GUIGroup: Enabling Functional Layout Grouping with Multimodal Large Language Models

Yulei Zhang, Yifan Yang, Zejun Zhang, Xiaoqian Li, Kui Liu, Zhenchang Xing, Xin Xia, Lingfeng Bao

***Abstract*—Modern graphical user interfaces usually contain a large number of scattered GUI widgets, which need to be organized into functionally related layout groups to provide a good user experience and design consistency. As the core task of GUI automated analysis, functional layout grouping can not only support interface design evaluation and optimization, but also provide an important foundation for downstream tasks such as GUI to code generation, interface understanding and automated testing. Previous functional layout grouping research mainly relies on traditional computer vision methods to group interface elements through heuristic rules and psychological principles. Unfortunately, due to the lack of adaptability of rule-based methods and insufficient understanding ability in dense layouts, the performance of these methods is not satisfactory. To address these limitations, we propose the GUIGROUP method to achieve accurate functional layout grouping by identifying GUI widgets in screenshots and combining widget detection techniques with MLLM. The evaluation on a dataset of 1,612 GUIs manually collected from 596 Android apps shows that GUIGROUP significantly outperforms the baseline method with an F1 score of 0.5353, which is 12.65% higher than the best baseline. In addition, ablation studies deeply verify the key contributions of each step, and user studies further confirm the superiority and practical value of the method in real application scenarios.**

***Index Terms*—GUI, MLLM, functional layout grouping**

I. INTRODUCTION

WITH the increasing complexity and diversity of modern mobile graphical user interfaces (GUIs), there is a growing need to understand UI layouts structurally, not as isolated visual elements, but as hierarchies of task-oriented components that jointly implement user intents. The core of this structural understanding lies in functional layout grouping, which clusters widgets into semantically consistent units based on their role in the interface. This capability serves as a critical bridge across the GUI ecosystem. For designers, functional layout grouping enables systematic extraction of recurring functional modules from a large-scale UI corpus, thus helping them to systematically analyze design patterns. For developers, it facilitates modular GUI-to-code generation by understanding UI images, transforming flat lists of controls into nested, reusable parts. In automated user interface testing, test scripts must be able to reliably locate and interact with functional modules, rather than interacting with fragile individual elements. The lack of accurate functional layout grouping can negatively impact design analysis and automation. Therefore, developing a method for functional layout grouping is crucial.

Human understanding of GUI relies neither solely on low-level visual perception nor solely on the induction of patterns. Instead, it combines low-level visual grouping with high-level functional reasoning based on the principles of design. Drawing from atomic design methodology [1], interfaces are inherently hierarchical: atomic elements (e.g., buttons, input fields) combine into functional molecules (e.g., search components), which further aggregate into organisms (e.g., navigation bars) that serve distinct user goals within specific task contexts. This composite structure reflects how humans interpret interfaces from a cognitive perspective. A functional layout is an independent module in the interface. This module has a clear functional goal and is usually composed of multiple basic UI widgets(Atoms and Molecules) and is classified as Organisms. Functional layouts usually have a specific responsibility, such as navigation, search, content display or interactive operation.

Existing approaches for functional layout grouping primarily rely on hand-crafted heuristics [2]–[4] or purely vision-based deep learning [5]. While effective in specific scenarios, they fundamentally struggle to bridge the gap between low-level visual cues and high-level functional semantics. As illustrated in Fig.1, strict rule-based methods often fail to associate a section title (e.g., “Ringtones” and “Animated Call Screens”) with its corresponding content due to rigid spatial constraints (Fig. 1a), while purely vision-based models tend to over-segment components based on pixel differences, ignoring their semantic unity (Fig. 1b). These failures highlight a critical limitation: the inability to comprehend the textual context and logical relationships inherent in GUIs.

This motivates the introduction of Multimodal Large Language Models (MLLMs). In recent years, multimodal large language models (MLLMs) such as GPT-4V [6], Qwen-VL [7] and Claude [8] have shown great potential in understanding and analyzing GUIs with complex layouts due to their advanced semantic understanding capabilities [9], [10]. These models can accurately identify and describe the functional relationships between different UI widget. Crucially, MLLM is able to address most of the aforementioned limitations. They generalize using pre-trained world knowledge, no longer

Yulei Zhang, Yifan Yang, Xin Xia and Lingfeng Bao are with The State Key Laboratory of Blockchain and Data Security, College of Computer Science and Technology, Zhejiang University, Hangzhou, China. Lingfeng Bao is also with Hangzhou High-Tech Zone (Binjiang) Insti-tute of Blockchain and Data Security. E-mail: {yuleizhang, yifan_yang, lingfengbao}@zju.edu.cn, xin.xia@acm.org

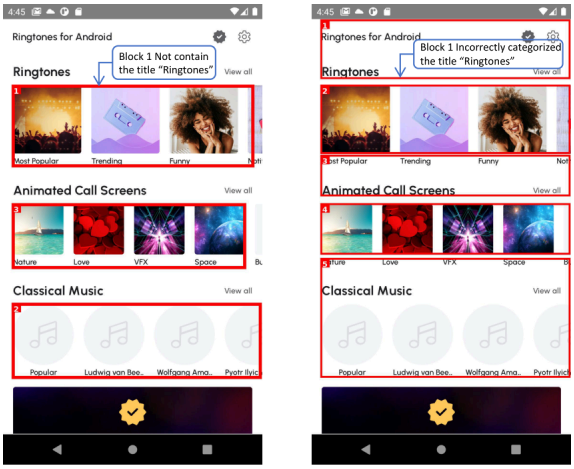
Zejun Zhang is with Nanjing University. E-mail: zejun.zhang@nju.edu.cn

Zhenchang Xing is with the CSIRO’s Data61 and Australian National University. E-mail: zhenchang.xing@anu.edu.au

Xiaoqian Li is an independent researcher. E-mail: lixiaoqian008@gmail.com

Kui Liu is with Huawei. E-mail: kui.liu@huawei.com

Lingfeng Bao is the corresponding author.



(a) Based on rules (b) Based solely on vision

Fig. 1. Typical causes of grouping mistakes: (a) Over-Segmentation (b) Layout Occlusion

bound by fixed coding rules; they can reason about functional semantics beyond visual appearance and correctly group elements based on common uses; they reduce reliance on large amounts of GUI specific training data because they encode rich prior information about the user interface structure and user tasks during multimodal pre-training [9], [10]. However, MLLMs face a key limitation: they cannot accurately generate coordinate information for GUI widgets and layouts [11], [12]. Despite progress in widget-based training, they still face challenges in fully understanding GUI layouts and accurate bounding box prediction [13], [14].

To overcome the limitations of both purely visual and purely MLLM approaches and more systematically group functionally related widgets in GUI screenshots at the layout level, we propose GUIGROUP, a GUI widget grouping method that combines widget detection techniques and MLLM. The method aims to automatically identify and organize scattered GUI widgets into functionally related layout groups, enabling better understanding of interface structure.

First, we employ traditional techniques for GUI widget detection to obtain the coordinates and bounding boxes. Specifically, we use OCR to extract text and YOLOv8 object detection model to identify non-text widgets, respectively. Then, we apply an MLLM to perform functional semantic analysis at the layout level on the GUI screenshot. MLLM has strong image understanding capabilities. It can generate precise functional layout information in natural language, including the layout's name, functionality, and more. To further ensure precise coordinate information, we grouped the functional layouts by combining the coordinates of specific widgets detected with the functional layout information, using MLLM for coordinate localization. Finally, it should be acknowledged that widget detection algorithms may exhibit limitations in comprehensively identifying GUI widgets or achieving precise localization. GUIGROUP includes a widget sanity review mechanism that goes from detecting missing GUI widgets to understanding and merging them.

Since there is no existing dataset for functional layout grouping, we manually created a comprehensive dataset by annotating functional layouts of GUI screenshots from two datasets. We collected screenshots from the MUD dataset [15] and our newly collected LDT dataset (The **L**atest **D**esign **T**rends Dataset). To ensure the reliability of our ground truth, we carefully filtered the data and had each screenshot annotated independently by two individuals, followed by a thorough review process. In total, we annotated 1,031 screenshots from the MUD dataset and 581 screenshots from the LDT dataset.

In summary, we make the following contributions:

- We propose GUIGROUP, a four-stage functional layout grouping framework that, to our knowledge, is the first to formally instantiate the Atomic Design methodology as a computational pipeline for GUI layout understanding. Unlike end-to-end MLLM approaches that suffer from spatial hallucination, GUIGROUP introduces a principled semantic-to-geometric decoupling, enabling both semantic richness and spatial precision simultaneously.
- To assess the effectiveness of the proposed approach, we build a dataset by manually annotating the functional layouts of more than 1,600 GUI screenshots, sourced from both the MUD dataset and our newly collected LDT dataset, with explicit annotation guidelines grounded in the Atomic Design framework.
- We evaluate GUIGROUP on the dataset and show that it outperforms all baselines. Additionally, ablation studies confirm the effectiveness of key components in GUIGROUP, and user studies demonstrate the practical usefulness of our approach.

The remainder of this paper is organized as follows: Section II describes our GUIGROUP approach. Section III introduces the experimental setup and dataset. Section IV presents the evaluation results. Section V discusses key findings and limitations. Section VI reviews related work, and Section VII concludes the paper.

II. APPROACH

In this section, we describe the process of GUIGROUP, which is shown in Fig. 2. Given a GUI screenshot, GUIGROUP first detects the text and non-text widgets with accurate coordinate information. It then employs MLLM to conduct functional semantic analysis and generate descriptions of the functional layouts. Next, GUIGROUP integrates these results to pinpoint the coordinates of each layout. Finally, a review step is performed, where MLLM identifies any missing widgets and merges layouts as needed.

A. GUI widget Detection

A functional layout in a GUI screenshot is composed of basic widgets such as icons, buttons, and texts. Identifying these widgets is crucial for effectively grouping and understanding the functional layout. We classify widgets in GUI screenshots into two categories: text and non-text. To detect text widgets, we employ OCR technology, which has proven effective in extracting text from screenshots [3]. Specifically, we use PaddleOCR [16], a deep learning-based tool that supports over 80 languages. PaddleOCR works well out-of-the-box for

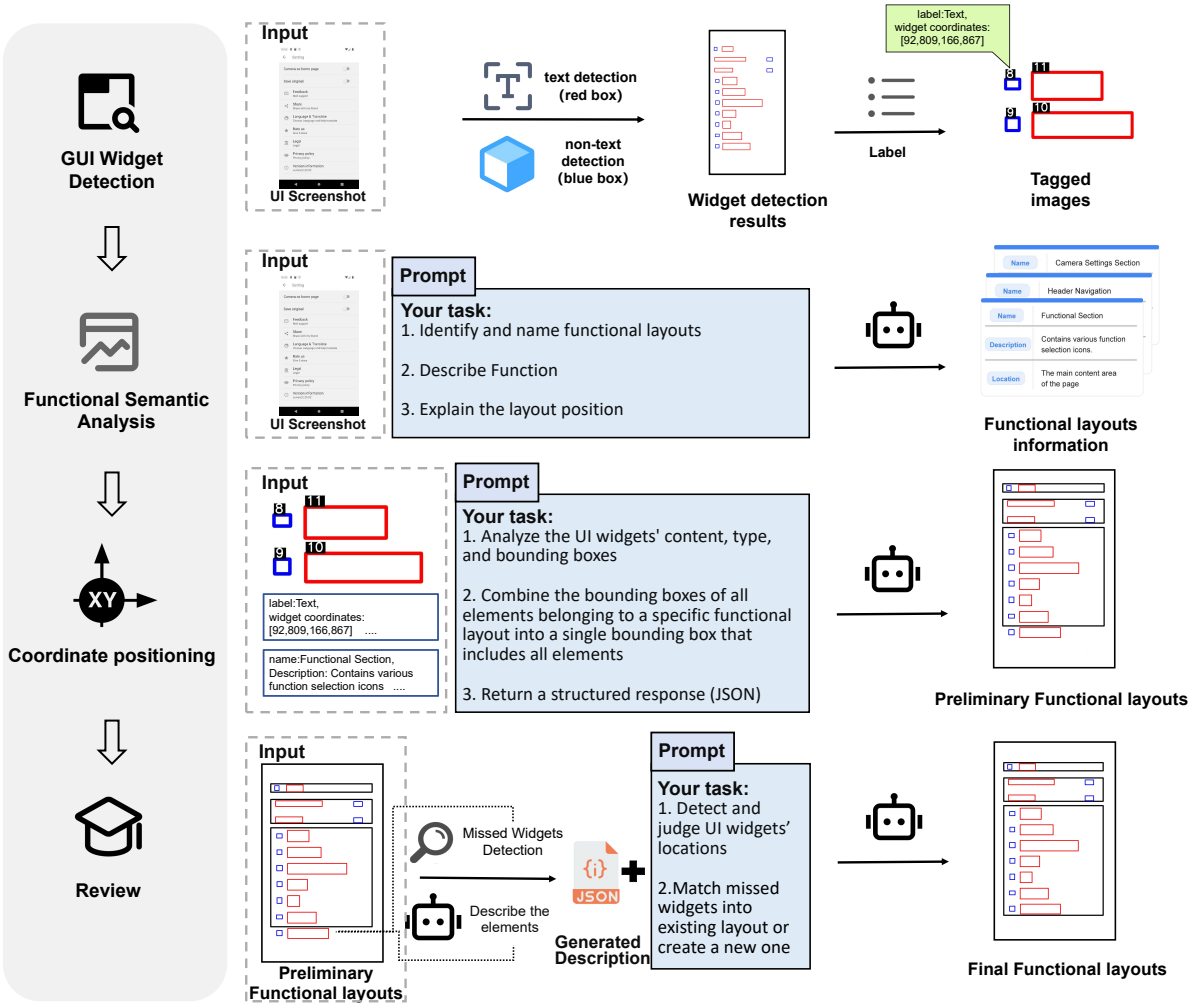


Fig. 2. Overview of GUIGROUP

mobile GUI screenshots without requiring application-specific filtering or customization. For non-text GUI widgets, YOLOv8 [17] is an ideal choice for our research because it achieves the best balance between accuracy, efficiency, and number of parameters. We use a version of YOLOv8 trained by Liu et al. [18] specifically for identifying GUI widgets. This model, trained on a dataset of 1,392 screenshots from public applications, can accurately detect eight basic GUI widgets, including buttons, text views, and sliders. To avoid overlap between text and non-text widgets, we apply Intersection-over-Area (IoA)-based filtering: when the IoA between a detected text region and a non-text bounding box exceeds 0.7, the non-text detection is suppressed, as text widgets generally provide more precise boundary information. Furthermore, when the area of text contained within a non-text widget exceeds 0.8 of the non-text widget's total area, the non-text widget is removed to prevent excessive inclusion. Text lines are subsequently merged into paragraphs by computing their intersection areas while accounting for line spacing. Widgets residing in the top status bar region — specifically, those whose bounding box top coordinate is anchored at the top of the screen and whose height is less than 4% of the total image height — are excluded, as they represent system-level UI elements rather than application-level functional components. In addition, to enhance the understanding ability of MLLM, we use the algorithm of Lu et al. [19] to assign a unique ID to each bounding box and record color information. As shown in Fig. 3(b), the GUI screenshot is from the Beauty Camera APP in the MUD dataset. After detection, the widgets are highlighted: red bounding boxes outline text widgets, while blue boxes mark non-text widgets. Each detected widget is then assigned a unique number and a unique color bounding box. This color annotation serves a dual purpose: it allows the MLLM to unambiguously refer to specific widgets by ID during functional semantic analysis and coordinate positioning, and it reduces spatial ambiguity in dense GUI layouts where multiple widgets are in close proximity.

B. *Functional Semantic Analysis*

MLLMs face challenges in accurately identifying the coordinates and positions of visual elements in images, where they are prone to hallucinations [9], [20]. Therefore, we start with simulating the human cognitive process of understanding GUI screenshots by adopting a top-down approach. Instead of focusing on fine-grained details first, we let the MLLM initially perceive the coarse-grained functional layout of GUIs. We prompt the MLLM to analyze the overall functional layout of the GUI screenshot. The MLLM is prompted with an explicit definition of functional layout grounded in Atomic Design [1] theory — specifically, that a functional layout

corresponds to the organism level of the hierarchy, composed of multiple widgets that together serve a coherent functional purpose. This definition is embedded directly in the prompt to constrain the MLLM's reasoning scope. Specifically, we ask the MLLM to recognize GUI screenshots and output a structured natural language description for each identified functional layout, comprising three fields: (1) name — a concise label for the layout (e.g., "Search Bar", "Navigation Menu"); (2) description — a brief functional explanation of what the layout does; and (3) contextual location — a relative spatial description (e.g., "located at the top of the screen, spanning the full width") expressed in natural language without any pixel-level coordinates. Critically, this step performs only high-level semantic reasoning and does not predict or rely on any pixel-level coordinates or bounding boxes. This deliberate avoidance of coordinate prediction at this stage is a key design decision, as it prevents the MLLM from being exposed to tasks it is known to perform poorly on, thereby reducing hallucination risk. The prompt we used can be found in the appendix. Note that we guide the MLLM to perform a top-down layout breakdown while ignoring the content belonging to the phone itself such as topbar. As shown in Fig. 2, the original screenshot is input, and the functional semantic analysis step guides MLLM to identify targeted and independent functional layouts through UI functional layout analysis prompts. These layouts are typically composed of multiple UI widgets. The final functional layout information is shown on the right. As shown in Fig. 3(c), the MLLM initially identifies two main functional layouts: the camera setting section and the functional section, as well as their descriptions. The structured output is subsequently serialized as plain text and concatenated into the prompt of the coordinate positioning stage as prior semantic knowledge, enabling precise spatial positioning in the next step.

C. Coordinate positioning

Although MLLMs are not capable of directly and accurately localizing functional layouts within GUI screenshots, we find that their understanding of the underlying semantic layout remains relatively reliable. This aligns with recent findings in UI understanding literature: for instance, Ferret-UI [9] and OmniParser [19] observe that off-the-shelf MLLMs struggle with fine-grained spatial grounding, yet works like DeclarUI [21] and UI-Vision [13] demonstrate their effectiveness in reasoning about high-level functional structure and widget relationships from visual inputs. Based on this observation, we find that MLLMs can more effectively identify and localize the functional layout of GUI screenshots when provided with precise widget information, including their spatial coordinates and semantic layout descriptions. As shown in the third column of Fig. 2, to achieve this, we organize three inputs into a unified prompt: first, the color-annotated GUI screenshot is provided as the visual input, where each detected widget is highlighted with a unique color label to facilitate visual reference; second, the detected widgets are serialized as a structured list, where each entry contains the widget's unique ID, category label, and bounding box coordinates in pixel values; third, the functional layout descriptions generated during the previous semantic analysis stage are appended as prior semantic context. The MLLM is then prompted to merge the bounding boxes of all widgets with a specific feature layout into a single bounding box containing all the widgets. The final output is a set of functional layout entries in JSON format, each comprising a layout name and its corresponding bounding box coordinates.

The Fig. 3(d) shows an actual example of a mobile application settings interface, where MLLM successfully groups and positions the various UI widgets in the interface into semantically related functional layouts.

D. Review

Although GUIGROUP obtains an initial set of functional layouts with precise coordinates after the previous step, we observe that the coordinate positioning step may leave certain detected widgets unassigned to any functional layout. The example provided in Fig. 3(d) does not include the back button and "Settings" text at the top of the image, nor does it include the last line of the function section after coordinate positioning, which contains the icon and the corresponding "Version Information" label. Therefore, GUIGROUP reviews and refines the functional layout based on all detected GUI widgets. Specifically, for each detected widget, if it is not enclosed within the bounding box of any existing functional layout, it is flagged as missing. The corresponding GUI screenshot is then marked for review, with the bounding boxes of all missing widgets retained and color-highlighted on the original screenshot for subsequent processing. By analyzing and comparing the functional layout of the upper and lower widgets, it is possible to correct these missing widgets.

To enhance the semantic review capabilities of MLLMs, we frame the final stage of our pipeline as a lightweight verification agent—a role commonly adopted in modern LLM-based systems where a single model assumes distinct functions through specialized prompting [21]. Concretely, this agent operates in two steps. In the first step, the MLLM takes as input the GUI screenshot with all missing widgets highlighted by colored bounding boxes, and produces as output a unique widget ID and a semantic description for each missing widget. Screenshots with missing widgets highlighted by colored bounding boxes are provided as visual input, helping the model better identify and understand each widget [11], [19]. Descriptions are generated one by one to reduce inter-widget interference. Once all missing widgets have been described, in the second step, the MLLM takes as input the GUI screenshot corresponding to each missing widget, along with its widget ID, semantic description, and the previously obtained functional semantic analysis results. The MLLM is then asked to determine, for each missing widget, whether it can be integrated into one of the existing functional layouts. If a merge is suggested, the widget's bounding box is combined with that of the target layout to form the minimum enclosing rectangle, thereby refining its spatial boundaries. If not, missing widgets that share similar semantics or spatial proximity are grouped together to form a new functional layout. The Fig 3(e)-(f) illustrates this process. In the first

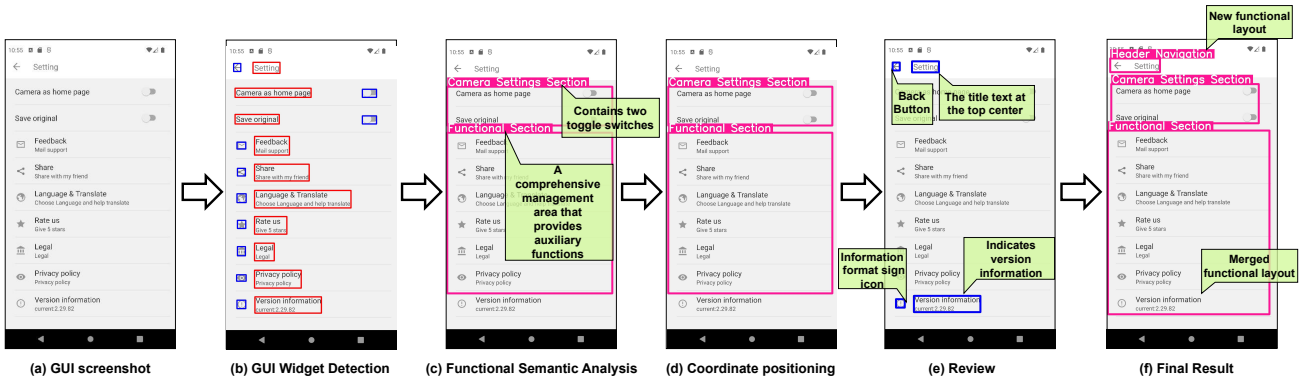


Fig. 3. An example of our approach: (a) Initial UI screenshot; (b) The widget detection result, red box for text and blue box for non-text; (c) Preliminary judgment of functional layout output, including name and description; (d) Perform functional layout coordinate positioning; (e) Check and describe the missing widgets and categorize them; (f) The final output.

step, MLLM identified and described a single missing widget. It identified the "exclamation mark" icon at the bottom and described it as an "information format sign icon", while it identified the "back" icon at the top and described it as a "back button". In the second step, MLLM grouped the missing icons and identified widgets at the bottom into "Feedback", "Share", "Language and Translation", "Rating", "Legal", "Privacy Policy", and "Version Information" and merged them into a correct functional layout. For the missing widgets at the top, MLLM found that they could not be merged with the camera settings section below, so it created a new functional layout for them. The final functional layout structure is highlighted with a pink border.

III. EXPERIMENTS

In this section, we first describe the dataset and experiment setting used in the evaluation, followed by the baseline methods and evaluation metrics.

A. Dataset

To rigorously verify the effectiveness and practicality of our approach, we construct a comprehensive ground-truth dataset. We conduct experiments on two datasets to ensure comprehensive evaluation across diverse UI designs. The first is the MUD dataset [15], which is based on applications from the Rico dataset [22]. While these applications represent earlier app design paradigms, they provide valuable diversity in layout patterns. To complement this and capture the full spectrum of UI design evolution, we also collect a new dataset of GUI screenshots from popular contemporary Android applications. This combination ensures our evaluation covers both traditional and modern interface designs, providing a more comprehensive assessment of layout diversity.

- **The MUD Dataset** was created by Feng et al. [15] and contains 18,132 unique GUI screenshots from 3,300 applications. To ensure dataset quality and representativeness, we implemented a multi-stage filtering process. We first excluded screenshots lacking corresponding application identifiers and eliminated duplicate entries from the original MUD dataset. Subsequently, we observed that certain GUI screenshots were overly simplistic and failed to adhere to established GUI design principles [4], [23], which we manually removed to maintain dataset integrity. Finally, to achieve comprehensive GUI diversity coverage while maintaining a manageable dataset size, we selected up to three screenshots with distinct interface designs from each application. As a result, we selected a final set of 1,031 GUI screenshots from 517 Android applications.
- **The LDT Dataset.** To collect a more up-to-date and representative dataset, we crawled Android applications from Android App Store, a platform that organizes apps into 33 functional categories. For each category, we selected the top three applications with at least 50 million downloads. This process resulted in a total of 79 applications, ensuring wide-ranging coverage and high popularity across various app types. For each app, we manually collected screenshots of all its functional layouts to ensure comprehensive coverage of diverse UI patterns and layouts. We set a minimum threshold of five screenshots per app to capture representative samples of each application's interface diversity and explore its various functional layouts as thoroughly as possible. Finally, we selected 581 complex GUI screenshots for our analysis.

Functional Layout Annotation The original metadata for the datasets contained noise, which affected the quality of the annotations [24]. Therefore, we use LabelMe[1] for manual annotation, and the existing metadata only serves as a basis for annotation. Our annotation process involved two experienced annotators who independently labeled the functional layout of each screenshot using LabelMe. Adhering to a strict protocol, they identified functional layouts as semantically meaningful UI blocks serving distinct user purposes. Key principles included enforcing consistent granularity with a flat, non-overlapping hierarchy, grouping widgets based on functional equivalence rather than visual styling, and ensuring pixel-aligned bounding boxes. The detailed annotation guidelines are provided in Appendix. To validate annotation quality, we conducted cross-validation on a 10% subset (161 screenshots), achieving Cohen's Kappa coefficients of 0.78 for MUD and 0.89 for LDT, indicating substantial to almost perfect inter-annotator agreement. Disagreements between annotators were

[1]LabelMe is an open-source image annotation tool available at https://github.com/wkentaro/labelme.

resolved through discussion and consultation with a third expert annotator(the cross-validation guideline is shown in Appendix). Finally, we implemented a quality assurance process where 10% of the annotations were double-checked to ensure consistency and accuracy. In the MUD and LDT datasets, we created 4,320 and 3,113 functional layouts, respectively, which is the largest high-quality functional layout dataset to date.

B. Experiment Setting

The remarkable capabilities of MLLMs are crucial for fully understanding and analyzing the diverse input formats of modern research tasks. We selected version 20241022 of Claude-3.5-sonnet, which boasts strong visual capabilities and has been widely used in research [13], [21]. We set temperature to 0.0, top_p to 0.9, and max_token to 8192. To mitigate the inherent stochasticity and potential hallucinations of MLLMs, following common practice, we employ a majority voting strategy. Specifically, we perform three independent inference runs for each query. If an answer appears at least twice among the three outputs, it is selected as the valid result. In the rare case where all three outputs diverge, we default to the result of the first run to ensure a consistent output format.

C. Baseline

We selected the most recent studies utilizing traditional techniques for functional layout recognition as baseline methods. Additionally, we included basic prompting and Chain-of-Thought (CoT) prompting as baseline approaches to evaluate the effectiveness of different prompting strategies with MLLMs. To our knowledge, no studies have yet applied MLLM to functional layout grouping, but some GUI agents have begun to consider grounding as the primary objective of the task. Therefore, in this experiment, we divided our baseline into four categories. The first category is vision-inspired methods, including Psychologically-Inspired approach [3], VisionTasker [25], and LayoutCoder [26]; the second category is based on MLLM prompting, employing basic prompts and chain-of-thought prompting; the third category is GUI agent methods, covering GUI-G^2 [27] and V2P [28]; and the fourth category combines OmniParser's [19] widget detection capabilities with the grouping strategy proposed in this paper to form a fusion optimization scheme.

- **Psychologically-Inspired approach** [3] use psychology-inspired rules, detected elements are perceptually grouped and merged to obtain specific layout information. This method applies Gestalt principles [4] such as proximity and similarity to group detected UI elements into functional blocks, mimicking how humans naturally perceive visual layouts.
- **VisionTasker** [25] segments the UI interface into semantic blocks by detecting and grouping visual boundary lines that naturally separate functional blocks. This method uses color quantization, edge detection, and line segment grouping to identify rectangular areas formed by the intersection of horizontal and vertical boundaries, and treats the UI layout as ordered functional blocks separated by visual dividing lines.
- **LayoutCoder** [26] is a UI code generation framework that includes an element relationship construction module to group structurally similar components. We adapt this module as a baseline by using its grouping results as functional layout predictions.
- **Basic Prompt** We directly prompt the MLLM to identify and group UI widgets into functional layouts without providing specific reasoning steps or intermediate guidance.
- **Chain-of-Thought Prompt** We designed step-by-step reasoning prompts for each module in GUIGROUP, explicitly instructing the MLLM to decompose complex tasks into sequential operations with clear dependencies. We break down the task into individual steps. For example, the model first performs perceptual analysis to identify salient elements, then performs spatial reasoning to determine the precise location.
- **V2P** [28] is a precise GUI element localization method that enhances GUI agent positioning capabilities through suppression attention mechanisms and Fitts' Law-inspired 2D Gaussian heatmap modeling. We adapt it as a baseline by first applying our method for functional semantic analysis, then using V2P's localization approach to detect functional layouts.
- **GUI-G^2** [27] is a GUI localization method that models interface elements as continuous Gaussian distributions rather than binary targets, combining Gaussian point rewards for precise localization with coverage rewards for spatial alignment evaluation, and adaptively adjusting variance based on element size. Similarly, we first apply our method for functional semantic analysis, then use GUI-G^2 to detect and localize functional layouts as a baseline method.
- **OmniParser** [19] is a comprehensive method for parsing user interface screenshots into structured and easy-to-understand elements for clickable actions. We adapt it as a baseline by: (1) using OmniParser to detect UI widgets, and (2) applying our grouping method to form functional layouts.

D. Metrics

To evaluate the grouping capability of GUIGROUP and the baseline methods in grouping functional layouts for GUI screenshots, we perform coordinate-based layout-level evaluation using spatial information from screenshots. We employ Intersection over Union (IoU) to establish optimal matches between ground-truth (GT) and identified functional layouts. Higher IoU values mean better spatial alignment.

Our matching protocol operates as follows: An identified functional layout is considered a candidate match for a ground-truth layout if its IoU exceeds a predefined threshold. In this study, we set the threshold values from 0.5 to 0.9 in increments of 0.1. In cases where multiple candidates exist, the one with the highest IoU value is selected as the final match. Based on the matching results at the layout level, predictions matching their corresponding ground-truth layouts are classified as true positives (TP); otherwise, they are considered false positives (FP). Any true layout that does not correspond to any predicted group is classified as a false negative (FN). Based on the

TABLE I
THE PERFORMANCE OF FUNCTIONAL LAYOUT DETECTION(IOU@0.5:0.9)

Dataset	Version	Precision	Recall	F1
MUD	Psychologically-Inspired	0.3757±0.1667	0.2536±0.1125	0.3028±0.1344
	VisionTasker	0.2297±0.1190	0.1694±0.0878	0.1950±0.1010
	Basic Prompt	0.3959±0.1568	0.5092±0.2016	0.4454±0.1764
	CoT	0.4003±0.1542	0.5146±0.1983	0.4503±0.1736
	OmniParser	0.3533±0.1275	0.3200±0.1154	0.3359±0.1212
	V2P	0.1516±0.1208	0.1451±0.1153	0.1484±0.1179
	GUI-G^2	0.4611±0.1733	0.4735±0.1779	0.4672±0.1756
	LayoutCoder	0.1450±0.0816	0.2931±0.1650	0.1940±0.1092
	GUIGROUP	**0.4904±0.2103**	**0.5186±0.2224**	**0.5041±0.2162**
LDT	Psychologically-Inspired	0.3514±0.1691	0.2724±0.1310	0.3069±0.1476
	VisionTasker	0.2727±0.1390	0.2903±0.1480	0.2812±0.1433
	Basic Prompt	0.4723±0.1807	0.5293±0.2207	0.4992±0.1910
	CoT	0.4895±0.1850	0.5385±0.2034	0.5128±0.1938
	OmniParser	0.4243±0.1599	0.3952±0.1489	0.4092±0.1542
	V2P	0.1537±0.1137	0.1347±0.0996	0.1436±0.1062
	GUI-G^2	0.5174±0.1792	0.4820±0.1669	0.4991±0.1728
	LayoutCoder	0.1947±0.0865	0.4044±0.1797	0.2629±0.1168
	GUIGROUP	**0.5705±0.2154**	**0.5760±0.2175**	**0.5732±0.2164**
Total	Psychologically-Inspired	0.3647±0.1500	0.2615±0.1076	0.3046±0.1253
	VisionTasker	0.2516±0.1155	0.2200±0.1010	0.2348±0.1078
	Basic Prompt	0.4254±0.1660	0.5176±0.2020	0.4670±0.1822
	CoT	0.4343±0.1660	0.5246±0.2004	0.4752±0.1816
	OmniParser	0.3835±0.1411	0.3515±0.1294	0.3668±0.1350
	V2P	0.1526±0.1178	0.1408±0.1087	0.1464±0.1131
	GUI-G^2	0.4834±0.1756	0.4771±0.1733	0.4802±0.1744
	LayoutCoder	0.1661±0.0836	0.3398±0.1709	0.2231±0.1122
	GUIGROUP	**0.5243±0.2010**	**0.5467±0.2189**	**0.5353±0.2143**

matching results, we compute: (1) precision ($\frac{TP}{TP+FP}$) (2) recall ($\frac{TP}{TP+FN}$), and (3) F1-score ($\frac{2\times precision\times recall}{precision+recall}$)

IV. EVALUATION

In this section, We evaluate our proposed method by answering the following questions.

- **RQ1** (Effectiveness) How does our method perform when grouping functional layouts compared to the baseline?
- **RQ2** (Ablation Study) How does each individual module in our method improve the overall performance?
- **RQ3** (Usefulness) How do users perceive the practicality and usability of our method for grouping functional layouts?

A. RQ1-Effectiveness

1) Motivation: To evaluate the effectiveness of GUIGROUP on identifying functional layouts, we compared it with baseline methods. This RQ evaluates the overall performance of GUIGROUP, integrating widget detection techniques and MLLM.

2) Method: The input of the experiment is the original screenshot of the GUI, and the output is the functional semantic description and precise position coordinates of each functional layout of the GUI. The results are then evaluated according to the indicators outlined in Section III.D and compared with the baseline method to comprehensively evaluate the performance of the method.

3) Result: Table I shows the comprehensive performance comparison of GUIGROUP with eight baseline methods on key indicators such as precision, recall, and F1 score. GUIGROUP achieves superior performance with significant improvements of 20.72%, 4.21%, and 12.65% in precision, recall, and F1-score respectively compared to the best performing baseline. On the MUD dataset, GUIGROUP achieved an F1 score of 0.5041 under the IoU@0.5:0.9 evaluation metric. Compared to the first category of vision-inspired methods—specifically the Psychologically-Inspired approach (0.3028), VisionTasker (0.1950), and LayoutCoder (0.1940)—it achieves improvements of 66.48%, 158.5%, and 159.9%, respectively. Among methods based on prompts for MLLMs, GUIGROUP also surpasses Basic Prompting (0.4454) and Chain-of-Thought (0.4503), with improvements of 13.18% and 11.95%. Furthermore, GUIGROUP notably exceeds the GUI agent methods GUI-G^2 (0.4672) and V2P (0.1436), as well as the method OmniParser (0.3668), by margins of 7.90%, 239.6%, and 40.83%, respectively. On the more structured LDT dataset, GUIGROUP's F1 score further increases to 0.5732, maintaining its leading position among all compared methods.

Among the traditional approaches, the Psychologically-Inspired method consistently outperforms VisionTasker and LayoutCoder across both datasets. The Psychologically-Inspired method achieves F1-scores of 0.3028 on MUD and 0.3069 on LDT, demonstrating the effectiveness of incorporating human perceptual patterns in layout grouping tasks. The MLLM approaches show substantial improvements over traditional methods. Notably, even the basic prompting strategy outperforms both traditional baselines, highlighting the inherent advantages of MLLMs for functional layout grouping tasks. Among them, the CoT method further improves upon basic prompting, indicating that carefully designed prompt engineering can effectively guide MLLMs toward more thorough and comprehensive functional layout analysis.

OmniParser's performance is due to its widget detection module being optimized for downstream interaction tasks, resulting in the omission of a large number of non-interactive yet structurally significant fine-grained widgets and causing information loss in the input representation. Although its subsequent use of an MLLM for reasoning allows it to slightly outperform pure visual methods, it still significantly lags behind the basic prompting strategy. This result indicates that high-quality, fine-grained UI widget information is a prerequisite for accurate functional layout grouping. A powerful language model alone cannot compensate for critical information loss occurring. Similarly, GUI agent methods show limited performance on this task, which is closely related to their design for interaction tasks. Among them, GUI-G^2 significantly outperforms V2P, suggesting greater adaptability to structural perception. However, both are constrained by predefined action spaces and lack direct modeling of the high-level semantics of functional layout grouping, making it difficult to accurately identify logical layouts within complex layouts.

An interesting observation is the consistent performance hierarchy maintained across both datasets. All methods except V2P exhibit superior performance on the LDT dataset compared to MUD, with GUIGROUP showing approximately 14% higher F1-score on LDT (0.5732) versus MUD (0.5041). This performance differential can be attributed to the more standardized and regular GUI layouts of popular applications in the LDT dataset, which align better with systematic grouping methodologies, whereas the MUD dataset contains

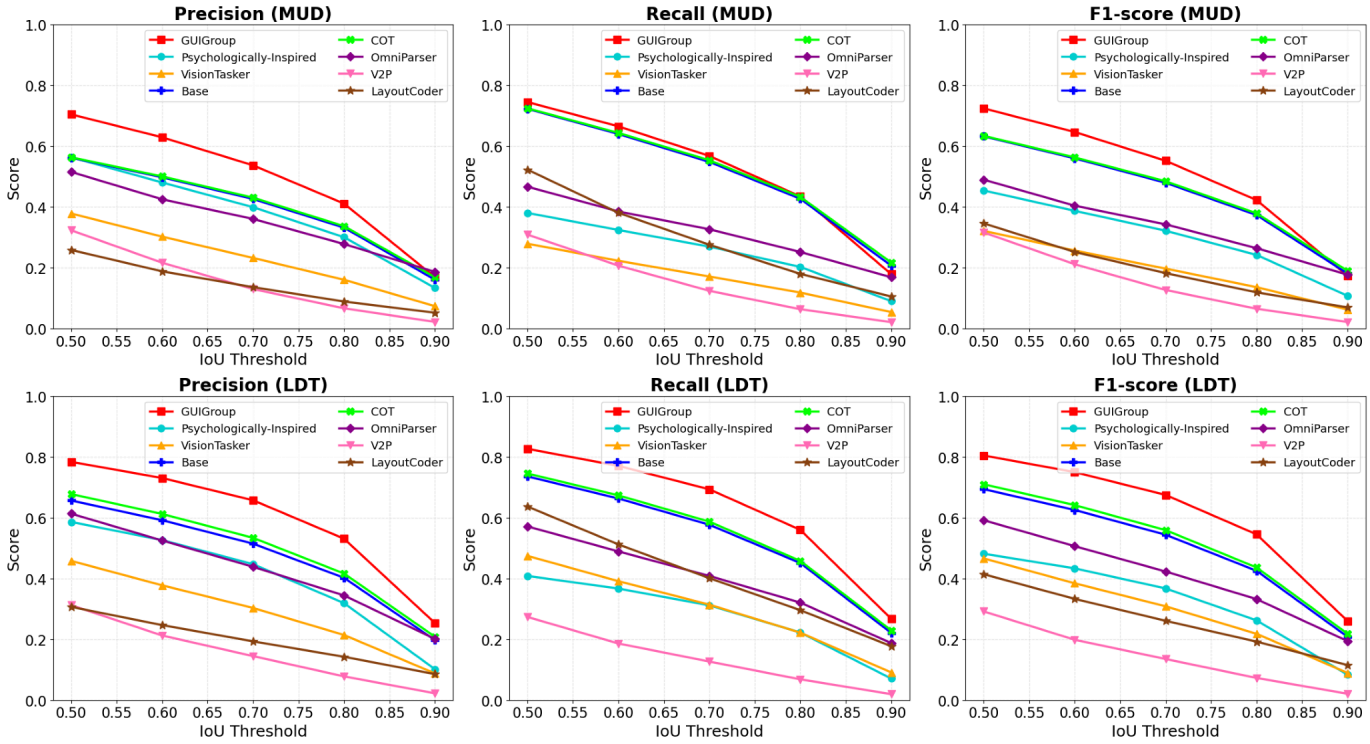


Fig. 4. Performance at different IoUs.

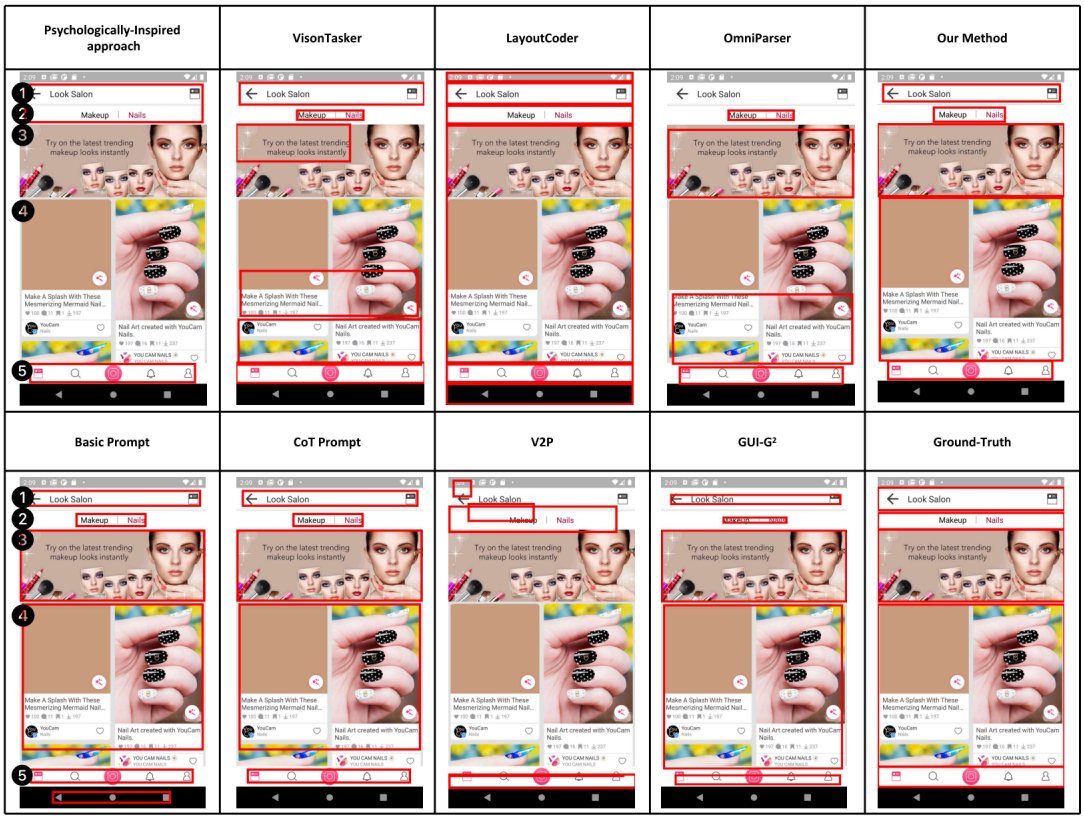


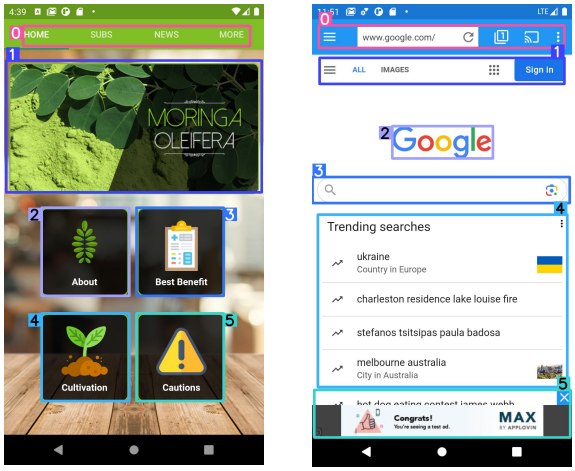
Fig. 5. Examples of functional layout grouping results and ground-truth (red box - grouping result).

more complex and irregular mobile interfaces that pose greater challenges for layout analysis algorithms.

Fig. 4 shows the performance analysis under different IoU thresholds. As the IoU threshold becomes stricter from 0.5, the performance indicators of all methods show a downward trend, which is in line with the general law of object detection tasks. On the LDT dataset, when the IoU threshold is 0.5, the precision, recall, and F1 score of GUIGROUP reach about 0.78, 0.79, and 0.78, respectively; even at the strictest IoU threshold of 0.9, it can still maintain a performance level of about 0.23, 0.24, and 0.24. This trend is consistent in both datasets.

Case Study. Fig. 5 shows our advantage through a concrete example. We found that almost all baselines have certain shortcomings. In this case, the accurate functional layout consists of five parts, numbered ❶ to ❺ in the figure. The Psychologically-Inspired approach only considers areas that are consistent with human cognition and have similar functions and designs, while ignoring areas that still have specific functions, such as the "advertising column" and "user display area" numbered ❶ and ❷ in the figure. Although VisionTasker uses a vision-based method in the recognition process to recognize the "advertising column" and "user display area", the functional layout it generates is incomplete and does not conform to human cognitive logic. LayoutCoder incorrectly includes the status bars at the top and bottom of the phone in the functional layout, and further incorrectly merges the "advertising bar" and "user display area" into one area. While OmniParser can partially identify the "user display area", its results are incomplete, and it completely omits the "top navigation bar" numbered ❶. In the method based on MLLM prompts, the simplest form of Basic Prompt incorrectly includes the device status bar in the functional layout. After introducing the CoT strategy, the overall results are more accurate, although they generally conform to the actual layout, their boundary coordinates still have some deviation, specially in layout ❹. In GUI agent based methods, V2P's recognition capability is weak, failing to accurately reconstruct any functional layouts. In contrast, GUI-G^2 only has minor errors at the coordinate level, such as the inaccurate height of the layout ❺. Our method not only generates functional layouts that are consistent with human cognition, but also minimizes missing functional widgets, achieving the closest match to the ground truth.

Error Analysis. We identified the main sources of error in the algorithm by manually checking the differences between the grouping results and the true annotations. The first is the over-segmentation problem. As shown in Fig. 6(a), the algorithm independently identifies the four plant knowledge base modules numbered ❷-❺. In fact, these four modules are consistent in functional semantics and visual style and should belong to the same functional layout. Although fine-grained segmentation is perceptually acceptable, it is considered a matching failure in the IoU calculation. The second is the layout occlusion problem. Fig. 6(b) shows that the bottom of the "trending searches" area numbered ❹ is occluded by the advertising module numbered ❺, resulting in an incomplete functional area. Through detailed examination of 50 randomly sampled error cases, we further validated these observations.



(a) Over-Segmentation (b) Layout Occlusion

Fig. 6. Typical causes of grouping mistakes: (a) Over-Segmentation (b) Layout Occlusion

TABLE II
THE PERFORMANCE COMPARISONS IN ABLATION STUDY(IOU@0.5:0.9)

Variant	Precision	Recall	F1
GUIGROUP	**0.5243±0.2010**	**0.5467±0.2189**	**0.5353±0.2143**
GUIGROUP-f	0.3529±0.1652	0.3627±0.1698	0.3577±0.1675
GUIGROUP-d	0.3816±0.1773	0.3791±0.1761	0.3804±0.1767
GUIGROUP-r	0.5230±0.2138	0.5405±0.2209	0.5316±0.2173
GUIGROUP-fr	0.3497±0.1650	0.3452±0.1626	0.3474±0.1637

Over-segmentation emerged as the most frequent issue, occurring in 25 instances where semantically coherent modules were incorrectly separated. Layout occlusion problems were identified in 18 cases, typically involving overlapping elements that compromise functional area integrity. The remaining 7 cases stemmed from other questions such as widget detection inaccuracies or semantic misinterpretation, confirming that over-segmentation and layout occlusion represent the primary challenges in functional layout grouping.

RQ1 GUIGROUP is more effective in functional layout grouping compared to all eight baseline methods. On the MUD dataset, GUIGROUP achieves F1 score improvements ranging from 7.90% to 239.6% , with an average improvement of 73.56% across all baselines. On the LDT dataset, the improvements range from 11.78% to 299.2%, with an average of 91.03%. This method maintains its advantage across different IoU thresholds (0.5-0.9), improving both precision and recall while generating layouts that are more consistent with human cognitive patterns.

B. RQ2-Ablation Study

1) Motivation: our method uses four steps–GUI widget detection, functional semantic analysis, coordinate positioning, and review for functional layout grouping. In this section, we explore the impact of these four different steps through

ablation experiments to ensure that they all contribute to the performance.

2) Method: We build four variants of GUIGROUP, which are as follows:

- **GUIGROUP-d** completely omits the GUI widget detection module. Considering that Claude-3.5-sonnet can identify widget coordinates at low resolution, it is provided with a low-resolution GUI screenshot scaled to 1024×768, which directly identifies and locates the functional layout.
- **GUIGROUP-f** removes the functional semantic analysis step and lets MLLM determine both the functional layout and its corresponding coordinates at the coordinate positioning stage.
- **GUIGROUP-r** removes the review step and does not perform any processing on missing widgets to evaluate its role on complex screenshots.
- **GUIGROUP-fr** removes both functional semantic analysis and review steps, preserving only GUI widget detection and coordinate positioning.

It is worth noting that since our evaluation requires specific coordinate information for the functional layout, the coordinate positioning step cannot be separated out for ablation.

3) Result: Table II presents the performance comparison of GUIGROUP and its variants on the functional layout grouping task on the whole dataset. From IoU threshold 0.5 to 0.9, all model variants demonstrate superior performance compared to the best baseline method in F1 score. Notably, even the most basic variant GUIGROUP-fr achieves an F1 score of 0.3474 representing a substantial 10.43% relative improvement over the baseline performance (0.3046), which underscores the significant advantage of MLLM-based approaches over widget detection methods.

Specific analysis of the performance of each variant: GUIGROUP-d drops to 0.3816, 0.3791 and 0.3804 in the IoU (0.5:0.9) range, which is significantly lower than the complete model. This significant decline reveals that widget detection serves as a critical "visual anchor" for the MLLM. Without precise widget coordinates, the MLLM struggles to ground its high-level semantic understanding onto specific image regions, leading to "floating" bounding boxes that fail to align with actual UI boundaries. The interaction between widget detection and MLLM inference is therefore essential: detection provides the spatial precision, while the MLLM provides the grouping logic. The indicators of GUIGROUP-f also showed a significant decline, with precision, recall and F1 score dropping to 0.3529, 0.3627 and 0.3577 respectively, indicating that functional semantic analysis is a key factor in performance. If MLLM cannot fully understand the semantics and contextual information of the functional layout, relying solely on coordinate positioning will seriously affect the effect. Without the functional semantic analysis module, the model cannot distinguish between visually adjacent but functionally distinct widgets. This proves that functional semantic analysis acts as a filter to refine coordinate-based proposals. It is worth noting that GUIGROUP-fr is lower than GUIGROUP-f in all metrics, which shows that even if the functional semantic analysis is not accurate enough, the review mechanism can still help correct errors and supplement the missing functional layouts. This is because the review module can provide corrective feedback, which can salvage performance even when the initial semantic features are weak.

The review mechanism shows modest improvements in aggregate IoU metrics (0.5:0.9), as the initial three-step pipeline already produces relatively complete functional layouts. Therefore, we conducted a fine-grained analysis to examine the review mechanism's specific contributions. Among 1,612 test images, 341 instances (21.2%) triggered the review process, with 317 cases (93.0%) resulting in modified functional layouts and 266 cases (78.0%) achieving IoU improvements. At the functional layout level, among 338 modified layouts that matched ground truth, 272 regions (80.4%) demonstrated IoU enhancements. The average IoU increased from 0.6722 to 0.7610, representing a substantial 13.21% improvement. These results indicate that while the review mechanism affects a limited subset of predictions, its impact on those specific cases is significant and consistently beneficial, serving as an effective refinement tool for addressing boundary ambiguities and edge cases.

In general, the precision, recall, and F1 score of the complete GUIGROUP model at each IoU threshold are better than its variant version, which fully proves the importance of the four steps.

> **RQ2** The design of each step of GUIGROUP can improve performance. GUI widget detection provides LLM with more accurate coordinate information, which helps to generate the functional layout accurately. Semantic function analysis ensures that LLM can identify the functional layout as perfectly as possible. The review step can effectively refine boundary precision for a selective subset of functional layouts.

C. RQ3-Usefulness

1) Motivation: Although the quantitative metric (IoU) validates the technical effectiveness of our functional layout grouping method, two key issues remain. First, as an overlap measure, IoU may not fully reflect the semantic correctness of functions. As shown in Fig. 6 (a), in cases of over-segmentation, although each sub-region may be semantically reasonable and meaningful, the IoU will score lower due to its pixel-level incomplete match with the ground truth. Second, the existing results do not yet answer a core question: Can our method truly support the daily tasks of UI designers and developers and improve the efficiency of human understanding of interface structure? To this end, we conducted a user study to evaluate the usability and practicality of the generated functional layout groupings in real-world work scenarios, thereby validating the actual effectiveness of the method from a human perspective.

2) Method: **Participants.** We recruited 30 participants from local institutions, all of whom had more than two years of front-end development experience. However, they were not familiar with the GUI tools used in the user study. We randomly divided these 30 participants into 5 experimental

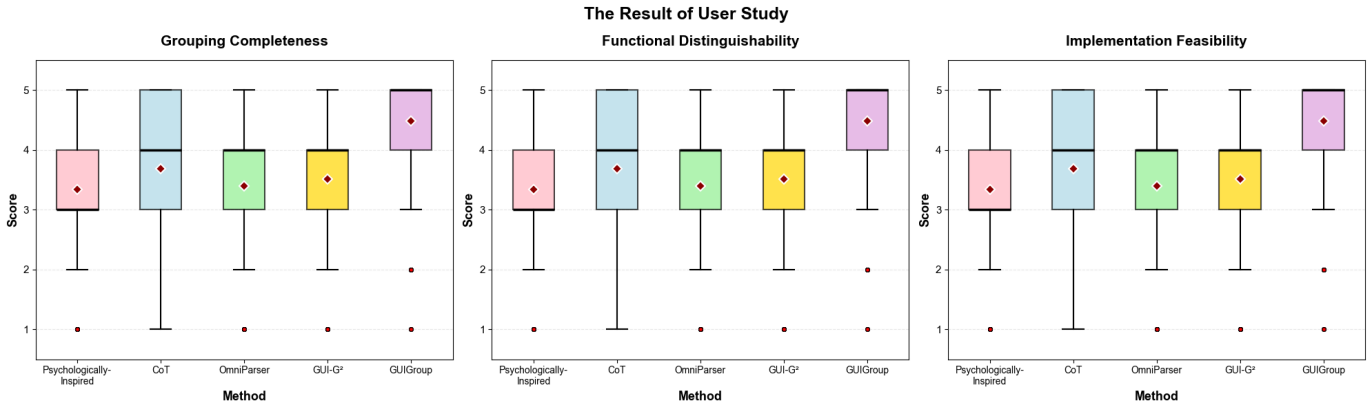


Fig. 7. User Study Results

TABLE III
GROUP TASK ALLOCATION IN THE USER STUDY

Group \ Screenshots	I_1	I_2	I_3	I_4	I_5
$G1$	A	B	C	D	E
$G2$	E	A	B	C	D
$G3$	D	E	A	B	C
$G4$	C	D	E	A	B
$G5$	B	C	D	E	A

groups (denoted as $G1$, $G2$, $G3$, $G4$ and $G5$), each containing 6 participants. The grouping process took into account the balanced distribution of participants' years of development experience and technical background to minimize the potential impact of inter-group differences on the experimental results.

Procedure. We randomly selected 50 GUI screenshots from LDT dataset and applied five functional layout grouping methods to them.

It is worth noting that, as categorized in Section III.A, existing approaches fall into four major paradigms. Given the large number of candidate methods, we selected the best-performing representative methods from each paradigm. These methods were evaluated based on IoU in our tests and were incorporated into user study. Therefore, the methods participating in our user study were Psychologically-Inspired, CoT, OmniParser, GUI-G^2, and our GUIGROUP. To minimize discrepancies caused by presentation methods, all tool outputs are presented in a consistent visual format, such as a uniform color palette, line thickness, and overlay styles. This ensures that differences in perceived quality stem from grouping logic, rather than aesthetics or interface flaws.

Next, we divided the screenshots into five sets, labeled as I_1, I_2, I_3, I_4 and I_5. Each set has four screenshots. We referred to the Psychologically-Inspired, CoT, OmniParser, GUI-G^2, and our GUIGROUP as A, B, C, D and E, respectively. Accordingly, we denoted the image set I_i processed by tool X ($X = A, B, C, D, E$) as I_{iX}. Table III presents the task assignments for each group in the user study.

We employ a 5×5 Latin square design to assign methods to image sets across the five participant groups. For example, participants in $G1$ were assigned screenshots containing I_{1A}, I_{2B}, I_{3C}, I_{4D} and I_{5E}. This design ensures that participants will not see the results of multiple tools processing the same original screenshot, thereby avoiding the judgment bias caused by direct comparison. It also ensures balanced evaluation across image sets. Each screenshot processed by a tool is evaluated by five participants. Participants will use a 5-point Likert scale (1 is the worst and 5 is the best) to score at three perspectives: *grouping completeness*, *functional distinguishability*, and *implementation feasibility*. *Grouping completeness* refers to whether the groups detected by a tool can cover all relevant functional layouts without omissions. *Functional distinguishability* refers to whether the functional positioning and purpose of each block of the screenshot can be clearly interpreted and differentiated. *Implementation feasibility* evaluates how straightforward it is to develop or reconstruct the code based on the identified functional layouts.

3) Result: Fig. 7 shows the results of our user study. GUIGroup (purple) consistently outperforms all baseline methods, achieving the highest median scores (5.0) and mean scores (4.52, 4.49, and 4.42, marked by red diamonds) across *Grouping Completeness*, *Functional Distinguishability*, and *Implementation Feasibility*. For these three key metrics, our method represents improvements of 20.61%, 20.29%, and 24.49% over the best baseline, respectively. To rigorously validate these observations, we conducted a non-parametric Friedman test based on the mean scores of the 30 participants. The results revealed statistically significant differences among the evaluated methods across all three metrics ($\chi^2(4) \geq 63.22$, $p < 0.001$), with large effect sizes (Kendall's $W \geq 0.52$). Post-hoc pairwise comparisons using the Wilcoxon signed-rank test with Bonferroni correction confirmed that GUIGroup significantly outperforms all baseline methods ($p < 0.001$ for all pairs). Furthermore, the effect sizes (r) for the comparisons between GUIGroup and the baselines were consistently large ($r \in [0.840, 0.874]$), underscoring the substantial practical advantage of our approach. The box plots demonstrate that GUIGroup exhibits the most compact distribution with the smallest interquartile ranges, indicating high consistency in user ratings and strong agreement among participants. Most ratings concentrate in the 4-5 range with minimal outliers. In contrast, the CoT (blue) distribution is the most dispersed, with a large number of low-scoring outliers, indicating significant differences in user ratings. CoT (blue) performed well across all three metrics, but the scores varied significantly, indicating potential inconsistencies in user understanding. OmniParser

TABLE IV
EFFICIENCY AND SCALABILITY OF FOUR-STEP GUI FUNCTIONAL LAYOUT GROUPING

Processing Stage	Processing Time (s)	Token Numbers
GUI widget detection (text)	1.3	–
GUI widget detection (non-text)	0.8	–
Functional semantic analysis	10.8	2,559
Coordinate localization	9.0	4,472
Review	28.2	14,265
Total sequential processing	**50.1**	**21,296**

(green), Psychologically-Inspired (pink) and GUI-G^2 (yellow) approaches demonstrate moderate performance. It is worth noting that although there are differences in the specific scores of the five groups of participants (some groups have higher overall scores, while others are relatively conservative), all groups agree that our method performs best. This consistent ranking of advantages and disadvantages indicates that the performance differences between tools are objective, rather than the subjective preferences of a specific evaluation group. At the same time, this shows that our method can effectively help front-end developers generate more complete, clear-featured, and practical page widget grouping solutions. The main reason is that our method can more accurately identify the functional attributes of different layouts of the screenshot, reduce widget omissions and incorrect groupings, and improve the recognition between different functional layouts.

RQ3 The user study validated that our method for grouping functional layouts can bring tangible benefits to real practitioners. The method performed well on all three evaluation dimensions. All participants unanimously recognized the superiority of our method.

V. DISCUSSION

A. Efficiency and Scalability

Performance efficiency is critical for the practical deployment of our method, especially when processing large amounts of GUI data. Our method contains four main processing stages, each with different performance characteristics. Table IV shows the efficiency and scalability statistics of GUIGROUP, including the average time required to execute each step and the average number of tokens consumed. The front-end GUI widget detection stage is relatively lightweight, with text and non-text detection taking only 1.3 seconds and 0.8 seconds, respectively. The functional semantic analysis and coordinate location stages take similar time (10.8 seconds and 9.0 seconds, respectively), but the token consumption is increasing (from 2,559 to 4,472). This growth pattern reflects that the results of functional semantic analysis are used as context information to feed the MLLM in the coordinate location stage, which requires the latter to process richer semantic information. The most noteworthy is the review stage, which takes 28.2 seconds and consumes 14,265 tokens alone. This is because the review stage requires a detailed analysis of widgets that may have been lost in the previous processing. Overall, the complete GUI analysis process takes 50.1 seconds to process 21,296 tokens. , it is crucial to clarify that this 50.1 seconds duration represents the worst-case sequential execution. According to our statistics, the highly time-consuming Review stage is only triggered in 21.2% of complex cases where the initial functional groupings are deemed ambiguous. For the vast majority of screens, the layout is successfully resolved without triggering this final stage, significantly reducing the average processing time. Although the current latency is acceptable for offline analysis scenarios with a limited number of critical screens, we acknowledge that practical deployment for large-scale GUI analysis or real-time design assistance requires concrete optimization strategies. To address this, we propose several practical solutions for future deployment: 1) Enhancing Throughput for Large-Scale Analysis: To mitigate the latency of commercial APIs, we propose deploying lightweight local MLLMs to eliminate network overhead. Additionally, independent screenshot analyses can be executed in parallel batches to significantly boost processing throughput for large datasets. 2) Optimizing Latency for Real-Time Tools: To ensure a smooth user experience in interactive applications, we propose an asynchronous two-stage pipeline. The system instantly returns preliminary layouts from the fast initial stages, while the intensive Review stage runs asynchronously in the background, updating the UI with refined suggestions only if necessary.

TABLE V
THE PERFORMANCE COMPARISONS BETWEEN DIFFERENT LANGUAGE MODELS(IOU@0.5:0.9)

Version	Precision	Recall	F1
Claude-3.5-sonnet	**0.5282±0.2125**	**0.5640±0.2269**	**0.5455±0.2194**
GPT-4o	0.4768±0.2031	0.4768±0.2301	0.4768±0.2031
Qwen2.5-VL-72B	0.5176±0.1736	0.5007±0.1679	0.5090±0.1707

B. Model Adaptability and Performance

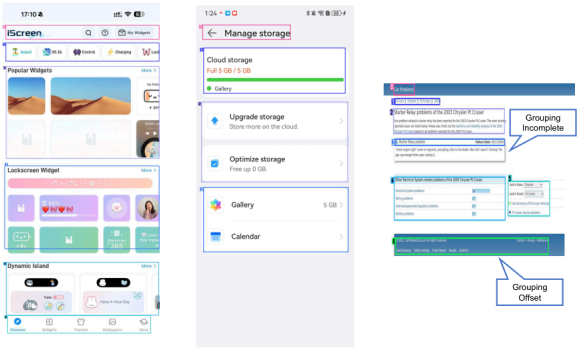
In GUIGROUP, the core capabilities come from the image understanding and reasoning capabilities of MLLMs. To systematically verify the cross-model adaptability of the method, we tested GPT-4o and Qwen2.5VL-72B on 100 images randomly selected from the MUD and LDT datasets, respectively, replacing the original Claude-3.5-sonnet. The results show that there are significant differences in the performance of different MLLMs. Table V shows the performance of different multimodal large language models on the dataset. Claude-3.5-sonnet performs best, demonstrating its ability to integrate image understanding with coordinate-based reasoning. The results of GPT-4o and Qwen2.5VL-72B are both quite competitive, verifying the robustness and cross-model generalization ability of our method. The performance gap between the models stems from their differences in their ability to integrate visual understanding with structured information.

C. Functional layout recognition application

Modular UI2code is the area that benefits most directly. By identifying interface areas with similar functions and

TABLE VI
CROSS-PLATFORM PERFORMANCE COMPARISON(IOU@0.5:0.9)

Platfrom	Precision	Recall	F1
iOS	0.5344±0.1944	0.6057±0.2203	0.5679±0.2066
HarmonyOS	0.5935±0.2567	0.5615±0.2429	0.5771±0.2496
Web	0.5153±0.2024	0.4400±0.1728	0.4747±0.1864



(a) iOS (b) HarmonyOS (c)Web

Fig. 8. Examples from different platforms: (a) iOS (b) HarmonyOS (c) Web

understanding their structural relationships, we are able to generate modular code that is more in line with development practices. For repeated elements, the system can identify their commonalities and generate loop structures, which significantly reduces code redundancy and solves a major problem of image generation code [29]–[31]. At the same time, our method establishes clear mappings between interface elements and code snippets, enabling developers to easily understand and modify generated code. This structured understanding improves code quality and provides a practical foundation for automated code generation [21], [32]. In addition, in terms of user experience design and evaluation, our method provides a tool for objectively analyzing the functional layout of GUIs. Designers can use the functional layout detection results to evaluate the intuitiveness and usability of the interface and identify potential design problems, such as overcrowding of functional layouts or difficulty in finding key functions. This data-based design evaluation method can supplement traditional user research and provide a more comprehensive basis for design optimization.

D. Adapting to other platforms

Our method has good cross-platform adaptability in theory, and this feature deserves further discussion. Although modern user interfaces are presented in different forms on different platforms, the semantic expressions of their functional layouts are essentially similar. For mobile devices, although Android and iOS have significant differences in underlying architecture and design language, both follow similar interaction patterns and functional organization principles [33]. Tablet devices usually maintain a similar functional structure to mobile phones, but only make adaptive adjustments in spatial layout [34]. Although desktops tend to use more compact icons and less descriptive text, their functional organization logic remains consistent with other platforms [35].

To empirically validate the cross-platform adaptability of our method, we conducted a pilot study on 20 representative GUI screenshots sampled from each of three additional platforms: iOS, Web, and HarmonyOS. As summarized in Table VI, our method maintains robust performance on mobile-centric platforms, achieving an F1-score of 0.5679 on iOS and 0.5771 on HarmonyOS. We attribute this to two factors: first, these platforms share similar interaction patterns and widget proportions with Android, ensuring reliable widget detection; second, the underlying MLLM inherently possesses cross-platform knowledge, enabling it to understand the subtle stylistic differences across mobile operating systems. As shown in Fig. 8(a)(b), the predicted functional layout groupings on iOS and HarmonyOS have only minor errors.

However, a noticeable performance drop is observed on Web interfaces (F1-score: 0.4747). As hypothesized in our earlier discussion, this degradation is primarily attributable to the limitations of our widget detector rather than the semantic reasoning capability of the MLLM. Since the detector was trained exclusively on mobile GUIs, it struggles with the denser layouts, larger aspect ratios, and more different component scales typical of Web environments, often missing widgets or producing incomplete bounding boxes. As illustrated in Fig. 8(c), while the functional grouping of Box ❻ is semantically correct, a positional offset occurred due to a widget detection error; additionally, the lower portion of Box ❸ was entirely missed as the corresponding widgets were not detected. These findings suggest that replacing the mobile-specific detector with a more general-purpose widget detection model would be a promising direction to improve Web performance in future work.

E. Threat to Validity

Internal Validity We face internal validity threats related to dataset annotation. The first threat comes from subjective judgment differences in the annotation process. Different annotators may have different understandings of the classification and layout boundaries of the functional layouts of the same GUI screenshot, especially when dealing with complex images. Although we have developed detailed annotation guidelines and implemented a two-person cross-validation mechanism, the fatigue and attention fluctuations of the annotators may still affect the annotation quality. The second threat comes from the output uncertainty of MLLM, which may produce hallucinations and lead to inaccurate functional parsing results. To mitigate this threat, we conduct three independent experiments on each test case and adopt a majority voting mechanism. The third threat is that the IoU score used for functional layout similarity evaluation may not intuitively reflect the accuracy of GUI functional layout recognition, especially when dealing with differences in users' understanding of the GUI. We mitigate this threat through a user study (detailed in Section IV.C) that validates the correlation between our IoU-based metrics and human perception of functional layout similarity.

External Validity One potential limitation is that, although our research dataset has been carefully constructed to cover interface samples from different application areas, it may not

fully represent all possible GUI design variations in the real world. Although we have worked hard to collect the latest GUI designs of popular applications, including some emerging design patterns and innovative layouts, considering the rapid iteration and continuous innovation of GUI design, our dataset may not be able to fully cover all possible design patterns.

F. Limitation

The evaluation in this work is limited to standard mobile user interfaces composed of common widget types (e.g., buttons, text fields, lists), which constitute the majority of real-world UI automation scenarios. The proposed method has not been tested on more complex interface patterns, such as custom or rare widgets, multi-panel layouts, dynamic UI states, or deeply nested hierarchical structures. Although the framework is designed to support hierarchical parsing and semantic reasoning via MLLMs—suggesting potential extensibility to such cases—its effectiveness on these challenging configurations remains unverified. Future work will explore extensions using state-aware screenshot sequences and enhanced prompting strategies to address these scenarios.

VI. RELATED WORK

In this section, we present the two most relevant aspects of our work, namely GUI widget grouping and MLLMs on GUI visual understanding.

A. GUI Widget Grouping

Most existing studies focus on grouping GUI to enhance human comprehension of interface layouts. UI widget grouping can be categorized into three hierarchical types [36]: layout-level grouping [3], component-level grouping [5], [37], [38], and element-level grouping [36], [39], [40]. Element-level grouping operates at the finest granularity, merging fragmented visual elements into coherent basic components. For example, it can group scattered icons, each of which is fragmented but actually constitutes a button. UILM [39] automatically detects and merges fragmented layers through view hierarchy design and visual learning. Chen et al. [36] achieve end-to-end automatic grouping through UI sequence prediction. Component-level grouping identifies GUI widgets with independent semantics. Most component recognition research is based on this level [17], [41]. For example, REMAUI [42] combines OCR and computer vision techniques to automatically convert screenshots or concept designs to executable mobile app interfaces. UIED [37], [38] uses a hybrid method of traditional computer vision methods and deep learning methods to detect and group GUI widgets.

Layout-level grouping divides the GUI into functional areas based on human perception, which operates at the same granularity as our work. The work of Xie et al. [3] is most similar to our study. They grouped discrete components into perceptual groups using Gestalt psychology principles, but only focused on similar elements and ignored functional diversity. Song et al. [25] proposed a probability-based block partitioning method that exploited gradient information for boundary detection.

Recently, layout-level grouping has become an essential step in UI-to-code generation tasks. Methods such as LayoutCoder [26] and DCGen [32] first segment GUI screenshots into layouts before generating corresponding code structures. However, these approaches primarily rely on visual boundaries rather than semantic functional understanding. In contrast, our work segments GUIs into functional layouts by considering both visual coherence and the semantic purpose of UI components, enabling more accurate layout identification that aligns with user perception and functional intent. This distinction is particularly important for tasks requiring functional organization understanding.

B. MLLMs on GUI visual understanding

Multimodal large language models have shown great potential in recognizing and understanding images. MLLMs such as Claude [8], GPT-4V [6], and QWen2.5-VL [7] can understand and analyze images based on questions provided by users. However, they cannot generate specific GUI widget coordinates or overall layout information [9]. To address these limitations, methods that enhance MLLM image understanding with visual cues have emerged. Yang et al. [12] proposed a "label set" approach to improve the localization ability of GPT-4V by adding region labels. By generating structured bounding boxes and labels from UI screenshots, the action prediction performance of GPT-4V can be improved to various user tasks, enabling more accurate responses through targeted cues [19].

Building on this foundation, recent research has improved the localization ability of MLLM-based GUI agents [10], [43], [44]. SeeClick [45] is pre-trained with GUI ground truth data from web and mobile platforms for element localization. Ferret-UI [9] introduces "arbitrary resolution" capabilities to enhance granularity of detail across multiple platforms, and Ferret-UI 2 further improves on this [46]. Beyond element localization, recent work has begun to address spatial reasoning in GUI contexts. GUI-G^2 [27] models GUI elements as continuous Gaussian distributions on the interface plane, establishing a new paradigm for spatial reasoning in GUI interaction tasks. V2P [28] introduces a suppression attention mechanism that minimizes the model's focus on irrelevant regions to highlight the intended region. However, these methods primarily focus on component-level localization for interaction tasks, with limited attention to layout-level segmentation that captures the hierarchical structure and functional organization of GUI components. [13]

While these MLLM-based approaches have advanced GUI understanding, they differ from our work in key aspects. First, existing methods primarily focus on element-level or component-level understanding for interaction tasks, whereas our approach targets layout-level segmentation for layout comprehension. Second, prior works either rely on visual detection alone or use GUI agents for direct element localization based on visual appearance or actionable properties. In contrast, we combine visual detection with MLLM-based semantic reasoning to identify layouts based on their functional purposes rather than just visual appearance or actionable properties. Third, our approach provides both layout-level and component-level

understanding, bridging the gap between coarse layout analysis and fine-grained widget detection.

VII. CONCLUSION

In this paper, we propose a novel functional layout grouping method named GUIGROUP, which combines traditional widget detection technology with MLLM. Our method obtains basic interface element information through widget detection, then uses MLLM to perform functional semantic analysis and coordinate positioning, and finally obtains complete functional layout detection results through review. We collected GUI screenshots from two high-quality datasets and constructed a GUI functional layout dataset through manual annotation. Experimental results show that our method significantly outperforms all baseline methods in F1 score, while also detecting other functional layouts that cannot be identified by baselines. Future work will extend our method to apply to dynamic interaction analysis, leveraging advanced MLLM capabilities to enhance GUI code generation and automated testing.

ACKNOWLEDGMENT

This work is supported by the Zhejiang Pioneer (Jianbing) Project (2025C01198), the National Science Foundation of China (No. 62372398, No. 72342025), Zhejiang Provincial Science Foundation of China (No. LQK26F020002), and the Fundamental Research Funds for the Central Universities (No. 226-2025-00067).

DATA AVAILABILITY

Our code and experimental results are available at https://github.com/memoryzyl/GUIGroup.

REFERENCES

[1] B. Frost, *Atomic design*. Brad Frost Pittsburgh, 2016.
[2] X. Zhang, L. De Greef, A. Swearngin, S. White, K. Murray, L. Yu, Q. Shan, J. Nichols, J. Wu, C. Fleizach *et al.*, "Screen recognition: Creating accessibility metadata for mobile applications from pixels," in *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, 2021, pp. 1–15.
[3] M. Xie, Z. Xing, S. Feng, X. Xu, L. Zhu, and C. Chen, "Psychologically-inspired, unsupervised inference of perceptual groups of gui widgets from gui images," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2022. New York, NY, USA: Association for Computing Machinery, 2022, p. 332–343. [Online]. Available: https://doi.org/10.1145/3540250.3549138
[4] Thalion, "Ui design in practice: Gestalt principles," Feb 2020. [Online]. Available: https://uxmisfit.com/2019/04/23/ui-design-in-practice-gestalt-principles/
[5] S. Xiao, Y. Chen, Y. Song, L. Chen, L. Sun, Y. Zhen, Y. Chang, and T. Zhou, "Ui semantic component group detection: Grouping ui elements with similar semantics in mobile graphical user interface," *Displays*, vol. 83, p. 102679, 2024.
[6] OpenAI, "Gpt-4v(ision) system card," 2023.
[7] P. Wang, S. Bai, S. Tan, S. Wang, Z. Fan, J. Bai, K. Chen, X. Liu, J. Wang, W. Ge *et al.*, "Qwen2-vl: Enhancing vision-language model's perception of the world at any resolution," *arXiv preprint arXiv:2409.12191*, 2024.
[8] Anthropic, "The claude 3 model family: Opus, sonnet, haiku," 2024.
[9] K. You, H. Zhang, E. Schoop, F. Weers, A. Swearngin, J. Nichols, Y. Yang, and Z. Gan, "Ferret-ui: Grounded mobile ui understanding with multimodal llms," in *European Conference on Computer Vision*. Springer, 2024, pp. 240–255.
[10] B. Gou, R. Wang, B. Zheng, Y. Xie, C. Chang, Y. Shu, H. Sun, and Y. Su, "Navigating the digital world as humans do: Universal visual grounding for gui agents," *arXiv preprint arXiv:2410.05243*, 2024.
[11] B. Zheng, B. Gou, J. Kil, H. Sun, and Y. Su, "Gpt-4v (ision) is a generalist web agent, if grounded," *arXiv preprint arXiv:2401.01614*, 2024.
[12] J. Yang, H. Zhang, F. Li, X. Zou, C. Li, and J. Gao, "Set-of-mark prompting unleashes extraordinary visual grounding in gpt-4v," *arXiv preprint arXiv:2310.11441*, 2023.
[13] S. Nayak, X. Jian, K. Q. Lin, J. A. Rodriguez, M. Kalsi, R. Awal, N. Chapados, M. T. Özsu, A. Agrawal, D. Vazquez *et al.*, "Ui-vision: A desktop-centric gui benchmark for visual perception and interaction," *arXiv preprint arXiv:2503.15661*, 2025.
[14] T. Luo, L. Logeswaran, J. Johnson, and H. Lee, "Visual test-time scaling for gui agent grounding," *arXiv preprint arXiv:2505.00684*, 2025.
[15] S. Feng, S. Ma, H. Wang, D. Kong, and C. Chen, "Mud: Towards a large-scale and noise-filtered ui dataset for modern style ui modeling," in *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, 2024, pp. 1–14.
[16] P. Authors, "Paddleocr, awesome multilingual ocr toolkits based on paddlepaddle." https://github.com/PaddlePaddle/PaddleOCR, 2020.
[17] R. Varghese and M. Sambath, "Yolov8: A novel object detection algorithm with enhanced performance and robustness," in *2024 International Conference on Advances in Data Engineering and Intelligent Computing Systems (ADICS)*. IEEE, 2024, pp. 1–6.
[18] R. Liu, X. Teoh, Y. Lin, G. Chen, R. Ren, D. Poshyvanyk, and J. S. Dong, "Guipilot: A consistency-based mobile gui testing approach for detecting application-specific bugs," *Proceedings of the ACM on Software Engineering*, vol. 2, no. ISSTA, pp. 753–776, 2025.
[19] Y. Lu, J. Yang, Y. Shen, and A. Awadallah, "Omniparser for pure vision based gui agent," *arXiv preprint arXiv:2408.00203*, 2024.
[20] Z. Bai, P. Wang, T. Xiao, T. He, Z. Han, Z. Zhang, and M. Z. Shou, "Hallucination of multimodal large language models: A survey," *arXiv preprint arXiv:2404.18930*, 2024.
[21] T. Zhou, Y. Zhao, X. Hou, X. Sun, K. Chen, and H. Wang, "Declarui: Bridging design and development with automated declarative ui code generation," *Proceedings of the ACM on Software Engineering*, vol. 2, no. FSE, pp. 219–241, 2025.
[22] B. Deka, Z. Huang, C. Franzen, J. Hibschman, D. Afergan, Y. Li, J. Nichols, and R. Kumar, "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*, 2017, pp. 845–854.
[23] S1T2, "Apply crap to design: S1t2 blog," Aug 2021. [Online]. Available: https://s1t2.com/blog/step-1-generously-apply-crap-to-design

[24] G. Li, G. Baechler, M. Tragut, and Y. Li, "Learning to denoise raw mobile ui layouts for improving datasets at scale," in *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, 2022, pp. 1–13.

[25] Y. Song, Y. Bian, Y. Tang, G. Ma, and Z. Cai, "Visiontasker: Mobile task automation using vision based ui understanding and llm task planning," in *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology*, 2024, pp. 1–17.

[26] F. Wu, C. Gao, S. Li, X.-C. Wen, and Q. Liao, "Mllm-based ui2code automation guided by ui layout information," *Proceedings of the ACM on Software Engineering*, vol. 2, no. ISSTA, pp. 1123–1145, 2025.

[27] F. Tang, Z. Gu, Z. Lu, X. Liu, S. Shen, C. Meng, W. Wang, W. Zhang, Y. Shen, W. Lu *et al.*, "Gui-g ^{2}: Gaussian reward modeling for gui grounding," *arXiv preprint arXiv:2507.15846*, 2025.

[28] J. Chen, L. Chen, D. Wang, L. Gan, C. Zhuang, and J. Gu, "V2p: From background suppression to center peaking for robust gui grounding task," *arXiv preprint arXiv:2508.13634*, 2025.

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

[30] C. Chen, T. Su, G. Meng, Z. Xing, and Y. Liu, "From ui design image to gui skeleton: a neural machine translator to bootstrap mobile gui implementation," in *Proceedings of the 40th International Conference on Software Engineering*, 2018, pp. 665–676.

[31] S. Feng, S. Ma, J. Yu, C. Chen, T. Zhou, and Y. Zhen, "Auto-icon: An automated code generation tool for icon designs assisting in ui development," in *Proceedings of the 26th International Conference on Intelligent User Interfaces*, 2021, pp. 59–69.

[32] Y. Wan, C. Wang, Y. Dong, W. Wang, S. Li, Y. Huo, and M. R. Lyu, "Automatically generating ui code from screenshot: A divide-and-conquer-based approach," *arXiv preprint arXiv:2406.16386*, 2024.

[33] Y. Gao, X. Hu, T. Xu, X. Xia, and X. Yang, "A rule-based approach for ui migration from android to ios," *arXiv preprint arXiv:2409.16656*, 2024.

[34] H. Zhan, Y. Huang, D. Liu *et al.*, "Pairwise gui dataset construction between android phones and tablets," *Advances in Neural Information Processing Systems*, vol. 36, pp. 59 860–59 872, 2023.

[35] Y. Xu, L. Yang, H. Chen, H. Wang, Z. Chen, and Y. Tang, "Deskvision: Large scale desktop region captioning for advanced gui agents," *arXiv preprint arXiv:2503.11170*, 2025.

[36] L. Chen, Y. Chen, S. Xiao, Y. Song, L. Sun, Y. Zhen, T. Zhou, and Y. Chang, "Egfe: End-to-end grouping of fragmented elements in ui designs with multimodal learning," in *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*, 2024, pp. 1–12.

[37] M. Xie, S. Feng, Z. Xing, J. Chen, and C. Chen, "Uied: a hybrid tool for gui element detection," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 1655–1659.

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

[39] Y. Chen, Y. Zhen, C. Shi, J. Li, L. Chen, Z. Li, L. Sun, T. Zhou, and Y. Chang, "Ui layers merger: merging ui layers via visual learning and boundary prior," *Frontiers of Information Technology & Electronic Engineering*, vol. 24, no. 3, pp. 373–387, 2023.

[40] J. Li, T. Zhou, Y. Chen, Y. Chang, Y. Zhen, L. Sun, and L. Chen, "Uldgnn: A fragmented ui layer detector based on graph neural networks," *arXiv preprint arXiv:2208.06658*, 2022.

[41] C. Yongxin, Z. Tonghui, and C. Jie, "Ui2code: How to fine-tune background and foreground analysis," *Retrieved Feb*, vol. 23, p. 2020, 2019.

[42] T. A. Nguyen and C. Csallner, "Reverse engineering mobile application user interfaces with remaui (t)," in *2015 30th IEEE/ACM international conference on automated software engineering (ASE)*. IEEE, 2015, pp. 248–259.

[43] W. Hong, W. Wang, Q. Lv, J. Xu, W. Yu, J. Ji, Y. Wang, Z. Wang, Y. Dong, M. Ding *et al.*, "Cogagent: A visual language model for gui agents," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 14 281–14 290.

[44] Y. Qin, Y. Ye, J. Fang, H. Wang, S. Liang, S. Tian, J. Zhang, J. Li, Y. Li, S. Huang *et al.*, "Ui-tars: Pioneering automated gui interaction with native agents," *arXiv preprint arXiv:2501.12326*, 2025.

[45] K. Cheng, Q. Sun, Y. Chu, F. Xu, Y. Li, J. Zhang, and Z. Wu, "Seeclick: Harnessing gui grounding for advanced visual gui agents," *arXiv preprint arXiv:2401.10935*, 2024.

[46] Z. Li, K. You, H. Zhang, D. Feng, H. Agrawal, X. Li, M. P. S. Moorthy, J. Nichols, Y. Yang, and Z. Gan, "Ferret-ui 2: Mastering universal user interface understanding across platforms," *arXiv preprint arXiv:2410.18967*, 2024.