported issues across four open-source systems. Despite the significant influence of the GUI on user experience and acceptance, non-functional GUI testing for visual smells has attracted limited attention in the existing literature. This paper focuses more on non-functional GUI testing, i.e., visual smells that do not cause application crashes but have negative impacts on application usability and user experience. The purpose of visual smell detection methods is to enable computers to understand GUIs (i.e., extract GUI features), detect visual smells (i.e., identify abnormal visual features from normal GUIs and locate them), and provide detection reports.

Visual smells in apps are inherently subjective and variable, as they challenge the ability to accommodate diverse preferences. Furthermore, employing checklists to test visual properties is time-consuming and error-prone. Consequently, visual smells present more challenges in detection compared to non-visual smells. Guideline documents such as Apple HIG [25], Google Material Design [8], an Microsoft IDL [26] (also known as design systems) allow developers to use consistent formatting to direct GUI designs. Recent work [10,27,28] has paid considerable attention to visual GUI testing and design smell detection.

Moren et al. [27] introduced GVT, an automated method for identifying GUI design discrepancies in mobile apps. This tool also examines the distribution of various industrial design inconsistencies. The primary distinction between GVT and UISGPT lies in their differing objectives and applications. The GVT processes mockups to check if a mobile app's GUI matches its intended design, helping developers correct deviations from the original concept.

Zhao et al. [10] developed a deep learning-based technique for evaluating GUI animation effects against Google Material Design's "don't" guidelines. They trained a GUI animation feature extractor to capture the spatiotemporal characteristics of GUI animation. Using the k-nearest neighbor (KNN) algorithm, they identified similar animations that violate specific guidelines. Their study focused on nine "don't" guidelines, with 1000 GUI screenshots labeled for each.

Liu et al. [21] proposed the Owleyes tool for text overlap, image loss, flower screen, and five other design smells caused by device compatibility issues. This method first located design smells on GUI images, but the located areas were too large, and the classification model for the five specific design smells had poor generalization ability. The use of synthesized UIs for training instead of real UIs could lead to false positives.

Subsequent tools like Nighthawk [23] and Woodpecker [29] are derived from Owleyes. Nighthawk enhances the end-to-end ResNet model, while Woodpecker correlates the target area with the component source code. There are still problems such as poor generalization ability, false positives caused by data enhancement methods, and limited smell types.

Alotaibi et al. [30] defined a graph-based component-size relationship model for visual defects such as component stacking caused by the scaling of GUIs. This method requires static analysis of GUIs to extract hierarchical structures and generate size relationship graph models. It is less applicable than methods that only require GUI images as input and detect fewer visual defect categories.

Chen et al. [31] followed the ideas of Zhao et al. [10] to detect smells in image-rich applications (such as games). They not only enhanced the realism of synthesized data by injecting defective snippets into the code but also extended the number of types of detectable smells to eight. Similarly, this method requires access to the source code of the application for code injection and has only been verified on game UIs, resulting in generally poor applicability.

Zhang et al. [9] focused on detecting consistency between application programs, GUI policies, and legal regulations. Through static analysis and normalization of expert knowledge, they detected four types of design smells that violate the Google Material Design specifications. However, the effectiveness of the method in practice remains to be verified due to the need for application source code and the limited amount of information involved in these four design smells.

Su et al. [32] proposed dVermin, which detects visual defects in GUIs when magnified. This method has achieved good performance in response to scaling issues but cannot handle other types of design smells.

Previous studies have mainly focused on typesetting (text overlap [21,23,29], size anomaly [9]), images (flower screen [21,31]), layout (distortion caused by proportional enlargement [30,32]), and animation (shadow loss after animation [10]) in specific design defects.

The limitations of existing techniques include the necessity of different models for each task type. Furthermore, participants have reported challenges in interpreting feedback from these models [33] and wished for explanations of design issues in natural language [34]. Additionally, prior research necessitated design prototypes or the crowdsourcing of UI models for benchmarks.

Our research addresses these limitations. UISGPT supports the evaluation of any aspect of any UI using design guidelines from any unstructured text source. Moreover, UISGPT provides explanations for each detected guideline violation in natural language.

2.2. LLMs for GUI Testing

Breakthrough advancements in LLMs have led to research suggesting their use in assisting developers with various tasks, such as code generation [35,36], program repair [37,38], and code summarization [39]. However, in mobile GUI testing, existing random/rule-based [40,41], model-based [42,43], and learning-based methods [44] often fail to capture the semantic information of GUI pages and typically fall short of achieving comprehensive coverage [45,46].

Recent research [11,45,47] has focused on generating test inputs for mobile applications. Liu et al. [11] utilized LLMs to automatically generate semantic input text based on the GUI environment. GPTDroid [45] views input generation for mobile GUI testing as a Q&A task, using LLMs to interact with applications and generate test scripts. The GPTDroid method extracts both static and dynamic contexts of GUI pages and designs prompts for LLMs to better understand the GUI pages and the entire testing process. Zimmermann et al. [48]

leveraged LLMs to interpret natural language test cases and programmatically navigate the application under test. Yu et al. [49] investigated the capabilities of LLMs in generating and migrating mobile application test scripts. Taeb et al. [50] combined LLMs with UI element detection models to convert manual accessibility testing instructions into replayable, navigable videos, highlighting accessibility issues.

Some recent studies [13,51,52] have employed conversational agents via LLMs to simulate real user interaction with GUIs, and Duan et al. [51] enhanced the Figma design tool plugins with an LLM to make suggestions and optimize design prototypes. Vu et al. [52] empowered a virtual assistant with an LLM, enhancing the efficiency of mobile device virtual assistants in interpreting user commands. Wang et al. [13] classified and modeled UI dialogue interaction scenarios and designed a general mobile UI dialogue prompt technique. Liu et al. [53] used GPT-3 to simulate human testers interacting with GUIs. Their system has a broader coverage and detects more flaws than the existing baseline. Wang et al. [54] completed a comprehensive literature review of software testing with LLMs. They analyzed various studies using LLMs for unit test generation, test output validation, test input generation, bug analysis, and correction of identified bugs in code. These studies indicate the enormous potential of LLMs in the field of human–computer interaction and software testing. However, these studies did not apply LLMs as evaluation tools for mobile UIs.

3. Approach

This paper introduces an automated approach for the extraction of UI components and the detection of UI design smells according to design guidelines. Figure 2 illustrates an overview of our methodology, named UISGPT, which is sectioned into the following three primary stages: (1) During the guideline formalization phase, a specialized LLM-based tool is utilized to convert UI design guidelines from their natural language format into formal expressions suitable for analysis. This includes the parsing and identification of pertinent UI properties. (2) The component information extraction phase involves another LLM-based tool tasked with extracting atomic component information and respective values from UI designs, as encoded in HTML. (3) The final stage, validation, employs a tool to assess the UI's component attributes against predefined design guidelines for individual components. This comparison helps in pinpointing UI design violations, ultimately generating a detection report.

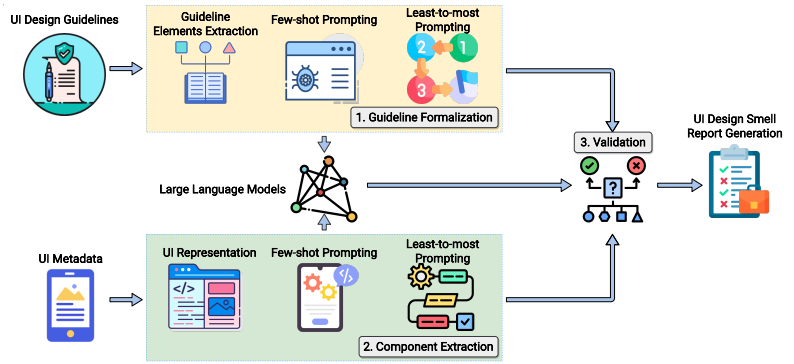


Figure 2. The system architecture of UISGPT.

3.1. Guideline Formalization

The first stage of UISGPT is to identify and extract the component property entities, as well as synthesize formalized guidelines from the textual design guidelines. Specifically, UISGPT employs an LLM pre-trained on a large-scale corpus to enable a deep comprehension of the component property entities referenced within the design guideline text.

To enhance the LLM's ability to discern component property entities and to transform guideline text into formalized instructions, a dictionary of these component property entities is applied, which was meticulously compiled from the corpus. Alongside this

dictionary, examples of input and output are provided, utilizing least-to-most prompting as an inferential technique to output the formalized guidelines.

Examples of the original design guidelines and their conversion into formalized first-order logic expressions are shown in Figure 3. By engaging in several tailored learning iterations within this framework, UISGPT adopts a similar prompting strategy to guide the LLM in extracting component property entities and formalized guidelines from the design guidelines.

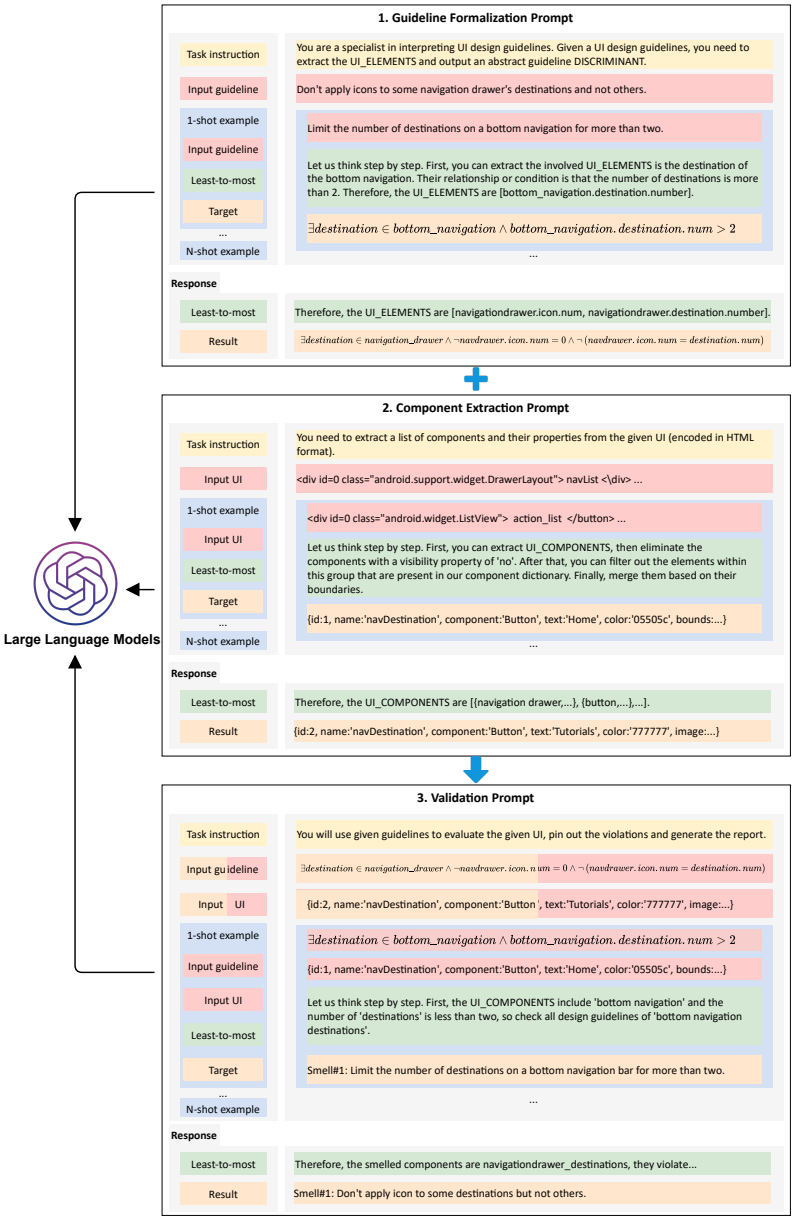


Figure 3. Overview of UISGPT with an example illustrating the proposed prompt structure. The prompt text is color-coded to denote its corresponding step within the prompt sequence. A prompt starts with task instruction, which describes the task. Following the instruction, one or more task examples pertinent to the target tasks is provided, designed to offer a practical understanding of the task requirements. Each example consists of an input design guideline, followed by a least-to-most thought (if applicable), leading to a task-specific output. These examples are arranged sequentially, each appended to the conclusion of its predecessor. Then, the prompt is fed as input to the large language model (LLM). The three modules, although they operate independently, utilize similar structures for their prompts. Specifically, the validation module combines the outputs of the rule formalization and component extraction modules as input.

3.1.1. Guideline Element Extraction

To effectively prompt an LLM to detect smells based on design guidelines, UISGPT needs to formalize these design guidelines, with an emphasis on specific component types and attributes such as text, size, and color. Subsequently, UISGPT extracts violation conditions from the guidelines. These conditions are defined as first-order logic, which relates closely to the detailed information of the design components. The variables, functions, predicate symbols (also known as relation symbols), and arity are all identified and extracted by the LLM.

To ensure the precision of the formalization, UISGPT adds two foundational limitations. First, variables must be expressed in the form of <component.attribute>, aligning the violation conditions directly with the component information. Second, non-logical predicate symbols (other than symbols such as greater than, less than, belongs to, etc.), such as "not allowed to appear simultaneously", must be defined to ascertain their explicit meanings. This clarity is pivotal for the LLM to leverage these symbols effectively in the logical reasoning process.

Through these steps, the structured prompt enables the LLM to parse design guidelines correctly, maintaining high standards of formalization quality.

3.1.2. Least-to-Most Prompting

In the process of formalizing the design guidelines, UISGPT adopts the least-to-most prompting strategy [55]. This technique enables complex reasoning to be broken down through smaller intermediate reasoning steps. Specifically, the least-to-most prompting process first decomposes the problem into sub-problems, then solves them one by one. Similar to the Chain-of-Thought prompting (CoT prompting) process [56], the problem to be solved needs to be decomposed into a set of sub-problems built upon each other. The difference is that in the least-to-most prompting process, the solution to a previous sub-problem is input into the prompt to try to solve the next problem. This strategy is particularly useful for UISGPT in the formalization of design guidelines [56,57], as it allows the LLM to gradually build up the formal representation from the most basic aspects to the most complex ones.

In the prompt, "Let us think step by step" is added to generate the reasoning steps progressively. In the demonstration examples (i.e., one-shot example in Section 3.1.3), the purposes of each reasoning step and the corresponding demonstration output are shown to the LLM, as illustrated in the green part of Figure 3. The least-to-most prompting strategy in UISGPT consists of two main stages. First, the prompt instructs the LLM to extract knowledge entities from the design guidelines presented in natural language. These entities include primary component types, component attributes (which we emphasize as a key parameter with *UI_ELEMENTS* in the prompt), and their logical relationships. Once these knowledge entities have been delineated, the prompt directs the LLM to transform the extracted information into a formal representation using first-order logic structures (which we emphasize as a key parameter with *DISCRIMINANT* in the prompt).

By following this structured approach, UISGPT ensures a coherent and systematic transformation of abstract design guidelines into an explicit logical framework, laying the groundwork for automated reasoning and validation of design guidelines.

3.1.3. Few-Shot Prompting

Few-shot prompting [58] is a technique that leverages a small number of examples to guide the LLM in solving a complex task. Few-shot examples that follow a structure and format similar to those of the target task provide significant benefits [59].

In the context of UISGPT, few-shot prompting facilitates the extraction of knowledge entities and the formalization of design guidelines. Specifically, as shown in the blue part of Figure 3, UISGPT aims to formalize the guideline "don't apply icons to some navigation drawer's destinations and not others". We expect UISGPT to first extract the involved *UI_ELEMENTS* from the design guideline text, then formalize these entities in a logical rep-

resentation. To achieve this, UISGPT uses a one-shot example. For example, given the input "limit the number of destinations on a bottom navigation for more than two", it follows a least-to-most prompting process to extract "bottom_navigation.destination.number". Then, it outputs a guideline (DISCRIMINANT), as shown in the first orange part of Figure 3.

Given the above example input and output, UISGPT is expected to perform a similar procedure and produce a similar output for new input.

3.1.4. Prompt Construction

Figure 3 shows an example of a "guideline formalization prompt". In the guideline element extraction stage, UISGPT uses few-shot prompting, that is, <Design guideline text> + <Example input> + <Example output>. In the formalization stage, UISGPT uses few-shot prompting and least-to-most prompting, i.e., <Guideline elements> + <Example input> + <Least-to-most> + <Example output>. Due to the advantages of few-shot learning and least-to-most reasoning, LLMs consistently generate a list to represent the extracted guidelines in the same format as the example output, which can be inferred using regular expressions.

3.2. Component Extraction

UISGPT is designed to process UI screenshots alongside Android view hierarchy data, which are input in JSON format. Figure 4 provides an illustrative example of a UI and a segment of its corresponding view hierarchy. The overall organizational structure is a tree, with interface components such as icons and text represented as leaves and clusters of elements or subgroups functioning as intermediate nodes. In this hierarchical tree, each node carries rich semantic information, including text identifiers, element names, and categories. Additionally, nodes contain stylistic details on how to present this information, such as positional coordinates, color, and font style.

To prompt the LLMs to detect design smells among GUIs, UISGPT needs to input UI information contained in JSON hierarchy files into the LLM. To prevent redundancy in the source file from exceeding the input window size and as previous studies [13,60,61] have shown that using an input format consistent with the training data of LLMs can enhance their performance, UISGPT converts the view hierarchy from JSON format into HTML syntax.

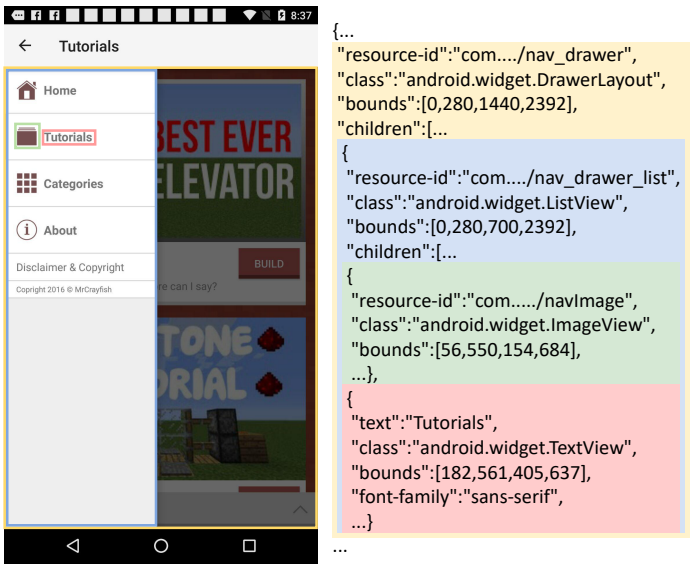


Figure 4. An example of a UI with a navigation drawer and its corresponding fragment of the view hierarchy. Each attribute list on the right is marked with the same color for the background and border box as the corresponding GUI element on the left.

3.2.1. UI Representation

UISGPT represents a mobile UI by converting its view hierarchy data into HTML syntax, which it inputs to the LLM. HTML is a naturally fitting format, as HTML's primary function as a markup language is already aimed at structuring web UIs. The transformation process involves systematically combing through the view hierarchy via a depth-first search. The goal is not a perfect one-to-one conversion because the view hierarchy was not created to be encoded in HTML. This phase is intended to fine tune the model's ability to comprehend and generate UI representations with greater accuracy.

In the beginning, UISGPT employs an intuitive heuristic methodology to equate classes within the view hierarchy to corresponding HTML tags that offer similar functionality. For instance, the TextView class is matched with the <p> tag, as both are used to display text. In a comparable vein, the Button class is paired with the <button> tag, and the ImageView is linked to the <img> tag. When it comes to more specialized classes present within the view hierarchy, such as <EditText>, <CheckBox>, and <RadioButton>, UISGPT opts for a blend of <input> and <label> tags to stay true to standard HTML syntax.

This research focuses primarily on the most commonly used classes to maintain simplicity. Classes not covered by specific mappings, like `<VideoView>`, are generally assigned to the `<div>` tag. Then, the text attributes of a component are inserted between the opening and closing HTML tags, adhering to the standard syntax of texts within HTML. The `resource_id` property often contains contextual information regarding a component's role or intention. For example, a `resource_id` named "`submit_btn`" suggests its function as a submission button. This identifier is inserted in the "class" attributes in the resultant HTML, providing contextual clues to the models for better screen understanding. For components like ImageView, UISGPT further includes `content_desc` within the "alt" attribute of the HTML tag, which enriches the GUI screen's accessibility features. Finally, according to the traversing order, UISGPT assigns distinctive numeric indexes to each element as the "id" attribute. This systematic approach ensures that each UI component is translated into an HTML representation that is reflective of its function and context within the view hierarchy.

3.2.2. Few-Shot Prompting

This stage is similar to the earlier stage described in Section 3.1.3, where writing is performed independently first, followed by discussions to identify the most representative step-by-step reasoning. A specific example of this reasoning process can be found in Figure 3.

3.2.3. Least-to-Most Prompting

LLMs sometimes face difficulty in comprehending and providing precise output when dealing with complex prompts that demand logical reasoning. UISGPT also uses the least-to-most prompting strategy, the principles and processes of which are described in Section 3.1.2. Figure 3 shows an illustrative example of a "component extraction prompt" and how to employ this strategy for the specific task of extracting button characterizations from a UI component.

3.2.4. Prompt Construction

Figure 3 shows an example of a "component extraction prompt". UISGPT combines the aforementioned information in Figure 3 as the input prompt, i.e., (<Available components> + <Component attribute + <Example input> + <Least-to-Most> + <Example output>). Subsequently, UISGPT uses the UI's HTML as a test prompt input and queries for the list of components. Leveraging the advantages of few-shot learning and least-to-most reasoning, the LLM consistently generates a list in the same format as the example output to display the extracted components. This can be inferred using regular expressions.

3.3. Validation

Figure 3 shows an example of a "validation prompt". In the validation stage, few-shot prompting and least-to-most prompting are employed, i.e., (<UI Representation> +<Guideline list> + <Example input> + <Least-to-most> + <Example output>). Due to the advantages of few-shot learning and least-to-most reasoning, LLMs consistently generate a list, specifying the the UI components that violate the design guidelines and provide the logical reasoning process.

3.4. Implementation

UISGPT detects UI design smells using APIs provided by LLMs, with parameter adjustments performed to optimize performance. The reason for selecting pre-trained LLMs is their excellent performance in natural language understanding and logical reasoning, benefiting from vast datasets and numerous parameters. Moreover, due to the substantial manual effort required, the research community currently lacks a smaller, more specialized training dataset for UI design smells.

For model selection, UISGPT chooses OpenAI's GPT-4-turbo model [16], one of OpenAI's high-performance models. Compared to the latest GPT-4o, some community feedback [62] indicates that GPT-4-turbo produces more stable outputs, more consistently follows instructions, and performs better in some programming tasks. Therefore, for tasks requiring high precision and adherence to specific instructions, such as UI design smell detection, UISGPT opts for GPT-4-turbo.

In parameter settings, UISGPT modifies two key parameters to better suit the smell detection task, namely temperature and maximum response length (also known as maximum token length). Temperature controls the randomness of the output, with lower temperatures (e.g., 0) producing more fixed and consistent outputs. UISGPT sets the temperature to 0.65 to balance determinism and creativity. For large models, outputs are expressed as tokens rather than just words. UISGPT increases the maximum response length from the default 256 to the model's allowed maximum of 4096 to extend the LLM's output.

4. Evaluation

This section benchmarks various LLMs and prompting methods to evaluate the performance of the UISGPT system. Our goal was to answer the following three research questions (RQs):

1. **RQ1:** How does the performance of UISGPT vary with different prompting strategies and LLMs?
2. **RQ2:** How accurate and effective is UISGPT in generating explanations compared to baselines?
3. **RQ3:** How useful are the reports generated by UISGPT in practice?

4.1. Experimental Setup

4.1.1. Dataset

This paper uses the open-source, real-world Rico Android app dataset [63] as the data source. To select a subset of GUIs from over 64,000 for manual analysis, this paper applies the following criteria to filter applications: (1) more than 500,000 downloads on the Google Play Store [64], indicating popularity; (2) a development history of over three years, suggesting stability; and (3) native Android UI, as the paper uses the Material Design component design guidelines for case studies, which primarily target native UI components. Manually inspecting all UIs would require a significant amount of time and effort. To reflect the true distribution, this paper adopts a statistical sampling method [65]. At a confidence level of 95%, this study sets the error rate at 0.05 to calculate the minimum number (*MIN*) of UIs that need to be reviewed. For these samples, this paper summarizes their functional categories in the Google Play Store. As shown in Figure 5, the sampled UIs cover 26 categories, with the top three being entertainment, communication, and shopping. Subsequently, two authors manually inspect all sample UIs to annotate whether they

contain design smells and which Material Design guidelines [4] are violated. Each UI receives two independent annotation results, and any discrepancy is discussed to reach a consensus. The number of components included in the sample UIs and the number of violated design guidelines are shown in Table 1. These 382 sampled real-world UIs are used for quantitative evaluation.

Table 1. The number of components (#Comp_Num), the number of violated design guidelines (#GL), and the number of design smells reported by UISGPT (#RDS) on GPT-4. The suffix "_R" represents the sample UI in the real world, and the suffix "_P" represents the UI prototype. "-" indicates the design smell is not injected into the UI prototype of this component type.

Component	#CompNum_R	#GL_R	#RDS_R	#CompNum_P	#GL_P	#RDS_P
Button	811	6	128	272	1	23
Top Bar	179	4	25	225	-	-
List	97	0	7	80	3	10
Text Field	99	2	10	182	2	12
Tab	66	5	8	34	-	-
Dialog	31	5	11	33	-	-
Banner	27	6	9	31	3	9
Navigation Drawer	30	6	12	70	1	7
Bottom Navigation	26	5	6	16	-	-
Floating Action Button	21	4	7	59	5	17
Bottom Bar	13	3	5	66	3	9
Total	**1400**	**46**	**228**	**1068**	**18**	**87**

This study collects 413 UI design prototypes from the popular Figma design tool and community [66]. These prototypes, sourced from well-known design communities like InVisionApp [67] and SketchRepo [68], showcase the best practices in material design. Manual inspection confirms that these prototypes are free of design smells. To test UISGPT's ability, this paper injects 26 UI design smells not found in the real-world UIs from Rico into the editable prototypes. This "injection" method is used to assess techniques for detecting UI design smells [10] and other program errors [69].

Specifically, this paper edits the UI design prototypes in the dataset. For each design guideline, the paper carefully selects 2 to 4 UI design prototypes that showcase the primary components of the guidelines. We then modify these primary components to replicate the violations, as illustrated in the Material Design guidelines. These adjustments included changing text content and styles, adding appropriate components or icons, and altering the size or position of components. As a result of this process, we create 80 UI design prototypes that violate 26 specific design guidelines. To evaluate the accuracy of UI design prototype assessments, we combine these 80 modified prototypes with the original 413 prototypes.

For the usefulness assessment, we need to select a small number of samples for qualitative analysis. These UIs should include as many component types and design smell types as possible. This paper categorizes design smells into easy, medium, and hard based on complexity. Easy smells involve one attribute or design concept within a component, medium smells involve multiple component attributes or design concepts within a component, and hard smells involve relationships between multiple components or conditional judgments of multiple component attributes or design concepts within a component. Finally, the paper selects 5 easy, 5 medium, and 5 hard design smells from nine popular components for qualitative analysis, as shown in Table 2.

Table 2. Examples used for the usefulness evaluation and the results of GPT-4V and GPT-4V in detecting design violations with prompt engineering.

No.	Example Description	Difficulty	GPT-4V	GPT-4VP
1	Using too long a text label in a text button	Easy	✗	✓
2	An outlined button's width should not be narrower than the button's text length	Medium	✗	✓
3	Truncating text in a top bar	Easy	✗	✗
4	Wrapping text in a regular top bar	Medium	✗	✗
5	Applying icons to some destinations but not others in a navigation drawer	Hard	✗	✗
6	Shrinking text size in a navigation drawer	Medium	✗	✗
7	Mixing tabs that contain only text with tabs that contain only icons	Easy	✗	✓
8	Truncating labels in a tab	Easy	✗	✗
9	Using a bottom navigation and tab together	Easy	✗	✓
10	Using a bottom navigation bar for fewer than three destinations	Medium	✗	✗
11	Using dialog titles that pose an ambiguous question	Hard	✗	✗
12	Using a single prominent button in a banner	Hard	✗	✗
13	Displaying multiple FABs on a single screen	Medium	✗	✓
14	Including fewer than two options in a speed dial of a FAB	Hard	✗	✗
15	Using a primary color as the background of text fields	Hard	✗	✗

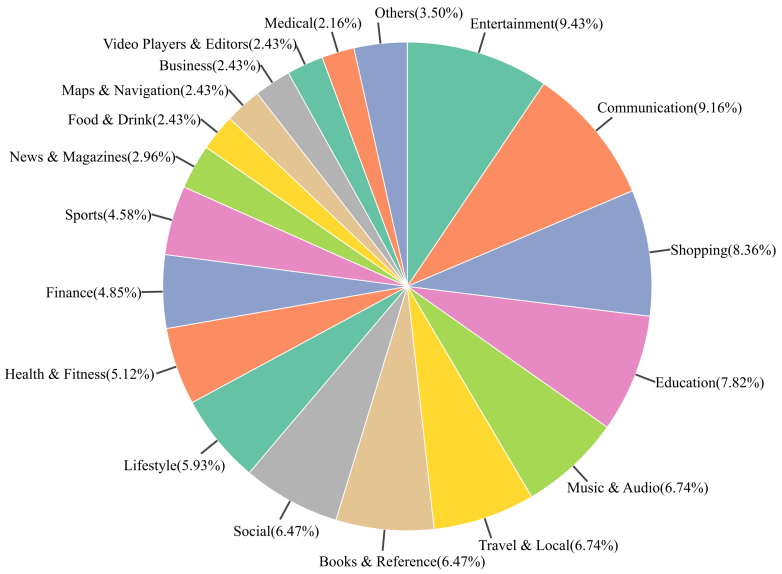


Figure 5. Distribution of sampled UIs from Rico by functional category in the Google Play Store [63].

4.1.2. Metrics

This paper uses precision, recall, and F1 score as metrics. As shown in Table 1, 382 sampled UIs have 1400 components and 46 violated guidelines based on human inspection. The number of reported design smells is marked as #RDS. These #RDS are manually checked to identify real design smells (i.e., true positive, #TP). Precision is the percentage of #TP among all #RS. False negative (#FN) is the number of actual design smells in the sampled UIs that are not reported. Recall is #TP/(#TP+#FN). The F1 score is 2*(precision*recall)/(precision + recall).

4.1.3. Participants

To evaluate the quality of smell reports, 12 front-end participants are recruited from different backgrounds, comprising 4 PhD students, 5 developers from IT companies, and 3 experts in the mobile app field. Their ages range from 25 to 35. Two participants have 1–2 years of development experience, five participants have 2–6 years of development experience, and five participants have more than 6 years of development experience, with the longest development experience among participants being about 15 years.

4.1.4. Prompting Strategies in RQ1

End-to-end prompting We start with the most simple direct prompting baseline with the following instruction:

> *As a professional Android UI reviewer, you are well-versed in Google Material Design guidelines. You will be given a UI screenshot, and you need to evaluate this UI using Material Design guidelines. You should return a list of UI components that do not conform to guidelines and point out how certain components of the UI violate the design guidelines. This should be done solely based on the provided design guidelines webpage. Do not rely on your own standards. Do not hallucinate any dependencies on external resources. Do not consider dynamic interactions and animation guidelines. Pay attention to the size, text, location, color, and overall layout of all static elements. The response must include the list of relevant UI components, the expected conforming design guidelines, the gap between the design and expected guidelines, and advice on fixing the gap.*

With only guideline formalization prompting The aforementioned prompt requires the model to accomplish all the tasks at once, i.e., understand design guidelines and use them to validate the UI layout. In reality, developers often want to verify whether a component complies with the design guidelines. They need expertise in design guidelines and professional knowledge of specific components. To reflect this situation, we explore a method using only design guideline formalization modules, i.e., first extracting a formal representation of the design guidelines, then attaching these formalized design guidelines to the prompts. In this case, we mitigate the difficulty of LLMs in understanding the design guidelines, thus allowing the model to focus more on guideline validation.

With only component extraction prompting Similarly, to reflect situations in which designers lack professional knowledge to convert designs into code implementations, this paper also explores using only modules for UI encoding and component extraction, then attaching the encoded HTML code to the prompts. In this case, the model can focus more on guideline validation.

4.1.5. Baselines

RQ1: To explore the performance of other state-of-the-art LLMs in the smell detection task, this paper uses proprietary PaLM 2 (Chat Bison 32k) [17], GPT-4(gpt-4-turbo-2024-04-09) [16], and open-source LLaMA 2 (Llama-2-70B-chat) [18] as baselines.

To compare the above raw-text models with multimodal models, we also choose GPT-4 with Vision [16,19] (GPT-4V-8K-2024-02-28), a state-of-the-art multimodal LLM with vision capabilities, as a baseline. As multimodal models only use UI screenshots as input, the metric is whether GPT-4V can handle the multimodal UI violation detection examples listed in Table 2. The prompt engineering formulates the role of GPT (professional UI reviewer) and the task (analyze UI and detect guideline violations), and the input includes not only UI screens but also the specific design guideline text to be checked. In GPT-4V without prompt engineering, the design guideline text is replaced by the specific documentation link.

RQ2: To evaluate the accuracy and effectiveness of the explanations generated by UIS-GPT, our experiments select LLM4mobile [13] and the method proposed by Duan et al. [51] as baselines.

LLM4mobile [13] introduces a prompting technique tailored to mobile UIs. This method has been shown to effectively support tasks such as screen summarization and

Q&A, making it a key benchmark [70] for the assessment of LLM performance in mobile UI contexts.

The method proposed by Duan et al. [51] employs an LLM to assess the usability of heuristic design principles on UI mockups and generates improvement feedback. Comparing this method with UISGPT highlights the differences between heuristic principle feedback and design smell detection reports.

Both systems represent state-of-the-art approaches in using LLMs for UI-related tasks, aligning closely with UISGPT's goals. LLM4mobile [13] focuses on mobile UI modeling, which is directly relevant to UISGPT's application context. The method proposed by Duan et al. [51] emphasizes the generation of usability feedback based on heuristic design principles, paralleling UISGPT's objective of producing explainable design smell reports. Comparing UISGPT with these baselines allows for a comprehensive evaluation of its ability to generate accurate and useful design smell reports.

4.2. RQ1: Quantitative Evaluation—Performance Study

This section evaluates the results of different prompting strategies (Section 4.1.4) on the baseline LLMs (Section 4.1.5-RQ1). The 382 sampled UIs are fed into GPT-4, PaLM 2, and LLaMA 2 to obtain lists of their output results. Three participants with at least three years of front-end design and development experience are recruited to independently annotate the results. Cohen's kappa [71] is applied to assess the inter-rater agreement, which is 0.949, indicating near-perfect consensus. For any instance where the two participants assign different labels, a third participant is assigned to supply an additional label to reconcile the conflict using a majority-vote strategy. Based on the final annotations, we calculate the three evaluation metrics for the models, namely precision, recall, and F1 score. All experiments are conducted on a computer with an Intel Xeon Gold 6226 2.7 GHz CPU, 64 GB memory, and four 8 GB NVIDIA GeForce RTX 2080 Ti GPUs.

Result: We find that PaLM 2 and LLaMA 2 all achieve considerably worse performance than GPT-4, as shown in Table 3 (bottom). The rest of Table 3 shows that UISGPT significantly outperforms other prompting methods across GPT-4, PaLM 2, and LLaMA 2.

Table 3. The performance of GPT-4, PaLM 2, and LLaMA 2 in UI smell detection with different modules.

Prompting	GPT-4		PaLM 2		LLaMA 2	
	Precision	Recall	Precision	Recall	Precision	Recall
End-to-End	0.178	0.139	0.164	0.193	0.159	0.122
With guideline formalization	0.420	0.352	0.467	0.303	0.435	0.285
With component extraction	0.554	0.503	0.539	0.451	0.517	0.313
UISGPT	**0.796**	**0.673**	0.779	0.605	0.672	0.476

Due to a lack of intermediate ideas and data, the end-to-end method often fails to understand the corresponding relationship between the components and design guidelines in UIs. It can only detect a few simple design guidelines, and the results contain many irrelevant pieces of information that need further filtering. By characterizing design guidelines, the overall performance is improved, but due to the complexity of the UI structure, the guideline formalization module still cannot correspond to UI components. The LLMs are good at handling natural language text, and the use of UI component extraction improves performance on end-to-end models. Combining guideline formalization with UI component extraction achieves the best performance on GPT-4, with a precision of 0.796 and a recall of 0.673. Situations that cannot be handled include cases where components are too large or too small, where the former results from a lack of structural blocks during HTML conversion and the latter results from possible neglect of a few entries under large windows.

In addition, Table 2 reveals the performances of GPT-4V and GPT-4V with prompt engineering. UISGPT can detect the corresponding design smells in these example UIs. GPT-4V uses design documentation links as input and cannot detect any violations. Even

though it provides some semantic explanations about the UI screen and design principles of UI components, most of the responses are unhelpful for violation detection tasks. GPT-4V with prompt engineering succeeds in some moderate and easy detection examples (e.g., examples 1, 2, 7, and 9). For the rest of the examples, its response indicates that the violation cannot be detected and requires further clarification. Moreover, although it completes the detection task in some instances (such as example 2), its performance varies in repeated experiments, and sometimes violations cannot be detected.

In conclusion, GPT-4V still has many shortcomings in detecting design violations. The prompting strategy and task framework of UISGPT also surpass those of other prompting strategies. UISGPT achieves the best performance on GPT-4, with a precision of 0.796 and a recall of 0.673.

4.3. RQ2: Quantitative Evaluation—Comparison with Baselines

This section compares the performance of UISGPT with that of other UI modeling methods (Section 4.1.5-RQ2) based on LLMs. Similar to the experimental steps described in Section 4.2, our experiments also validate 382 sampled UIs with participants annotating the results. Cohen's kappa between the two annotators is 0.953 (almost perfect agreement). Finally, we calculate the precision, recall, and F1 score for the models, as shown in Table 4.

Table 4. The performances of UISGPT, LLM4mobile [13], and the method proposed by Duan et al. [51] on GPT-4.

Model	Real-World UI			UI Prototype		
	Precision	**Recall**	**F1-Score**	**Precision**	**Recall**	**F1-Score**
LLM4mobile [13]	0.525	0.355	0.424	0.569	0.471	0.515
Duan et al. [51]	0.585	0.375	0.456	0.648	0.516	0.575
UISGPT	**0.796**	**0.673**	**0.729**	**0.837**	**0.770**	**0.802**

Result: Table 4 indicates that the performance of UISGPT is superior to that of the method proposed by Duan et al. [51] and better than that of LLM4mobile [13]. LLM4mobile [13] focuses more on screen summarization and screen question answering for UI modeling than tasks related to design violations. In contrast with the method proposed by Duan et al. [51], which introduces heuristic methods for evaluation, UISGPT still increases the F1 score from 0.456 to 0.729. The main reason for the performance improvement is that UISGPT formalizes the design guidelines, extracts the UI components and component properties needed for comparison, and asks the LLM to provide intermediate reasoning results through the thought chain.

For UI prototypes, UISGPT detects 67 out of 80 injected design smells and incorrectly reports 20 additional smells, achieving F1 = 0.802. Overall, the performance on design prototypes is better than that on real-world UIs, which may be due to the fact that design prototypes are simpler in terms of visual communication.

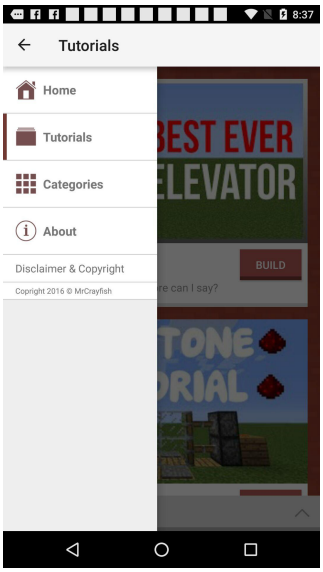
To compare the interpretability of the generated results, 12 participants are invited to score the output reports for the 15 UIs shown in Table 2. We pre-calculate the evaluation results of UISGPT and the two baselines for all 15 UIs to ensure that all participants see the same violation detection reports, enabling us to compute inter-rater agreements. Figure 6 displays an example input and output for a navigation drawer.

Each participant is presented with the same 15 UIs and has one week to rate the 45 detection reports. Their ratings are based on the entire report rather than a single violation because reading the entire report is more similar to the practical development process.

On average, participants spend 6.2 h to complete this task. For each report, participants are asked to select ratings for usefulness, content adequacy, and conciseness (on a 5-point Likert scale, with 1 being the least useful/adequate/concise and 5 being the most useful/adequate/concise). Usefulness measures the extent to which the generated explanations are useful for answering questions. Content adequacy evaluates the degree to

which the generated explanations adequately answer questions. Conciseness measures the extent to which the explanations contain unnecessary information.

Result: Figure 7 summarizes the users' ratings of the three features of explanations. The design smell reports generated by UISGPT outperform the feedback from the two baselines in terms of usefulness, content adequacy, and conciseness. Compared to the method proposed by Duan et al. [51], UISGPT achieves improvements of over 32% in usefulness and conciseness. Both baseline models perform poorly in terms of conciseness, which is consistent with their lower recall rates observed in the RQ1 performance evaluation (Section 4.2). In contrast, UISGPT's responses benefit from few-shot learning and are more focused on the design guidelines themselves rather than aesthetic principles. Seven participants note the superiority of UISGPT in pinpointing main components and design guidelines, making them more understandable to humans. Five participants remark that the strength of UISGPT lies in its more precise conclusions, with no redundant introductions of aesthetic principles.



LLM4mobile: ...Identified Design Violations:
1. Consistency and Standards:
- The navigation drawer icons for "Home", "Tutorials", "Categories", and "About" are consistent in style and color, but the top bar icons (back arrow, Facebook, and three-dot menu) have different styles. The inconsistency in iconography can confuse users.
- The header "Tutorials" should have a consistent alignment and spacing. The back arrow and the title "Tutorials" are misaligned, which affects the visual consistency.
2. Visibility of System Status:
- The battery and time indicators at the top are visible, which is good. However, the Facebook and other icons at the top left are not clearly explained, potentially leading to confusion about their function.
3. Aesthetic and Minimalist Design:
- The UI appears cluttered, especially with the Facebook and menu icons at the top. Simplifying the header could enhance visual appeal.
- The use of color in the navigation drawer is minimal and consistent, but the background of the screen peeking from behind the drawer has bright colors that clash with the drawer's minimalist design.
4. Error Prevention:...
Areas for Improvement:
- Consistency: Align the header elements and ensure a consistent style for all icons. (Smell#2)
...

Duan et al.: ...Guideline Violations:
1. Clarity and Simplicity:...
2. Consistency:
- Issue: The design lacks consistency in the alignment and spacing of icons and text. The "Home," "Tutorials," and "Categories" icons differ slightly in size and alignment from the "About" icon.
- Improvement: Ensure uniform size and alignment for all icons and text elements. (Smell#2)
3. Feedback:...
4. Accessibility: (Smell#1)
- Issue: The small font size and icons may be difficult for users with visual impairments to read.
- Improvement: Increase font size and ensure high contrast between text/icons and the background. Also, ensure that the UI is compatible with screen readers.
5. Visibility of System Status... 6. Navigation...
...

UISGPT: ...Material Design Guidelines Violations:
Violated Guideline#1:
Don't apply icon to some destinations but not others.
In the list of <navigation drawer>, destinations like "Home", "Tutorials", "Categories", and "About" have <ImageView>, which means they have icons, whereas "Disclaimer&Copyright" and "Copright 2016 c MrCrayfish" only have <TextView>.

Violated Guideline#2:
Don't shrink text size in a navigation drawer.
The <TextView> of <navigation drawer> have different size and alignment, that is, "Disclaimer&Copyright" and "Copright 2016 c MrCrayfish" are shrinked compared to "Home", "Tutorials", "Categories", and "About".

Violated Guideline#3:
The navigation drawer should not overlay the app bar.
In the screenshot, the <navigation drawer> overlays the <app bar>, which is not recommended. The <app bar> should remain visible to provide context to the user.
...

Figure 6. Example outputs from UISGPT, LLM4mobile [13], and the method proposed by Duan et al. [51] with a navigation drawer as input. The correct detection results are smell#1, "Don't shrink text size in a navigation drawer"; smell#2, "Don't apply icon to some destinations but not others"; and smell#3, "Don't use dividers to separate individual destinations". LLM4mobile [13] and the method proposed by Duan et al. [51] detect smell#1. The method proposed by Duan et al. [51] partially detects smell#2, pointing out the need for consistent styling, but does not specify that this is due to the missing icons, whereas UISGPT fully detects both smell#1 and smell#2. Smell#3 is not detected by any method because the dividers are not included in the view hierarchy file.

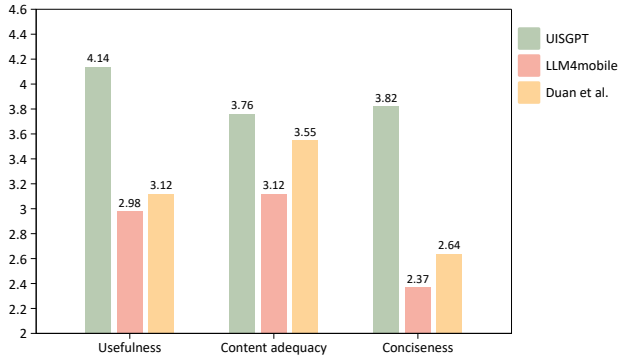


Figure 7. Average usefulness, content adequacy, and conciseness ratings of explanations across UISGPT, LLM4mobile [13], and the method proposed by Duan et al. [51].

4.4. RQ3: Usefulness Evaluation

In Section 4.3, participants were asked to rate the reports generated by UISGPT for the tasks listed in Table 2 using a five-point Likert scale.

Result: Figure 8 summarizes the participants' ratings of the three features of explanations. Twelve participants score an average of 4.14 for the usefulness of the generated explanations, indicating that participants believe UISGPT can effectively help answer design-related questions. The average scores for content adequacy and conciseness are 3.76 and 3.82, respectively. To verify the significance of the differences among these categories, we conduct a one-way analysis of variance [72]. The results ($F(2, 177) = 11.5, p < 0.0001, \alpha = 0.05$) indicate that there are statistically significant differences in user ratings across the different difficulty levels. A post hoc Tukey's honestly significant difference test [73] reveals that there are statistically significant differences between all three groups ($p < 0.05$). The difference between easy and hard is the largest and most significant ($p < 0.0001$).

Three participants consider the generated information used for inspiration in design to be less concise because they consider the information not helpful for answering the question. But in practice, "unnecessary" information is still component-related information and can lead to a wider range of relevant design guidelines through the use of hyperlinks. All participants find UISGPT's capability in identifying to be subtle and easy to overlook, with relatively simple design smells, including issues with font size, color contrast, and missing labels/icons. At different difficulty levels, the performance of the explanation gradually decreases as the difficulty increases. In particular, there is a more significant decline from easy to medium. One possible explanation is that the easy guidelines are primarily about regulating parts of the text styles (such as length) and whether multiple components appear together, without involving complex semantic analysis.

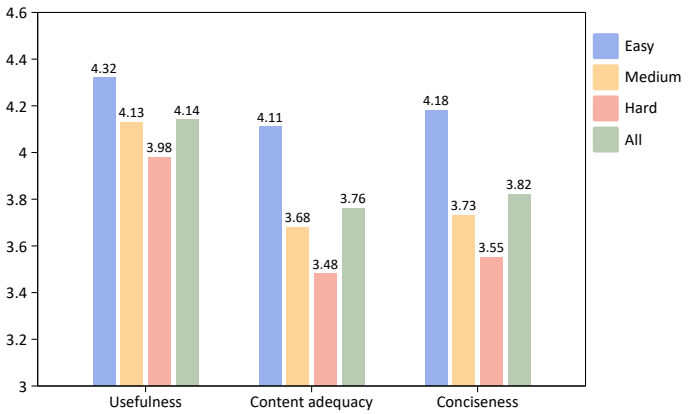
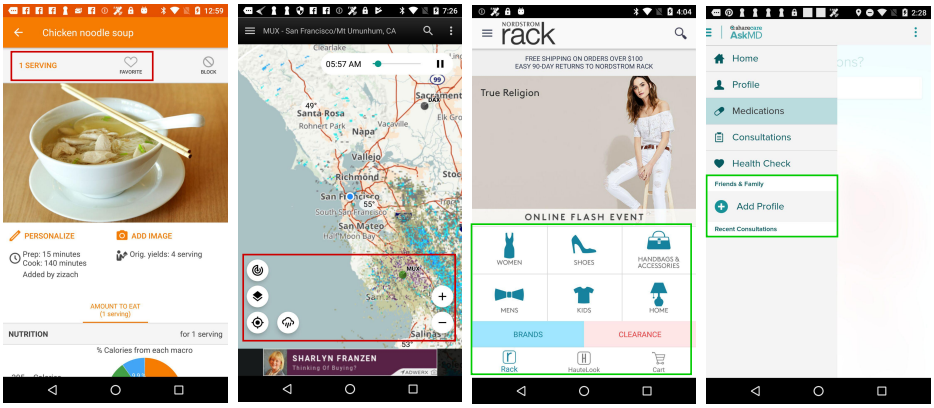


Figure 8. Average usefulness, content adequacy, and conciseness ratings of explanations across difficulty levels.

Participants also point out some instances where UISGPT performs poorly, such as the over-application of design guidelines without taking into account actual semantics (Figure 9c) and popular design trends (Figure 9d). Figure 9 contains examples of high- and low-rated reports. Figure 9a,b are considered useful and consistent. Figure 9a points out the presence of icons in the labels, while Figure 9b addresses the excessive use of FABs. Figure 9c,d are viewed as not helpful yet still consistent, as these issues are technically accurate, although some semantic labels in the UIs help users eliminate these smells.



(a) In the given design, the main component <tab> are composed of "1 SERVING", "FAVORITE", and "BLOCK". However, the sub-tab of "1 SERVING" does not contain an icon, while the sub-tab "FAVORITE" does. Adding an icon to "1 SERVING" could enhance visual coherence.
Violated Guideline: Tabs - Don't mixing tabs that contain only text with tabs that contain only icons.

(b) In a well-structured design, the number of <FABs> should less than three. However, the screen has floating action buttons named "ZoomIn", "ZoomOut", "FindMe", "CurrentConditions", "Product", and "Alerts". The number of FABs is six, which exceeds 3 three.
Violated Guideline: FABs - Don't displaying multiple FABs on a single screen.

(c): In the given design, the main components <bottom_navigation> and <tabLayout> appear at the same time, making it unclear which operation or label controls which content.

Violated Guideline: Bottom Navigation - Don't using a bottom navigation and tab together.

(d): To maintain visual consistency, the use of icons in the navigation drawer should remain uniform. In the list of <navigation drawer>, some items contain <ImageView> and <TextView>, whereas "Friends & Family" only has <TextView>.
Violated Guideline: Navigation Drawer - Don't applying icons to some destinations but not others in a navigation drawer.

Figure 9. Examples of UISGPT reports that participants find helpful or unhelpful, along with their corresponding UIs above (with the relevant group marked). (**a**,**b**) Considered useful and consistent. (**a**) Indicating the presence of icons in the labels. (**b**) Excessive use of FABs. (**c**,**d**) Not helpful yet still consistent, as these issues are technically accurate, although some semantic labels in the UIs help users eliminate these smells.

5. Threats to Validity

5.1. Internal Validity

In this study, two evaluators manually annotate the design smells in the sampled UIs. The results might be influenced by the evaluators' subjective biases. To reduce this threat, a double-validation process is employed to minimize subjectivity in the annotation task. For each task, the evaluators first complete their annotations independently, then discuss any discrepancies to reach a consensus.

In Section 3.4, a temperature of 0.65 was set, which implies that the LLM may produce different outputs even under identical input conditions. To mitigate the potential impact of this variability on results and stability, five repeated experiments are conducted. A majority-voting strategy is employed to determine the final result, thereby enhancing internal validity.

As described in Section 3.2.1, our approach converts the view hierarchy into HTML language, but there are two limitations. The native classes in the view hierarchy do not always match HTML tags, and including all attributes of the GUI components results in excessively lengthy HTML text. To mitigate these risks, this paper first uses the `<div>` tag to encapsulate classes not covered by specific mappings, ensuring no loss of metadata. Second, for excessively long inputs, we segment the input for the LLM.

5.2. External Validity

In this study, we use Google Material Design [8] as a case study. Since it is based on a specific design philosophy, it may differ from other design paradigms. Consequently,

our approach may not be applicable to other design systems and platforms, such as iOS or alternative design systems. Additionally, the applicability of our findings may vary depending on specific demographic factors.

6. Conclusions and Future Work

This paper proposes an LLM-based system that automatically analyzes scattered textual UI design guidelines and reviews the input UI for design violation detection. This work selects Google Material Design [8] as a case study and develops an Android design violation detection tool named UISGPT, which can analyze the input UI and generate an interpretable design violation detection report. This study involved experiments on several popular large models, including multimodal models such as GPT-4 with vision, indicating that the full system achieves the best performance on GPT-4. The evaluation demonstrates that UISGPT significantly outperforms existing solutions in UI design smell detection. The superiority of UISGPT lies in its ability to generate high-quality, interpretable reports that avoid hallucinations.

One of the critical advantages of UISGPT is its capacity to produce smell reports with minimal hallucinations, as evidenced by its highest recall. Deep learning methods are often regarded as black-box models [74], making it difficult to interpret their internal workings and detection results. In contrast, UISGPT, which is powered by advanced prompting techniques with LLMs, ensures that each detected design smell is accompanied by a clear, concise, and contextually relevant explanation. This enhances the utility of the reports for designers and developers, who rely on accurate and actionable insights to refine their user interfaces.

Moreover, the interpretability of UISGPT's reports sets a new benchmark in the field. UISGPT translates complex design guidelines into straightforward, easily understandable detection reports. This not only aids in the immediate identification and correction of design issues but also contributes to a deeper understanding of design principles among practitioners. The ability to bridge the gap between technical detection and human-readable interpretation is a testament to the robustness and user-centric nature of UISGPT.

In the future, one of the main directions is to expand UISGPT's accessibility and ease of use by developing practical tools like browser plugins. These plugins could operate as an interface layer between the developers and the underlying complexity of the design guidelines, offering instant recommendations and explanations directly within the design environment. As developers work on their projects, a plugin could detect potential design violations in real time and suggest improvements, effectively facilitating the design process and enhancing efficiency.

Another main direction is the use of specific design-related corpora to train domain LLMs. The resulting knowledge extractor can provide more precise and context-relevant knowledge, thereby improving the performance of UISGPT, in addition to allowing for a broader, more complicated source of design paradigms, such as the fusion of graphical design cases and textual blogs. By analyzing the design patterns and guidelines that have received positive feedback in the design community, the system can provide more personalized and community-validated suggestions. Such improvements would not only enhance the tool's response results through deeper insights but also refine its explanatory answers to accommodate a broader range of design queries.

In summary, based on the proposed LLM-based design violation detection framework and UISGPT, these future directions can assist designers and developers in creating aesthetic, functional, and user-centered designs with the aid of design guidelines.

Author Contributions: Conceptualization, B.Y. and S.L.; methodology, B.Y.; software, B.Y.; validation, B.Y.; formal analysis, B.Y.; investigation, B.Y.; resources, B.Y.; data curation, B.Y.; writing—original draft preparation, B.Y.; writing—review and editing, B.Y. and S.L.; visualization, B.Y.; supervision, S.L.; project administration, B.Y.; funding acquisition, S.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: The study was conducted in accordance with the Declaration of Helsinki, and the protocol was approved by the Ethics Committee of biomedical engineering & instrument science, Zhejiang University ([2024]No. 8) (approved on 26 June 2024).

Informed Consent Statement: All subjects provided informed consent for inclusion before they participated in the study.

Data Availability Statement: The UI design smell datasets that were used and analyzed during the current study are accessible at the following URL: https://github.com/deviohunter/UI-Design-Smell-GPT-Dataset (accessed on 19 July 2024). This dataset is distributed under a CC BY-NC-SA License. Version 1.0 of the software are accessible at the following URL: https://www.coze.com/store/bot/7387799161887965189?panel=1 (accessed on 19 July 2024).

Acknowledgments: Research work mentioned in this paper is supported by State Street Zhejiang University Technology Center.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Chen, J.; Chen, C.; Xing, Z.; Xia, X.; Zhu, L.; Grundy, J.; Wang, J. Wireframe-based UI design search through image autoencoder. *ACM Trans. Softw. Eng. Methodol.* **2020**, *29*, 19. [CrossRef]
2. Nielsen, J. 10 Usability Heuristics for User Interface Design. 1994. Available online: https://www.nngroup.com/articles/ten-usability-heuristics/ (accessed on 21 July 2024).
3. Galitz, W.O. *The Essential Guide to User Interface Design: An Introduction to GUI Design Principles and Techniques*, 3rd ed.; Wiley: Hoboken, NJ, USA, 2007.
4. Yang, B.; Xing, Z.; Xia, X.; Chen, C.; Ye, D.; Li, S. Don't do that! Hunting down visual design smells in complex UIs against design guidelines. In Proceedings of the 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), Madrid, Spain, 22–30 May 2021.
5. Ali, A.; Xia, Y.; Navid, Q.; Khan, Z.A.; Khan, J.A.; Aldakheel, E.A.; Khafaga, D. Mobile-UI-Repair: A deep learning based UI smell detection technique for mobile user interface. *PeerJ Comput. Sci.* **2024**, *10*, e2028. [CrossRef] [PubMed]
6. Aleksi, V. Guidelines Supported Wvaluation of User Interfaces with Generative AI. Master's Thesis, Aalto University, Espoo, Finland, 2024. Available online: https://aaltodoc.aalto.fi/items/39a59822-2d1d-473c-bd9e-127464bb8a13 (accessed on 21 July 2024).
7. Fowler, M.; Beck, K.; Brant, J.; Opdyke, W.; Roberts, D. *Refactoring: Improving the Design of Existing Code*; Addison Wesley: Boston, MA, USA, 1999.
8. Google. Google Material Design. Available online: https://m2.material.io/components/ (accessed on 21 July 2024).
9. Zhang, Z.; Feng, Y.; Ernst, M.D.; Porst, S.; Dillig, I. Checking conformance of applications against GUI policies. In Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, New York, NY, USA, 23–28 August 2021.
10. Zhao, D.; Xing, Z.; Chen, C.; Xu, X.; Zhu, L.; Li, G.; Wang, J. Seenomaly: Vision-based linting of GUI animation effects against design-don't guidelines. In Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering (ICSE), Seoul, Republic of Korea, 23–29 May 2020.
11. Liu, Z.; Chen, C.; Wang, J.; Che, X.; Huang, Y.; Hu, J.; Wang, Q. Fill in the blank: Context-aware automated text input generation for mobile gui testing. In Proceedings of the ACM/IEEE 45nd International Conference on Software Engineering (ICSE), Melbourne, Australia, 14–20 May 2023.
12. Feng, S.; Yuan, M.; Chen, J.; Xing, Z.; Chen, C. Designing with Language: Wireframing UI Design Intent with Generative Large Language Models. *arXiv* **2023**, arXiv:2312.07755.
13. Wang, B.; Li, G.; Li, Y. Enabling conversational interaction with mobile ui using large language models. In Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems, Hamburg, Germany, 23–28 April 2023.
14. Huang, Y.; Xu, J.; Jiang, Z.; Lai, J.; Li, Z.; Yao, Y.; Chen, T.; Yang, L.; Xin, Z.; Ma, X. Advancing transformer architecture in long-context large language models: A comprehensive survey. *arXiv* **2023**, arXiv:2311.12351.
15. Lee, P.; Bubeck, S.; Petro, J. Benefits, limits, and risks of GPT-4 as an AI chatbot for medicine. *N. Engl. J. Med.* **2023**, *388*, 1233–1239. [CrossRef]
16. OpenAI; Achiam, J.; Adler, S.; Agarwal, S.; Ahmad, L.; Akkaya, I.; Aleman, F.L.; Almeida, D.; Altenschmidt, J.; Altman, S.; et al. GPT-4 Technical Report. *arXiv* **2023**, arXiv:2303.08774.
17. Anil, R.; Dai, A.M.; Firat, O.; Johnson, M.; Lepikhin, D.; Passos, A.; Shakeri, S.; Taropa, E.; Bailey, P.; Chen, Z.; et al. PaLM 2 Technical Report. *arXiv* **2023**, arXiv:2305.10403.
18. Touvron, H.; Lavril, T.; Izacard, G.; Martinet, X.; Lachaux, M.-A.; Lacroix, T.; Rozière, B.; Goyal, N.; Hambro, E.; Azhar, F.; et al. LLaMA: Open and Efficient Foundation Language Models. *arXiv* **2023**, arXiv:2302.13971.
19. OpenAI. GPT-4V(ision) System Card. Available online: https://openai.com/research/gpt-4v-system-card (accessed on 21 July 2024).

20. Alegroth, E.; Gao, Z.; Oliveira, R.; Memon, A. Conceptualization and evaluation of component-based testing unified with visual gui testing: An empirical study. In Proceedings of the IEEE 8th International Conference on Software Testing, Verification and Validation (ICST), Dublin, Ireland, 16–20 April 2023.
21. Liu, Z.; Chen, C.; Wang, J.; Huang, Y.; Hu, J.; Wang, Q. Owl Eyes: Spotting UI Display Issues via Visual Understanding. In Proceedings of the IEEE/ACM 35th International Conference on Automated Software Engineering (ASE), Melbourne, Australia, 21–25 December 2020.
22. Su, Y.; Liu, Z.; Chen, C.; Wang, J.; Wang, Q. OwlEyes-online: A fully automated platform for detecting and localizing UI display issues. In Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Online, 19–28 August 2021.
23. Liu, Z.; Chen, C.; Wang, J.; Huang, Y.; Hu, J.; Wang, Q. Nighthawk: Fully automated localizing ui display issues via visual understanding. *IEEE Trans. Softw. Eng.* **2022**, *49*, 403–418. [CrossRef]
24. Issa, A.; Sillito, J.; Garousi, V. Visual testing of Graphical User Interfaces: An exploratory study towards systematic definitions and approaches. In Proceedings of the 2012 14th IEEE International Symposium on Web Systems Evolution (WSE), Trento, Italy, 28 September 2012.
25. Apple. Human Interface Guidelines. Available online: https://developer.apple.com/design/human-interface-guidelines/ (accessed on 21 July 2024).
26. Microsoft. Microsoft Interface Definition Language 3.0 Reference. Available online: https://learn.microsoft.com/en-us/uwp/midl-3 (accessed on 21 July 2024).
27. Moran, K.; Li, B.; Bernal-Cárdenas, C.; Jelf, D.; Poshyvanyk, D. Automated reporting of GUI design violations for mobile apps. In Proceedings of the 40th International Conference on Software Engineering, Gothenburg, Sweden, 27 May–3 June 2018.
28. Chen, C.; Feng, S.; Xing, Z.; Liu, L.; Zhao, S.; Wang, J. Gallery D.C. Design search and knowledge discovery through auto-created GUI component gallery. *Proc. ACM Hum. Comput. Interact.* **2019**, *3*, 180. [CrossRef]
29. Liu, Z. Woodpecker: Identifying and fixing Android UI display issues. In Proceedings of the IEEE/ACM 44th International Conference on Software Engineering (ICSE): Companion Proceedings, Pittsburgh, PA, USA, 22–27 May 2022.
30. Alotaibi, A.S.; Chiou, P.T.; Halfond, W.G.J. Automated repair of size-based inaccessibility issues in mobile applications. In Proceedings of the IEEE/ACM 36th International Conference on Automated Software Engineering (ASE), Melbourne, Australia, 14–20 November 2021.
31. Chen, K.; Li, Y.; Chen, Y.; Fan, C.; Hu, Z.; Yang, W. Glib: Towards automated test oracle for graphically-rich applications. In Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Online, 19–28 August 2021.
32. Su, Y.; Chen, C.; Wang, J.; Liu, Z.; Wang, D.; Li, S.; Wang, Q. The Metamorphosis: Automatic Detection of Scaling Issues for Mobile Apps. In Proceedings of the IEEE/ACM 37th International Conference on Automated Software Engineering (ASE), Rochester, MI, USA, 10–14 October 2022.
33. Schoop, E.; Zhou, X.; Li, G.; Chen, Z.; Hartmann, B.; Li, Y. 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, New Orleans, LA, USA, 30 April–5 May 2022.
34. Lee, C.; Kim, S.; Han, D.; Yang, H.; Park, Y.-W.; Kwon, B.C.; Ko, S. GUIComp: A GUI design assistant with real-time, multi-faceted feedback. In Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems, Honolulu, HI, USA, 25–30 April 2020.
35. Zeng, Z.; Tan, H.; Zhang, H.; Li, J.; Zhang, Y.; Zhang, L. An extensive study on pre-trained models for program understanding and generation. In Proceedings of the 31st ACM SIGSOFT international symposium on software testing and analysis (ISSTA), Online, 18–22 July 2022.
36. Poesia, G.; Polozov, O.; Le, V.; Tiwari, A.; Soares, G.; Meek, C.; Gulwani, S. Synchromesh: Reliable code generation from pre-trained language models. *arXiv* **2022**, arXiv:2201.11227.
37. Jiang, N.; Liu, K.; Lutellier, T.; Tan, L. Impact of code language models on automated program repair. *arXiv* **2023**, arXiv:2302.05020.
38. Nashid, N.; Sintaha, M.; Mesbah, A. Retrieval-based prompt selection for code-related few-shot learning. In Proceedings of the ACM/IEEE 45nd International Conference on Software Engineering (ICSE), Melbourne, Australia, 14–20 May 2023.
39. Ahmed, T.; Devanbu, P. Few-shot training LLMs for project-specific code-summarization. In Proceedings of the IEEE/ACM 37th International Conference on Automated Software Engineering (ASE), Rochester, MI, USA, 10–14 October 2022.
40. Li, Y.; Yang, Z.; Guo, Y.; Chen, X. Droidbot: A lightweight ui-guided test input generator for android. In Proceedings of the IEEE/ACM 39th International Conference on Software Engineering (ICSE): Companion Proceedings, Buenos Aires, Argentina, 20–28 May 2017.
41. Android Studio. UI/Application Exerciser Monkey. Available online: https://developer.android.com/studio/test/other-testing-tools/monkey (accessed on 21 July 2024).
42. Su, T.; Meng, G.; Chen, Y.; Wu, K.; Yang, W.; Yao, Y.; Pu, G.; Liu, Y.; Su, Z. Guided, stochastic model-based GUI testing of Android apps. In Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, Paderborn, Germany, 4–8 September 2017.
43. Choi, W.; Necula, G.; Sen, K. Guided gui testing of android apps with minimal restart and approximate learning *ACM Sigplan Not.* **2013**, *48*, 623–640. [CrossRef]

44. Pan, M.; Huang, A.; Wang, G.; Zhang, T.; Li, X. Reinforcement learning based curiosity-driven testing of Android applications. In Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA), Online, 18–22 July 2020.
45. Liu, Z.; Chen, C.; Wang, J.; Chen, M.; Wu, B.; Che, X.; Wang, D.; Wang, Q. Make LLM a Testing Expert: Bringing Human-like Interaction to Mobile GUI Testing via Functionality-aware Decisions. In Proceedings of the ACM/IEEE 46th International Conference on Software Engineering (ICSE), Lisbon, Portugal, 14–20 April 2024.
46. Su, T.; Wang, Y.; Su, Z. Benchmarking automated gui testing for android against real-world bugs. In Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Online, 19–28 August 2021.
47. Su, Y.; Liao, D.; Xing, Z.; Huang, Q.; Xie, M.; Lu, Q.; Xu, X. Enhancing Exploratory Testing by Large Language Model and Knowledge Graph In Proceedings of the ACM/IEEE 46th International Conference on Software Engineering (ICSE), Lisbon, Portugal, 14–20 April 2024.
48. Zimmermann, D.; Koziolek, A. Automating GUI-based Software Testing with GPT-3. In Proceedings of the 2023 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), Dublin, Ireland, 16–20 April 2023.
49. Yu, S.; Fang, C.; Ling, Y.; Wu, C.; Chen, Z. Llm for test script generation and migration: Challenges, capabilities, and opportunities. In Proceedings of the 2023 IEEE 23rd International Conference on Software Quality, Reliability, and Security (QRS), Chiang Mai, Thailand, 22–26 October 2023.
50. Taeb, M.; Swearngin, A.; Schoop, E.; Cheng, R.; Jiang, Y.; Nichols, J. Axnav: Replaying accessibility tests from natural language. *arXiv* **2023**, arXiv:2310.02424.
51. Duan, P.; Warner, J.; Li, Y.; Hartmann, B. Generating Automatic Feedback on UI Mockups with Large Language Models. In Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems, Honolulu, HI, USA, 11–16 May 2024.
52. Vu, M.D.; Wang, H.; Li, Z.; Chen, J.; Zhao, S.; Xing, Z.; Chen, C. GPTVoiceTasker: LLM-Powered Virtual Assistant for Smartphone. *arXiv* **2024**, arXiv:2401.14268.
53. Liu, Z.; Chen, C.; Wang, J.; Chen, M.; Wu, B.; Che, X.; Wang, D.; Wang, Q. Chatting with GPT-3 for Zero-Shot Human-Like Mobile Automated GUI Testing. *arXiv* **2023**, arXiv:2305.09434.
54. Wang, J.; Huang, Y.; Chen, C.; Liu, Z.; Wang, S.; Wang, Q. Software testing with large language models: Survey, landscape, and vision. *IEEE Trans. Softw. Eng.* **2024**, *50*, 911–936. [CrossRef]
55. Zhou, D.; Schärli, N.; Hou, L.; Wei, J.; Scales, N.; Wang, X.; Schuurmans, D.; Cui, C.; Bousquet, O.; Le, Q.; et al. Least-to-most prompting enables complex reasoning in large language models. *arXiv* **2022**, arXiv:2205.10625.
56. Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Ichter, B.; Xia, F.; Chi, E.H.; Le, Q.V.; Zhou, D. Chain-of-thought prompting elicits reasoning in large language models. In Proceedings of the 36th International Conference on Neural Information Processing Systems, New Orleans, LA, USA, 28 November–9 December 2022.
57. Zhang, Z.; Zhang, A.; Li, M.; Smola, A. Automatic Chain of Thought Prompting in Large Language Models. *arXiv* **2022**, arXiv:2210.03493.
58. Brown, T.B.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A. et al. Language models are few-shot learners. In Proceedings of the 34th International Conference on Neural Information Processing Systems, Online, 6–12 December 2020.
59. Min, S.; Lyu, X.; Holtzman, A.; Artetxe, M.; Lewis, M.; Hajishirzi, H.; Zettlemoyer, L. Rethinking the Role of Demonstrations: What Makes In-Context Learning Work? *arXiv* **2022**, arXiv:2202.12837.
60. Burns, A.; Arsan, D.; Agrawal, S.; Kumar, R.; Saenko, K.; Plummer, B.A. A dataset for interactive vision-language navigation with unknown command feasibility. In Proceedings of the 16th European Conference on Computer Vision, Online, 23–28 August 2020.
61. Feng, S.; Chen, C. Prompting Is All You Need: Automated Android Bug Replay with Large Language Models. In Proceedings of the ACM/IEEE 46th International Conference on Software Engineering (ICSE), Lisbon, Portugal, 14–20 April 2024.
62. OpenAI. GPT-4o vs. GPT-4-turbo-2024-04-09, GPT-4o loses. Available online: https://community.openai.com/t/gpt-4o-vs-gpt-4-turbo-2024-04-09-gpt-4o-loses/764328 (accessed on 21 July 2024).
63. Deka, B.; Huang, Z.; Franzen, C.; Hibschman, J.; Afergan, D.; Li, Y.; Nichols, J.; Kumar, R. 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, Québec City, QC, Canada, 22–25 October 2017.
64. Google. Google Play Store. Available online: https://play.google.com/store/ (accessed on 21 July 2024).
65. Singh, R.; Mangat, N.S. *Elements of Survey Sampling*; Springer: Dordrecht, The Netherlands, 2010.
66. Figma. Figma: The Collaborative Interface Design Tool. Available online: https://www.figma.com/ (accessed on 21 July 2024).
67. Invisionapp. Sketch Templates—Collections. InVision. Available online: https://www.invisionapp.com/free-resources/collections/sketch-templates (accessed on 21 July 2024).
68. Sketchrepo. Free Sketch Mobile App Prototypes, Templates, Wireframes and Concepts—Sketch Repo. Available online: https://sketchrepo.com/tag/free-sketch-app-design/ (accessed on 21 July 2024).
69. Taibi, D.; Janes, A.; Lenarduzzi, V. How developers perceive smells in source code: A replicated study. *Inf. Softw. Technol.* **2017**, *92*, 223–235. [CrossRef]
70. Deng, S.; Xu, W.; Sun, H.; Liu, W.; Tan, T.; Liu, J.; Li, A.; Luan, J.; Wang, B.; Yan, R.; et al. Mobile-Bench: An Evaluation Benchmark for LLM-based Mobile Agents. *arXiv* **2024**, arXiv:2407.00993.

71. Landis, J.R.; Koch, G.G. An application of hierarchical kappa-type statistics in the assessment of majority agreement among multiple observers. *Biometrics* **1977**, *33*, 363–374. [CrossRef]
72. Jowett, G.H.; Fisher, R.A. Statistical methods for research workers. *J. R. Stat. Soc. Ser. C Appl. Stat.* **1956**, *5*, 68. [CrossRef]
73. Tukey, J.W. Comparing individual means in the analysis of variance. *Biometrics* **1956**, *5*, 99–114. [CrossRef]
74. Buhrmester, V.; Münch, D.; Arens, M. Analysis of Explainers of Black Box Deep Neural Networks for Computer Vision: A Survey. *Mach. Learn. Knowl. Extr.* **2021**, *3*, 966–989. [CrossRef]