

CrowdGenUI: Enhancing LLM-Based UI Widget Generation with a Crowdsourced Preference Library

YIMENG LIU*, University of California, Santa Barbara, USA

MISHA SRA, University of California, Santa Barbara, USA

CHANG XIAO, Adobe Research, USA

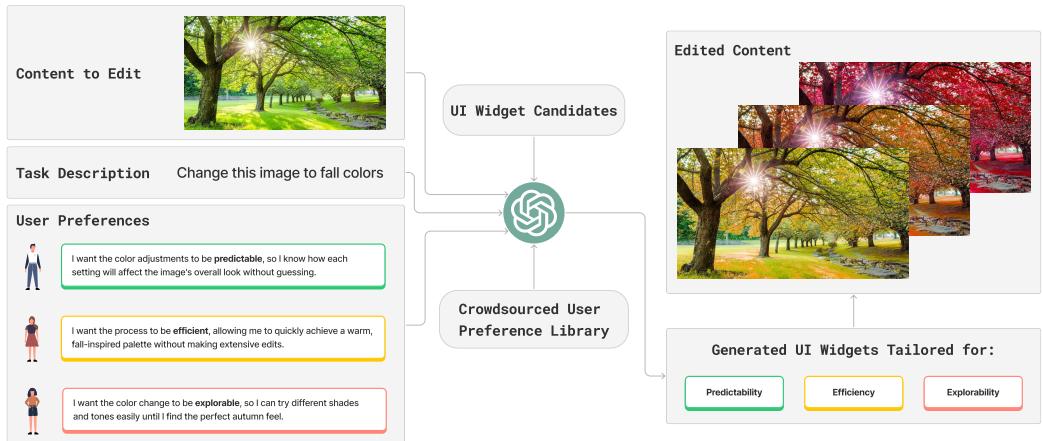


Fig. 1. *CrowdGenUI* is a framework that allows the generation of task requirement-aware and user preference-aligned UI widgets powered by LLMs. Based on input information specifying the content to edit, the task to perform, and various user preferences, the LLM-driven UI widget generation is enhanced by our crowdsourced user preference library together with a set of widget candidates for UI widget reasoning and code generation.

Large Language Models (LLMs) have demonstrated remarkable skills across various design domains, including UI generation. However, current LLMs for UI generation tend to offer generic solutions that lack a deep understanding of task context and user preferences in specific scenarios. We present *CrowdGenUI*, a framework that enhances LLM-driven UI generation with a crowdsourced user preference library. This approach addresses the limitations of existing methods by guiding LLM reasoning with user preferences, enabling the generation of UI widgets that align more closely with user needs and task-specific requirements. Using image editing as a test domain, we built this library from 50 users, capturing 720 user preferences, which include the predictability, efficiency, and explorability of multiple UI widgets. In a user study with 72 additional participants, our framework outperformed standard LLM-generated widgets in meeting user preferences and task requirements. We discuss these findings to inform future opportunities for designing user-centered and customizable UIs by comprehensively analyzing the extendability of the proposed framework and crowdsourced library.

CCS Concepts: • Human-centered computing → Collaborative and social computing; Graphical user interfaces; User studies; • Information systems → Crowdsourcing.

Additional Key Words and Phrases: user preference, UI generation, large language models

*Work done during an internship at Adobe Research

1 Introduction

Graphical user interfaces (GUIs) play a fundamental role in how most people operate computers and other computing devices. Carefully designed by expert interface designers, these GUIs aim to provide users with learnable ways to interact with software [2, 3]. Although effective, such interfaces can be complicated and sometimes overwhelming for users. In recent years, the emergence of large language models (LLMs) has led to increased interest in natural language interfaces. These language-based interfaces process human language input, analyze high-level intentions, and turn the interpreted intents into operations, providing a simplified way for users to interact with technology [21, 35, 38]. However, while language interfaces are promising, they are typically suited for high-level task execution, such as removing an object from an image or adding a filtering effect to videos. They often lack the precision required when users need to fine-tune specific parameters, such as adjusting the color in an image or customizing detailed settings in a video editing project.

To address the complexity of professional GUIs and the inaccuracy of natural language interfaces, researchers have begun to explore the integration of LLMs with UI generation to combine the high-level task understanding enabled by LLMs with the precise, granular control offered by GUIs [19, 49]. By turning natural language inputs into interactable UI widgets through UI generation, these systems provide an exciting hybrid interaction model that supports both user interaction with widgets for precise control and language as input for high-level task interpretation. Nevertheless, the existing UI generation systems also come with challenges. These LLM-generated UIs tend to be generic and may not fully account for the nuances of domain-specific tasks or the diverse preferences of individual users. Without specialized knowledge of interface design or a deeper understanding of user preferences, these UIs may fail to provide the optimal interaction experience.

To overcome these challenges, we propose a novel framework, *CrowdGenUI*, that enhances LLM-driven UI generation by incorporating a crowdsourced user preference library, as illustrated in Figure 1. This framework leverages the collective insights of users to guide the LLM in generating tailored UI widgets for different user preferences. While the framework is broadly applicable across various domains, we demonstrate its effectiveness in the context of image editing within a Python programming environment. To build the preference library, we collected data from 50 users, with no requirement on UI design or programming experience, capturing their preferences across three key dimensions of interface design, i.e., *predictability*, *efficiency*, and *explorability*. By integrating this library into the LLM’s UI widget generation process, our system generates widgets that are not only more precise but also better aligned with user preferences and task-specific requirements.

In our user study with a different set of 72 participants, we found that UI widgets generated based on our proposed framework were more aligned with user preferences and task requirements than those generic widgets generated by the LLM alone. This demonstrates the potential of our framework to improve LLM-based UI generation and suggests that crowdsourced user preference data can play a crucial role in creating user-centered user interfaces.

To summarize, this paper makes the following key contributions:

- We propose a general framework to enhance LLM-driven UI generation by incorporating a crowdsourced user preference library for task-aware and preference-aligned widget generation.
- As a concrete implementation of this framework, we develop a system for the domain of image editing within a Python programming environment. This system includes crowdsourced data from 50 users, capturing user preferences for UI widget configurations in image editing tasks.
- We conduct a user study with an additional 72 participants to evaluate the effectiveness of the implemented system. The results of this study are extensively discussed to inform future research on the development of customizable and user-centered UIs.

2 Related Work

To provide the background for our work, we introduce prior work on traditional fixed UI, natural language UI, and dynamically generated UI. We discuss their advantages and limitations and compare them with our proposed framework. A visual comparison is presented in Figure 2.

2.1 Traditional Fixed UI

Traditional UI design in professional digital software like Illustrator [2], Photoshop [3], Premiere Pro [5], and Blender [9] follows a designer-centric model, where a fixed set of tools and controls are designed for various tasks. These applications rely on well-established interface elements such as dropdown menus, sliders, text fields, and panels that allow users to manipulate images, graphics, videos, and 3D models. For example, Photoshop provides consistent tools for adjustments like brightness, contrast, and hue, while Illustrator offers vector editing through pre-defined tools. Premiere Pro delivers video editing features through timelines and control panels, and Blender offers a complex interface for 3D modeling and animation. These tools are powerful but often rigid, requiring users to adapt to the software's structure. As a result, their complexity can be overwhelming to users, and their interface may not always align with individual workflows or preferences, which limits the flexibility and personalization of user experience.

2.2 Natural Language UI

Using natural language as input to edit digital content has gained increasing interest due to its simplicity. This approach allows users to perform tasks like image, video, data visualization, 3D models, and human motion editing by simply describing the changes they want rather than navigating complex software interfaces. For example, natural language-based image editing tools, such as DALL-E [38] or Runway [45], allow users to generate and edit images by typing commands like “*make the sky bluer*” or “*add a tree in the background*”. Similarly, tools like Adobe Firefly [1] and Photoshop Generative Fill [4] enable image generation and edits using prompts like “*change to a vintage look*” or “*add a yellow bee in the circled region next to the flower*”. In video editing, tools like Descript [21] allow users to edit video content by editing a transcript, and tools like Runway and Premiere Pro with Sensei AI [6] also support video editing using natural language prompts like “*add a motion blur effect*”. Also, there are many research projects [11, 24, 25, 43, 60] that have significantly advanced the techniques for image and video editing using language as the primary modality. In the field of visualization, systems like Chat2VIS [35], NL4DV [46] and Data Formulator [53, 54] allow users to create charts and graphs by simply asking for a “*bar chart showing sales over time*”, and LIDA [22] enable the generation of grammar-agnostic visualizations and infographics using LLMs. For the generation and editing of 3D model [13, 41] and human motion [15, 33, 44, 48], prior research has investigated using language to describe users’ goals, such as “*create a 3D lemur taking notes in a journal*” or “*change the upper body motion to throwing a ball and keep the original running motion*”. These innovations are transforming traditional workflows by making advanced editing more accessible to non-expert users, reducing the technical execution gulf and enabling them to focus on creative intent.

However, natural language interfaces struggle with precise control over specific details and cannot effectively support interactive editing, limiting their ability to handle complex, fine-grained adjustments typically required in professional workflows.

2.3 Dynamically Generated UI

Recent research has studied generating dynamic UIs to provide users with precise, direct interactions for content editing. One prominent approach is the relaxation method, which involves creating UI

widgets that provide generalized control over variables within a function or query. For instance, Mavo [51] provides users with the capability to construct interactive HTML pages by incorporating unique attributes, which, in turn, recommend editing widgets tailored to those attributes. Heer et al. [28] employ query relaxation techniques, allowing users to broaden their selection effectively. Bespoke [50] utilizes user demonstrations to generate custom UIs for command-line applications. It applies rule-based heuristics to infer the semantic types of parameters in bash commands and thus dynamically create widgets for parameter manipulation. Another line of research, known as precision interfaces, uses SQL queries as a framework for UI generation [16, 18, 59]. These systems convert a sequence of input queries into interactive widgets, allowing users to adjust query parameters directly. An example is NL2Interface [17], which translates natural language commands into SQL queries and subsequently generates a UI for users to edit parameters within the queries.

In addition to generating dynamic UIs from command-line applications and queries, recent advancements have focused on creating UIs directly from natural language descriptions. This approach allows users to articulate their goals in natural language, and the system generates an interactive UI tailored to the task at hand using LLM’s reasoning capabilities. For instance, DynaVis [49] generates dynamic interfaces for visualization editing based on natural language commands. By enabling users to describe changes to their visualizations through simple text, such as “*rotate the x-labels by 30 degrees counter-clockwise*”, DynaVis dynamically creates suitable widgets, such as a slider for adjusting the rotation of x-labels. Similarly, Biscuit [19] generates UIs to help users follow machine learning tutorials in Jupyter Notebooks using natural language queries. Users can describe tasks like “*sample training data*” or “*adjust model hyperparameters*”, and Biscuit generates ephemeral UIs for the specified parameter adjustments.

Although natural language-based UI generation generates UIs without depending on fixed rules or query relaxation techniques and utilizes LLMs’ advantage to interpret intents from high-level descriptions, most prior work using standalone LLMs does not fully consider individual user preferences and domain-specific task requirements. This can result in interfaces that, while functional, may not be the most effective or usable for specific users, especially when personalization and task-specific requirements are needed.

3 User Feedback Collection and Principles for UI Design

While LLM-powered UI generation offers many advantages, it primarily relies on generic training data, lacking task-specific contexts and user-centered insights, such as individual preferences. User preference is part of user feedback and plays a crucial role in UI design and UI widget selection. The user feedback data is traditionally collected through labor-intensive methods like interviews, usability tests, A/B tests, and feedback forms [47, 55]. Given that LLMs are trained on large-scale human data, they provide a promising solution to incorporating user preferences into dynamically generated UIs. However, without dedicated user preference datasets, current LLMs cannot fully address this need. To fill this gap, we propose crowdsourcing a preference library to enhance LLM-driven UI widget generation with user preference data, motivated by the effectiveness of adopting crowdsourcing for user feedback collection to assist various design areas [23, 31, 34, 40, 56–58].

To determine the specific aspects of user preference, we refer to well-established UI design principles developed by prior research. McKay’s UI is Communication [37] emphasizes that an intuitive interface relies on effective communication that balances multiple principles, including predictability, efficiency, discoverability, and explorability. McKay argues that intuitive UIs align with users’ prior experiences, providing clear affordances and responsive feedback to guide interaction. Lidwell et al.’s Universal Principles of Design [32] expand on these ideas by highlighting fundamental design principles such as the flexibility-usability tradeoff, mapping, and mental models, which underscore the importance of aligning system controls with user expectations and simplifying

complex interactions. Together, these principles highlight that successful UIs are predictable and efficient and allow for exploratory interaction without overwhelming the user. Drawing from these foundational principles, we have identified *predictability*, *efficiency*, and *explorability* as three critical aspects to guide our approach to building a user preference library to enhance LLM-powered UI widget generation.

The remainder of the paper is organized as follows: first, we provide an overview of our proposed framework. Next, we describe the implemented system in the image editing domain and discuss the crowdsourced data collection process. This is followed by a detailed user evaluation, along with a discussion of the limitations, potential future work, and the conclusion.

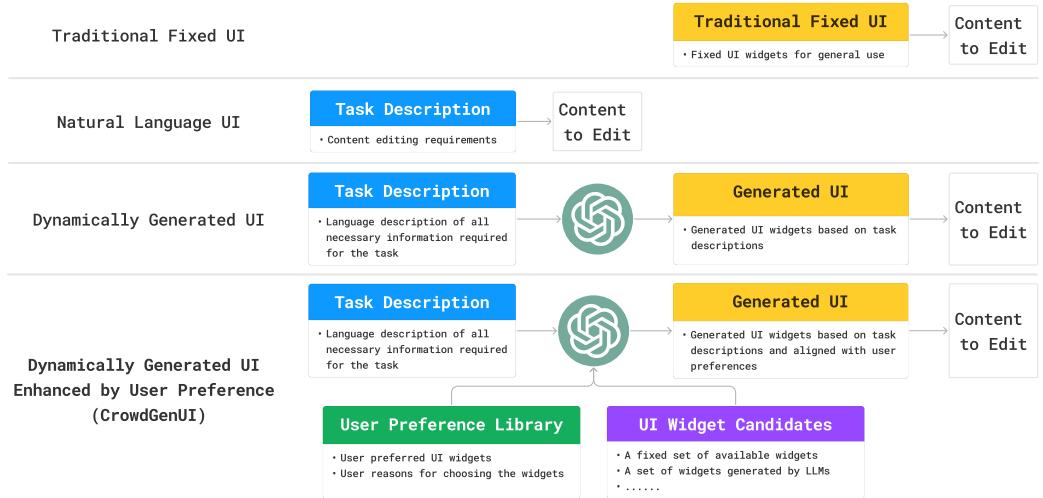


Fig. 2. Overview of our proposed *CrowdGenUI* framework and comparison with prior work. Our framework consists of four main components: it takes (1) task description in natural language as input, and this information is passed to the LLM together with (2) the collected user preference library and (3) a collection of UI widget candidates. All the input data is used to guide the LLM reasoning process to obtain (4) generated UI widgets that are aligned with user preferences and task requirements.

4 Framework Overview

Figure 2 provides an overview of our proposed framework, *CrowdGenUI*, and a visual comparison with previous research. Our framework improves existing UI widget generation systems that take task descriptions as input for LLMs to generate UIs dynamically. To align generated UI widgets with specific user needs, we collect a user preference library that captures users' preferred UI widgets and their rationale. This library is provided to the LLM alongside task descriptions and widget candidates to guide the UI generation process. For implementation, the widget candidates are selected from the widget list of Jupyter Widgets Library [30]. These candidates can be easily extended to include additional widget options, such as those from other widget libraries or LLM-generated widgets.

5 Crowdsourced Library-enhanced UI Widget Generation for Image Editing

Figure 3 illustrates the system we implemented for our proposed framework. In this section, we dive into the technical details of this system, including UI widget reasoning, integration of the crowdsourced library into the reasoning process, and UI widget code generation.

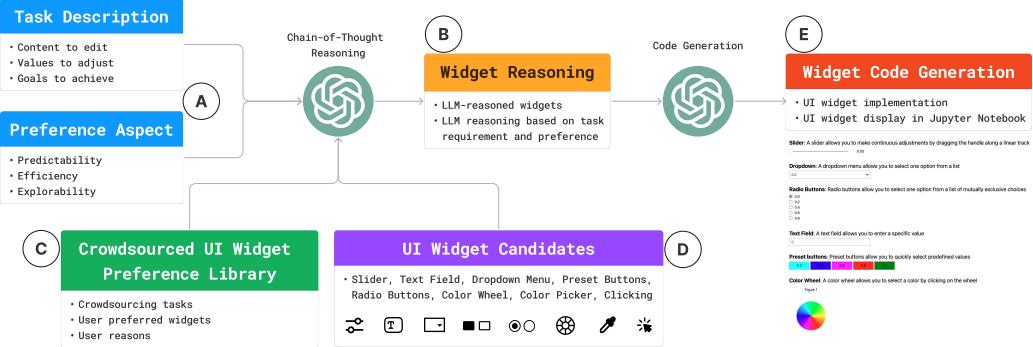


Fig. 3. Crowdsourced user preference library-enhanced UI generation by LLMs for image editing. The system starts from language input specifying the task description and preference aspect, as illustrated in ④. With this input, the LLM applies chain-of-thought reasoning to obtain reasoned widgets, as shown in ⑤. Based on this widget reasoning process, the crowdsourced user preference library, containing the crowdsourcing tasks, user-preferred widgets, and their rationale, is integrated to augment widget reasoning, as depicted in ⑥. Additionally, a list of UI widgets is fed to the LLM for reasoned widget implementation, as displayed in ⑦. Lastly, the reasoned widgets are passed to the LLM for code generation in Python. This allows widget implementation and display in Jupyter Notebook, as demonstrated in ⑧.

5.1 UI Widget Reasoning

The first step of the UI widget generation system is reasoning UI widgets based on task descriptions. These descriptions need to clearly specify the content to edit (e.g., an image), data manipulation requirements (e.g., adjusting the image color), and goals to achieve (e.g., changing the image to fall colors after color adjustment). In addition, user preference aspects, such as predictability, efficiency, and explorability, also need to be specified. Once the input information is prepared, as shown in Figure 3 ④, it is passed to LLMs for widget reasoning. We prompt the LLM to use a chain-of-thought reasoning approach for this process. Specifically, the LLM first analyzes the task description to identify the task requirements and the values that need adjustment. Next, we ask the LLM to consider different preference aspects, reason which UI widgets are most appropriate for each aspect, and provide justifications for its choices. The LLM responses are then encoded in JSON format, including the task name, the reasoned widgets for each preference aspect, and the corresponding reasoning, as demonstrated in Figure 3 ⑤. The specific prompt used for guiding the LLM's reasoning process can be found in Appendix B.1.

5.2 Integration of Crowdsourced User Preference Library

When integrating the crowdsourced user preference library into the LLM widget reasoning process, we enhance the basic chain-of-thought reasoning process by providing specific instructions on how to utilize the library. The library-enhanced reasoning offers the LLM descriptions of tasks used during crowdsourcing, user-selected widgets for each task and preference aspect, and the reasons users provided for their choices, as illustrated in Figure 3 ⑥. To support the implementation of LLM-reasoned widgets (see Section 6.2.4 for details), we prepare a list of UI widgets and feed it to the LLM to guide the reasoning process, as depicted in Figure 3 ⑦. With all the input data, the LLM is instructed to first identify the similarity between the given task and the crowdsourcing tasks and refer to the user-preferred widgets from those relevant tasks. Next, the LLM is prompted to consider the widgets that appeared most frequently for each preference aspect and base its

reasoning on this information. The LLM is also required to reference user-provided reasons to assess the appropriateness of its reasoned widgets and deliver its own reasoning. This multi-step process ensures that the crowdsourced data informs LLM reasoning for UI widget generation in a structured way. Like the basic reasoning method, the output needs to be in JSON format, detailing the relevant tasks, widgets, and reasoning based on the crowdsourced library. The detailed prompt used for this enhanced widget reasoning process is provided in Appendix B.2, and a subset of the crowdsourced library, containing data from five users, is included in Appendix B.3.

5.3 UI Widget Code Generation

Once our system determines which widgets to use, we also use the LLM to generate the Python code needed to implement the reasoned widgets, as displayed in Figure 3 (E). We provide an example code for the implementation of all the widget candidates for the LLM to reference during code generation. Our experiment shows that including example code significantly reduces bugs in the LLM-generated code. More importantly, the example code enables the LLM to successfully generate widget implementations that go beyond the default functionality of the Jupyter Widgets Library [30]. For instance, a color wheel widget requires additional use of the Interactive Canvas in Jupyter [29] and the implementation of mouse interactions on the wheel to map clicks to the corresponding colors. In addition to specific widget implementations, the example code also offers a structure to organize and display widgets in a clear, structured format within Jupyter Notebook for our user evaluation. The prompt used for code generation and the example code for LLM reference can be found in Appendix B.4.

5.4 Implementation Details

We use GPT-4o [39] as the LLM for both widget reasoning and code generation, with Python as the programming language to implement the widgets. Occasionally, the code generated by GPT-4o may have minor bugs and might not compile correctly. We manually fix these issues to ensure all code generated by the LLM runs properly during user evaluation. This setup aligns with our focus on LLM-driven widget reasoning and code generation rather than real-time UI generation. We use this system to serve as a prototype to demonstrate how LLM reasoning, when enhanced by a crowdsourced library, can effectively guide UI widget generation according to task requirements and user preferences, particularly in the image editing domain.

6 Crowdsourcing a Preference Library for UI Widgets

In this section, we introduce how we collect user preference data to build the library that enhances LLM-driven UI widget generation. Following our previous implementation, we gather preference data in the image editing domain to illustrate the process. We present the study setup and provide an analysis of the collected data.

6.1 Platform and Participants

The crowdsourcing study took place on Prolific [42]. We recruited participants with Python programming backgrounds solely to ensure they were familiar with running the provided Python code in Jupyter Notebook for UI widget interaction. They were required to interact with the Jupyter Notebook on desktops using a mouse and keyboard. In total, 50 users participated in the crowdsourcing study (Age: 29.86 ± 7.83 , Male: 37, Female: 12, Non-binary: 1). Each participant was compensated 10 USD for the 30-minute study. We did not require participants to have specific image editing or UI programming experience, as we aim to make the LLM-generated UI widgets fully usable for regular users and well-aligned with their preferences.

6.2 Crowdsourcing Setup

6.2.1 User Interface. In the crowdsourcing study, we offered each participant a Jupyter Notebook to work on eight image editing tasks, interact with several UI widgets to perform each task and provide their preferences for the widgets. Figure 4 shows the user interface for a crowdsourcing task—adjusting an image to fall colors. The interface contains the task requirement, multiple UI widgets, an image to edit, and a user preference selection panel.

6.2.2 Tasks. The crowdsourcing tasks cover three categories in the image editing domain. We introduce these categories and provide an overview of the tasks below. Some tasks may overlap across multiple categories. Our goal is to use these tasks to incorporate a range of value adjustments and data manipulation needs in image editing.

Adjust the color settings to gradually shift the image tones, creating a warm, autumnal atmosphere.

Ensure the final image after adjustments is the one that looks the most autumnal to you.

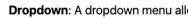
```
from image_editing.fall_color_adjustment import display_fall_color_widgets
%matplotlib widget
display_fall_color_widgets()
✓ 0.9s
```

In this task, we provide you with a set of widgets to adjust the color of an image. You can use the slider, dropdown, radio buttons, text field, preset buttons, and color wheel. Please explore all the widgets and observe the changes in the image.

Slider: A slider allows you to make continuous adjustments by dragging the handle along a linear track



Dropdown: A dropdown menu allows you to select one option from a list

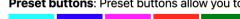


Radio Buttons: Radio buttons allow you to select one option from a list of mutually exclusive choices



Text Field: A text field allows you to enter a specific value

Preset buttons: Preset buttons allow you to quickly select predefined values



Color Wheel: A color wheel allows you to select a color by clicking on the wheel



Color Adjustment: 0.00



(a) Task description and provided UI widgets to perform the task.

Please select the UI widget that provides the greatest

- Predictability:** (allows you to obtain results without surprises or needing to deduce how to perform the interaction)
- Slider
 - Dropdown
 - Radio Buttons
 - Text Field
 - Preset Buttons
 - Color Wheel

- Efficiency:** (allows you to perform tasks with a minimum amount of effort)
- Slider
 - Dropdown
 - Radio Buttons
 - Text Field
 - Preset Buttons
 - Color Wheel

- Explorability:** (allows you to explore multiple possibilities and perform tasks with high flexibility)
- Slider
 - Dropdown
 - Radio Buttons
 - Text Field
 - Preset Buttons
 - Color Wheel

(b) UI widget preference selection panel.

Fig. 4. Crowdsourcing user interface for task: adjusting an image to fall colors (image_adjust_fall_color).

- **Continuous value adjustment:** These tasks need to adjust a value across a continuous range smoothly. Examples include adjusting image lightness, saturation, hue, and color balance.
- **Discrete value selection:** These tasks involve selecting from predefined options. Examples include choosing predefined values for image lightness, saturation, hue, and watermark positions.
- **Tasks emphasizing visual feedback:** These tasks require immediate visual feedback, such as color or position changes. Examples include changing an image to fall tones, matching an object’s color to a given color, positioning a watermark, and highlighting a specific image area.

The detailed task descriptions are presented in Table 1. We use these tasks to encourage exploration, such as in Tasks 1, 2, 3, 6, and 7, while also emphasizing precision, as seen in Tasks 4, 5, and 8. Two tasks are used as control tasks: `image_adjust_fall_color` and `image_color_match`. We instruct participants to ensure that the final images best meet the task requirements and use these tasks as attention checks to help identify untrustworthy responses. We randomize the order of tasks, ensuring that no particular task appears consistently in the same position to reduce order effects, such as fatigue or priming, that could influence user preferences and behaviors.

Table 1. Crowdsourcing task names and descriptions.

Task Name	Task Description
1 <code>image_adjust_lightness</code>	Experiment with lightness settings to see how different levels affect the image.
2 <code>image_adjust_saturation</code>	Boost the saturation to make the colors pop.
3 <code>image_adjust_hue</code>	Experiment with different hues to find a color tone that complements the overall mood of the image.
4 <code>image_adjust_fall_color</code>	Adjust the color settings to gradually shift the image tones, creating a warm, autumnal atmosphere. Ensure the final image is the one that looks the most autumnal to you.
5 <code>image_color_match</code>	Change the rocket’s color to match the provided reference color. Ensure the rocket’s color in the final image best matches the reference color.
6 <code>image_adjust_color_balance</code>	Experiment with different color balance settings to see how altering the red, green, and blue levels affects the overall color harmony of the image.
7 <code>image_place_watermark</code>	Experiment with different watermark positions to find the perfect balance between visibility and subtlety.
8 <code>image_place_vignette</code>	Darken the background using a vignette effect except for the human face by properly positioning the circle around the face.

6.2.3 *User Preference Aspects.* We use three key aspects of UI widget preferences, drawing from prior research on intuitive user interface design principles (recall Section 3). These aspects—predictability, efficiency, and explorability—reflect diverse user preferences when interacting with digital software, with the detailed descriptions below:

- **Predictability:** Users can achieve results without encountering unexpected outcomes or needing to figure out how to perform the interaction.
- **Efficiency:** Users can perform tasks with minimal effort.
- **Explorability:** Users can explore multiple possibilities and perform tasks with high flexibility.

6.2.4 *UI Widget Selection and Implementation.* UI widget selection needs to consider task and data manipulation requirements, such as input and output data types, and match the widget types with the specified task requirements. We choose UI widgets for the crowdsourcing tasks from the Jupyter Widgets Library [30], which offers both basic and advanced UI widgets, covering a

wide range of control needs. Specifically, for continuous value adjustment tasks, sliders  are typically ideal to allow for fine-grained control, and text fields  can be used for precise value input. For discrete value selection tasks, dropdown menus  and radio buttons  are often effective. For tasks emphasizing visual feedback, UI widgets like color pickers , color wheels , direct manipulation (e.g., clicking ), and preset buttons with preview overlays   allow users to see outcome previews through color or position visual cues.

In this work, we present one approach to UI widget implementation and selection, acknowledging that it does not cover all possible options. However, this method can be easily expanded by leveraging additional UI widget libraries, such as [10, 36, 52]. These libraries can facilitate data manipulation and content editing in software or websites and enable the inclusion of a broader range of widget types in various applications.

6.3 Crowdsourced UI Widget Preference Library

6.3.1 Overview. We distributed the crowdsourcing study to the 50 participants in three versions. Each version included all eight tasks and a subset of the three preference aspects, covering one, two, or all three aspects. This approach ensured that each participant provided their preferences for 15–18 preference selections out of the total $3 \text{ aspects} \times 8 \text{ tasks} = 24$ selections across all tasks, helping to prevent fatigue. The study yielded a UI widget preference library with 30 unique user responses for each aspect. In total, we obtained $30 \text{ responses} \times 3 \text{ aspects} \times 8 \text{ tasks} = 720$ user preferences. Figures 5 and 6 show the distribution of user preferences for all crowdsourcing tasks.

6.3.2 Analysis of User Preference Distribution. From the crowdsourcing results of all 30 responses to each preference aspect, we observe variations in user preferences for UI widgets regarding predictability, efficiency, and explorability across the eight tasks.

For continuous value adjustment tasks, including adjusting image lightness, saturation, and hue, many users preferred preset buttons with preview overlays for their predictability and efficiency, with reasons like “*The preset button’s background allows for users to figure out the lightness relatively quickly, and when clicking on the buttons, the button updates the lightness to how it looks on the buttons, allowing for users to need minimal deduction to understand what this button does*”. For explorability, sliders were favored for adjusting lightness and saturation. Participants commented that “*The slider allows users to slowly go through each increment and explore what effect they have on the image. They can go anywhere within the range to understand the effect of saturation on the image and precisely pinpoint which exact value of saturation they want with a slider*”. A color wheel was preferred to explore hue adjustment, with comments like “*The color wheel, due to it allowing you to select exactly what you want and visualizing it, allowing you to explore and experiment very easily*”.

For discrete value selection tasks, including choosing predefined values for image lightness, saturation, hue, and watermark positions, preset buttons were favored over radio buttons and dropdown menus in all three preference aspects. This was primarily due to the visual cues that preset buttons provided, such as user-provided reason saying that “*The preset buttons include a preview of how the saturation would look like. For example, -1.0 is fully gray, while 1.0 is a bright rainbow. This allows users who may not know what saturation means to understand what each button does; it changes how colorful the image is*”.

For tasks emphasizing visual feedback, preferences varied based on whether the task involved color or position adjustments. In color adjustment tasks, including changing image hue, applying fall colors, and adjusting color balance, many users preferred color wheels and color pickers for predictability and explorability, while preset buttons were preferred for efficiency, similar to continuous and discrete value adjustment tasks. However, for tasks requiring precise color adjustment, such as matching an object’s color to a given color, there was no clear consensus

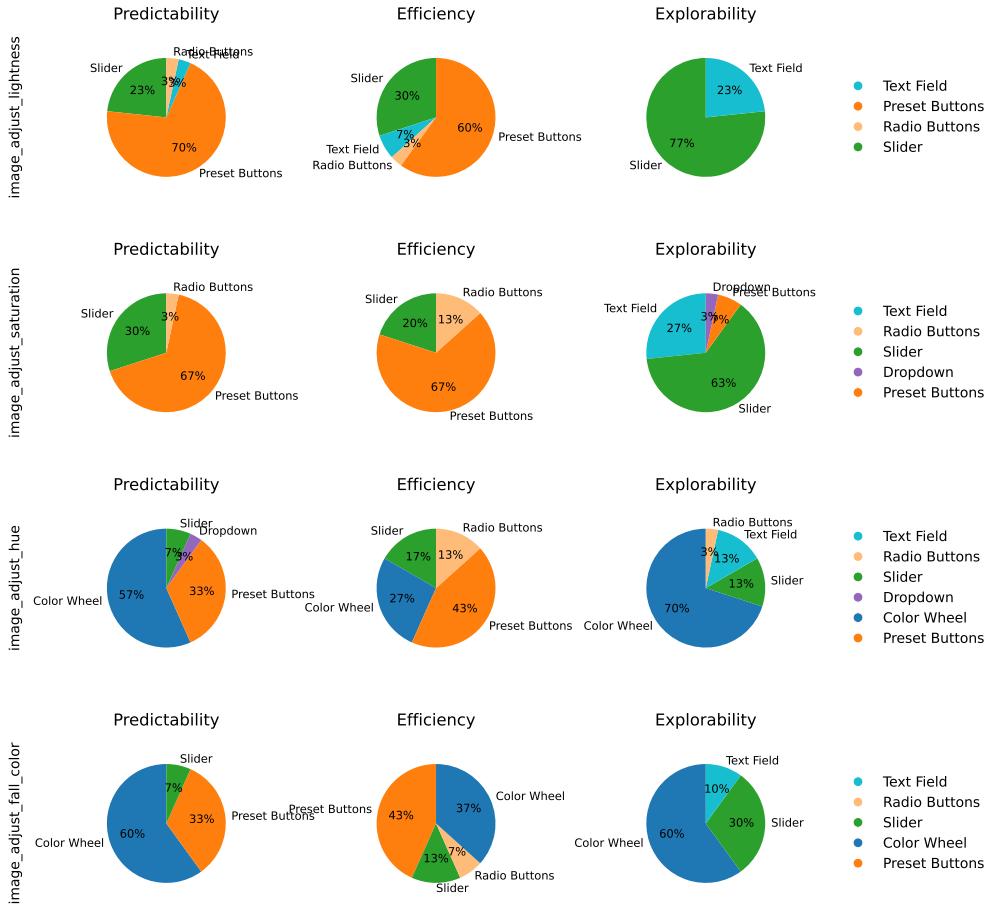


Fig. 5. User preference distribution of tasks 1-4 collected in the crowdsourcing study.

regarding predictability. Participants were evenly split among text fields, color wheels, color pickers, and preset buttons. The reasons for their preferences included “*Since I know what my target is, just typing it gives me what I want*” for the text field, “*It is easy to visualize the color*” for the color wheel, “*The color picker would accurately allow me to pinpoint the target color green and not have a doubt that it is correct*” for the color picker, and “*The preset button convey closely what color the rocket will have*” for the preset buttons. Regarding efficiency, most users preferred text fields, as they could quickly “*enter the hex number and see the desired result*”. For explorability, color wheels and color pickers were the most favored.

In position adjustment tasks, including placing a watermark or highlighting a specific area of an image, preset buttons and directly clicking on the image were preferred for predictability and efficiency, with reasons like “*The presets give an indication of what to expect and wouldn’t confuse me in this task*” and “*Clicking is both intuitive and efficient and I know what I’ll get*”. For explorability, users favored both clicking on the image since it was “*Easy to just click in slightly different places as needed to get the area where I want it*”, and using sliders because “*You can totally visually discern where you want the watermarks and/or image to go to as it is continuous*”.

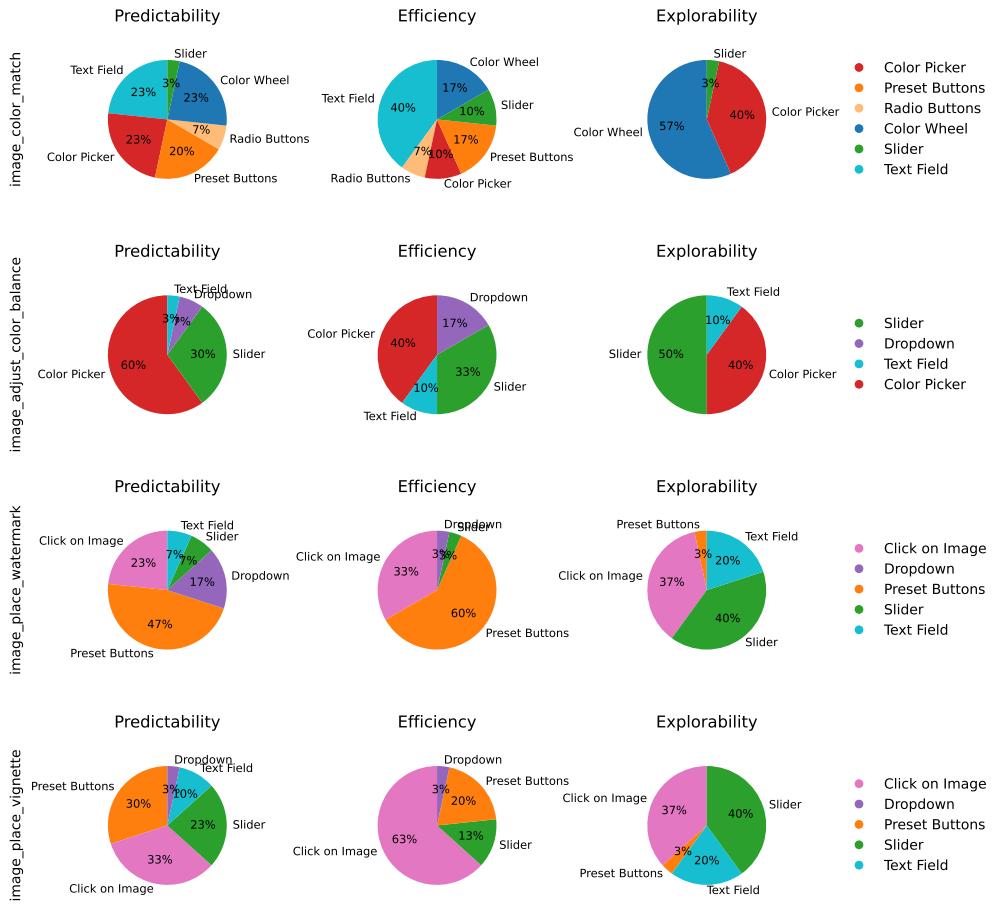


Fig. 6. User preference distribution of tasks 5-8 collected in the crowdsourcing study.

These collected user preferences are incorporated to guide UI widget reasoning in our user evaluation of the LLM-driven UI widget generation system. We present the user evaluation details and how the crowdsourced user preferences affect reasoned widgets for user evaluation tasks in the following section.

7 User Evaluation

In this section, we introduce our user evaluation to assess our implemented system for the crowdsourced library-enhanced UI widget generation framework. We outline the research questions to evaluate in the study, describe the evaluation setup, and present the evaluation results.

7.1 Research Questions

We aim to study the following research questions in our user evaluation, focusing on the effectiveness and generalizability of the UI widget generation framework and the impact of integrating the crowdsourced UI widget preference library:

- **RQ1—Adaptivity of the framework to new, related tasks:** Does the UI widget generation framework enable the generalization of crowdsourced library-enhanced widget reasoning and generation to perform tasks that differ from, but relate to, the original crowdsourcing tasks?
- **RQ2—Impact of the library on aligning generated widgets with user preferences:** Can the crowdsourced UI widget preference library, which is integrated into the UI widget generation framework, help generated widgets align more closely with user preferences for predictability, efficiency, and explorability compared to those generated without a library?
- **RQ3—Effect of library size on enhancing LLM reasoning:** Does the size of the crowdsourced UI widget preference library affect LLM widget reasoning effectiveness? Which size of the library provides the strongest support for LLM reasoning capabilities?

In the following subsections, we introduce the user evaluation setup designed to answer these research questions and analyze our evaluation results in alignment with them.

7.2 Platform and Participants

We conducted the user evaluation on Prolific [42]. Similar to the crowdsourcing study, the participants we recruited had Python programming backgrounds and worked on the user evaluation tasks in Jupyter Notebook on desktops. We recruited 72 participants (Age: 30.03 ± 9.41 , Male: 52, Female: 17, Non-binary: 3) and compensated each participant 10 USD for the 30-minute user study.

7.3 Evaluation Setup

7.3.1 Procedure. The user evaluation invited voluntary participants, who were first presented with a consent declaration at the beginning of the study. Once participants agreed to participate, they accessed the study interface via a URL linked to our server. Before starting the tasks, they were given a briefing that outlined the study requirements. Full details of the user evaluation briefing and instructions can be found in Appendix C. Following that, they worked on three tasks and answered six questions for each task to provide their preferences for LLM-generated widgets.

7.3.2 User Interface. Figure 7 shows the user interface for a task in the user evaluation—adjusting image tint. Each participant interacted with a Jupyter Notebook containing LLM-generated UI widgets to work on tasks distinct from those in the crowdsourcing study. The UI widgets were generated by the LLM alone and enhanced by our crowdsourced library. After interacting with all the offered widgets, participants were shown a preference selection panel where they chose their favored options for each task.

7.3.3 Tasks and Preference Aspects. The user study tasks differ from the crowdsourcing tasks but still fall within the same three categories, promoting either exploration or precision (recall Section 6.2.2). We provide an overview of these tasks and their categories below, with detailed descriptions available in Table 2. These tasks are designed to address RQ1. Note that two tasks belong to the design domain, i.e., `design_align_text` and `design_position_logo`, expanding beyond the image editing domain emphasized in the crowdsourcing study. Through these user evaluation tasks, we aim to test our framework’s generalizability to new tasks in both image editing and additional domains.

- **Continuous value adjustment:** Tasks include adjusting image exposure, tint, and temperature.
- **Discrete value selection:** Tasks include choosing predefined values for image exposure, tint, and temperature adjustment, text alignment, and logo positioning.
- **Tasks emphasizing visual feedback:** Tasks include changing an image’s color to spring colors, aligning text with the margin, and placing a logo.

Adjust the image tint to shift its color balance, creating both subtle and dramatic effects by adding a yellow or magenta hue.

```
from user_study_helpers.widget_code_runner import widget_code_runner
task_name = 'image_adjust_tint'
%matplotlib widget
widget_code_runner(user_id, task_name)
✓ 0.9s
```

Slider:

R: 255
 G: 255
 B: 255

Figure 2

Text Field:

R:	255
G:	255
B:	255

Tint Adjustment - R: 255, G: 255, B: 255



Preset buttons:

Red Green Blue Yellow Cyan Magenta Orange Purple

Color Wheel:

Figure 1



Color Picker:



(a) Task description and LLM-generated UI widgets to perform the task.

Question 6: Please select one from the two options below that make the most sense to support **efficiency** (allows you to perform tasks with a minimum amount of effort)

Option 1

- Color Picker:
 Score: 7

Reason: Referring to the efficiency reasons for 'image_color_match', the Color Picker allows users to directly choose the target color with minimal effort, making the task quick and straightforward.

- Color Wheel:

Score: 2

Reason: For 'image_adjust_hue', the Color Wheel lets the user rapidly select their desired hue with a quick visual reference, which is very efficient for making quick adjustments without needing to type values or use sliders.

- Text Field:

Score: 1

Reason: For 'image_color_match', the text field is noted for allowing precise input, which can be highly efficient for experienced users who know the exact values required. Inputting specific values directly lets users achieve the desired effect quickly, reducing the time spent on trial and error.

Option 2

- Preset Buttons:

Score: 8

Reason: Preset buttons enable quick application of predefined color casts (e.g., warmer tones or cooler tones) with minimal effort, facilitating a fast task completion.

- Slider:

Score: 2

Reason: Sliders allow users to make quick adjustments with a simple drag of the control. They can efficiently shift the color balance towards pink or green without navigating through complex menus.

Select your preferred option

Option 1

Option 2

Your reason of selecting the option...

(b) User preference selection panel for pairwise comparison. The panel contains six questions for each task, and we show one of them due to space limits.

Fig. 7. User evaluation user interface for task: adjusting image tint (image_adjust_tint).

The preference aspects: predictability, efficiency, and explorability, are drawn from the crowdsourcing study. We use the same aspects to evaluate whether user preferences collected from

Table 2. User evaluation task names and descriptions.

Task Name	Task Description
1 image_adjust_exposure	Adjust the image exposure by both decreasing and increasing it. Find the exposure level that appears the most natural and visually pleasing to you.
2 image_adjust_tint	Adjust the image tint to shift its color balance, creating both subtle and dramatic effects by adding a yellow or magenta hue.
3 image_adjust_temperature	Adjust the image temperature to warm and cool tones. Experiment with different settings to achieve the most desired effect for you.
4 image_change_to_spring	Transform the image to reflect vibrant spring colors, adding fresh, lively hues.
5 design_align_text	Align the text “Poster Title” perfectly along one of the margins. Experiment with various positions to achieve a balanced and visually pleasing design.
6 design_position_logo	Experiment with various placements for the logo to determine the most visually appealing position that enhances visibility and complements the overall design of the image.

crowdsourcing can effectively lead to preference-aligned widget generation for tasks in the user evaluation. This setup addresses **RQ2**.

7.3.4 UI Widgets for the Evaluation Tasks. Figure 8 shows an example of LLM-reasoned widgets for task `image_adjust_tint` enhanced by three different sizes of the crowdsourced preference library: `withlib10`, `withlib25`, and `withlib30`, as well as without any library: `withoutlib`. `Withlib10`, `withlib25`, and `withlib30` stand for the crowdsourced library with 10, 25, and 30 unique responses to each preference aspect. As introduced in Section 6.3.1, we obtain 30 responses to each aspect in total, i.e., `withlib30`, and use its subsets to construct `withlib10` and `withlib25`.

In Figure 8, the percentages in the pie charts show the frequency of each type of UI widget reasoned by the LLM over 10 iterations. For instance, in the pie chart for efficiency reasoned by `withoutlib` (the rightmost chart in the second row), the preset buttons are reasoned 8 times (80%), while the slider is reasoned 2 times (20%). Reasoned widgets differing in multiple iterations are caused by the ad-hoc nature of LLMs, i.e., they do not always generate all possible widgets in a single pass, similar to the observations in previous work [8, 49]. This observation reveals varying consistencies of LLM-reasoned widgets with different library sizes. Thus, we leverage the reasoning frequency to indicate the recommendation strength for each type of widget by assigning each a score in our user evaluation. For example, the frequencies of 8 for the preset buttons and 2 for the slider are used as scores 8 and 2 for them, as seen in Figure 7b Option 2. Due to space limit, we provide the LLM-reasoned widgets with different library sizes for all six tasks in Appendix A.

As seen in Figure 8, LLM-reasoned UI widgets differ significantly with and without the crowdsourced libraries across the three preference aspects. For predictability, the color wheel is the most frequently reasoned widget when using the library, while the slider takes precedence in the absence of a library. In terms of efficiency, both `withlib10` and `withlib30` favor the color picker, while `withlib25` most frequently reasons the color wheel, followed closely by the preset buttons and color picker. Without a library, preset buttons are the most commonly reasoned widgets. Regarding explorability, the color wheel is reasoned the most when libraries are utilized, whereas the color picker is the top choice without a library. Similar reasoning differences can be found in the reasoned widgets for other tasks in Appendix A. These findings prompt us to address **RQ3**: What is the optimal size of the crowdsourced library that maximizes LLM reasoning capabilities preferred by

users with regard to the three preference aspects? To investigate this question, we conduct pairwise comparisons between libraries and collect user reasons for their choice.

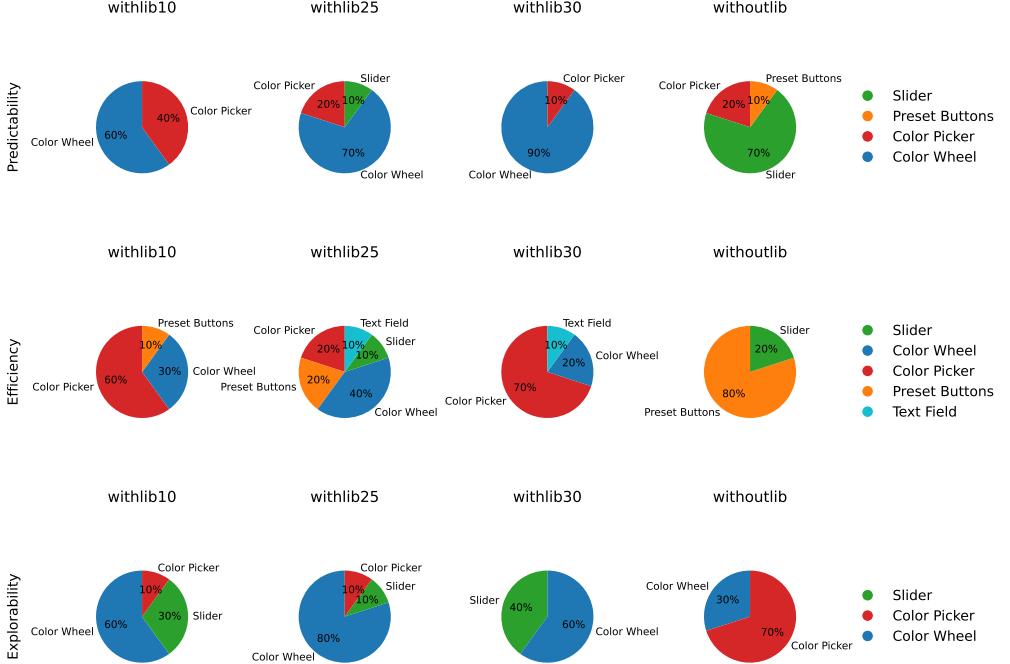


Fig. 8. LLM-reasoned UI widgets for `image_adjust_tint` with three sizes of the crowdsourced libraries and without any library. Withlib10, withlib25, and withlib30 refer to using the crowdsourced libraries with 10, 25, and 30 user responses for all aspects and tasks, while withoutlib refers to using no library.

7.3.5 Pairwise Comparisons. We conduct pairwise comparisons for the reasoned widgets, their reasoned frequency, and the LLM-provided reasoning enhanced by withlib10, withlib25, withlib30, and withoutlib. By including widgets reasoned over multiple iterations, 10 in our study, we aim to include as many relevant widgets as possible in the evaluation, enabling a comprehensive comparison of LLM reasoning capabilities augmented by different sizes of the libraries. Participants need to select their preferences by comprehensively considering the widgets, their frequencies, where higher frequency indicates a stronger recommendation, and the reasoning, which explains why each widget is deemed appropriate for the task and preference aspect (recall Figure 7b for an example of pairwise preference selection). Each preference aspect in each task results in six pairwise comparisons, as outlined in Table 3.

In total, this gives us $6 \text{ pairs} \times 3 \text{ aspects} \times 6 \text{ tasks} = 108$ comparisons. To prevent user fatigue, each participant was assigned 18 pairs, which consisted of all six pairs for each aspect across three tasks. Specifically, the user study was divided into two task sets, each containing three tasks. Each participant provided preferences for all three aspects of their assigned task set. The tasks were arranged in all six possible order combinations, and the aspect order was balanced using the Latin Square method [20]. See Table 4 for the detailed study setup. The six pairwise comparisons for

Table 3. Pairwise comparisons of different crowdsourced library sizes for each preference aspect in each task. Pair orders are randomized in the user evaluation.

Library Sizes Compared	
1	withlib10 vs. withlib25
2	withlib10 vs. withlib30
3	withlib10 vs. withoutlib
4	withlib25 vs. withlib30
5	withlib25 vs. withoutlib
6	withlib30 vs. withoutlib

each aspect of each task were presented in random order. To ensure robust statistical results, each pairwise comparison was tested by 12 users, resulting in 6 pairs \times 12 users = 72 total participants.

Table 4. User evaluation setup. The six user evaluation tasks are divided into two task sets. In each set, we arrange all six order combinations of tasks and aspects and randomize orders of pairwise comparisons.

Task Set 1			Task Set 2		
Order	Task Name	Aspect	Order	Task Name	Aspect
6 ×	image_adjust_exposure	Predictability	6 ×	image_adjust_tint	Predictability
	image_adjust_temperature	Efficiency		image_change_to_spring	Efficiency
	design_align_text	Explorability		design_position_logo	Explorability
6 ×	image_adjust_exposure	Efficiency	6 ×	image_adjust_tint	Efficiency
	image_adjust_temperature	Explorability		image_change_to_spring	Explorability
	design_align_text	Predictability		design_position_logo	Predictability
6 ×	image_adjust_exposure	Explorability	6 ×	image_adjust_tint	Explorability
	image_adjust_temperature	Predictability		image_change_to_spring	Predictability
	design_align_text	Efficiency		design_position_logo	Efficiency

7.4 Results

7.4.1 *Results of Each Aspect across All Tasks.* We present the user evaluation results of each preference aspect across all tasks in Figure 9. As can be seen, for predictability and explorability, users preferred the LLM-reasoned widgets with libraries over those without any library. In terms of efficiency, LLM-reasoned widgets with libraries were slightly more favored than those without. When the libraries were incorporated, withlib30 was more preferred than withlib10 and withlib25 for predictability and explorability, while for efficiency, withlib25 and withlib30 were almost equally favored and more preferred than withlib10.

7.4.2 *Results of Each Aspect in Each Task.* Next, we examine user preferences for LLM-reasoned widgets per aspect and task. Figures 10 and 11 depict results of the task sets 1 and 2. As observed, user preferences show varying trends depending on the task.

In task set 1, for `image_adjust_exposure` (see Figure 13 for LLM-reasoned widgets), widgets reasoned with libraries were more preferred than those without a library for both predictability and explorability, with withlib25 and withlib30 being nearly equally favored. For efficiency, withlib25 and withlib30 were preferred over `withoutlib`, and `withoutlib` was favored over withlib10. When LLM-reasoned widget types were similar by different libraries, such as 60%

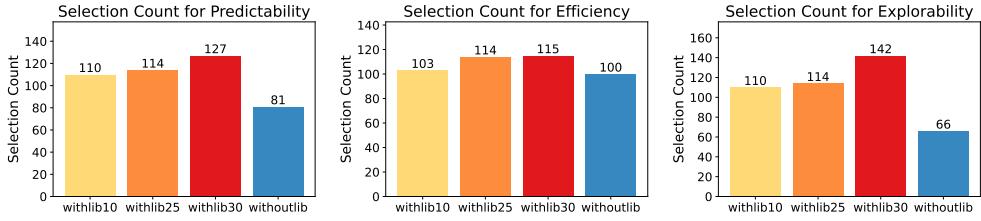


Fig. 9. Total user preference selection counts of three preference aspects across all tasks.

preset buttons and 40% slider by withlib25, and 70% preset buttons and 30% slider by withlib30, participants' feedback suggested that the slight score difference and LLM-provided reasoning helped guide their preference selection. For instance, participants chose withlib30 due to "good reasonings for any user which would prefer an easier time with Preset Buttons of higher score".

For image_adjust_temperature (see Figure 15 for LLM-reasoned widgets), withoutlib was preferred for predictability and almost equally favored as withlib25 and withlib30 for efficiency. Participants who preferred sliders reasoned by withoutlib over the color wheel and color picker reasoned with the libraries highlighted the slider's predictability. For example, they commented,

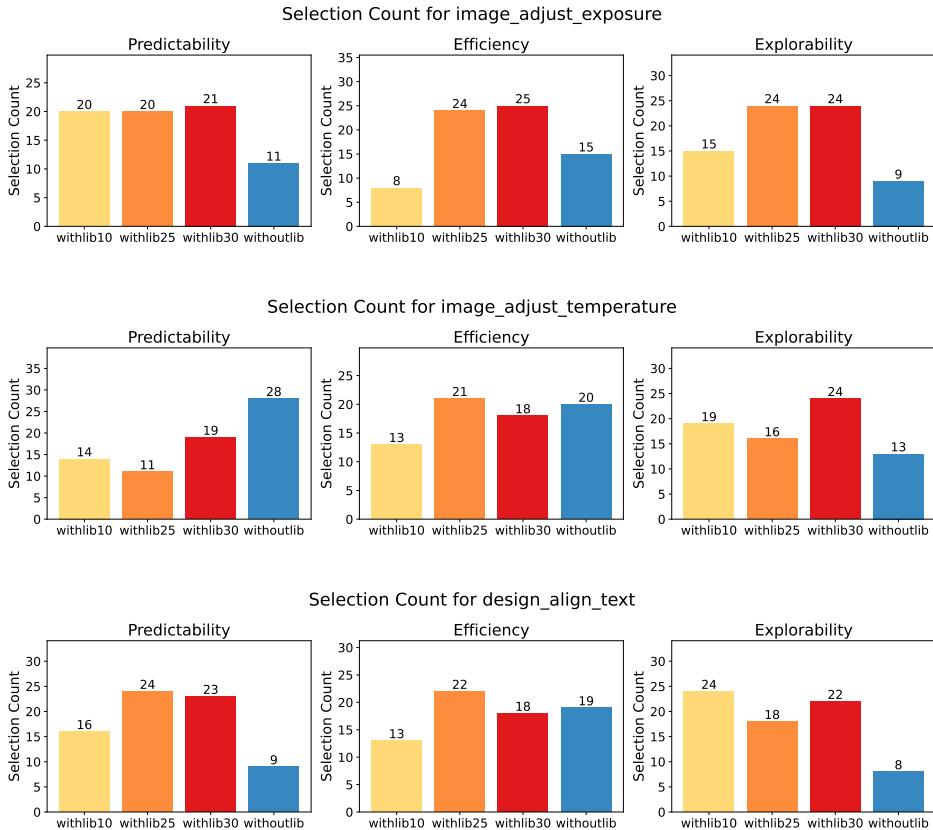


Fig. 10. User preference selection counts of tasks in task set 1.

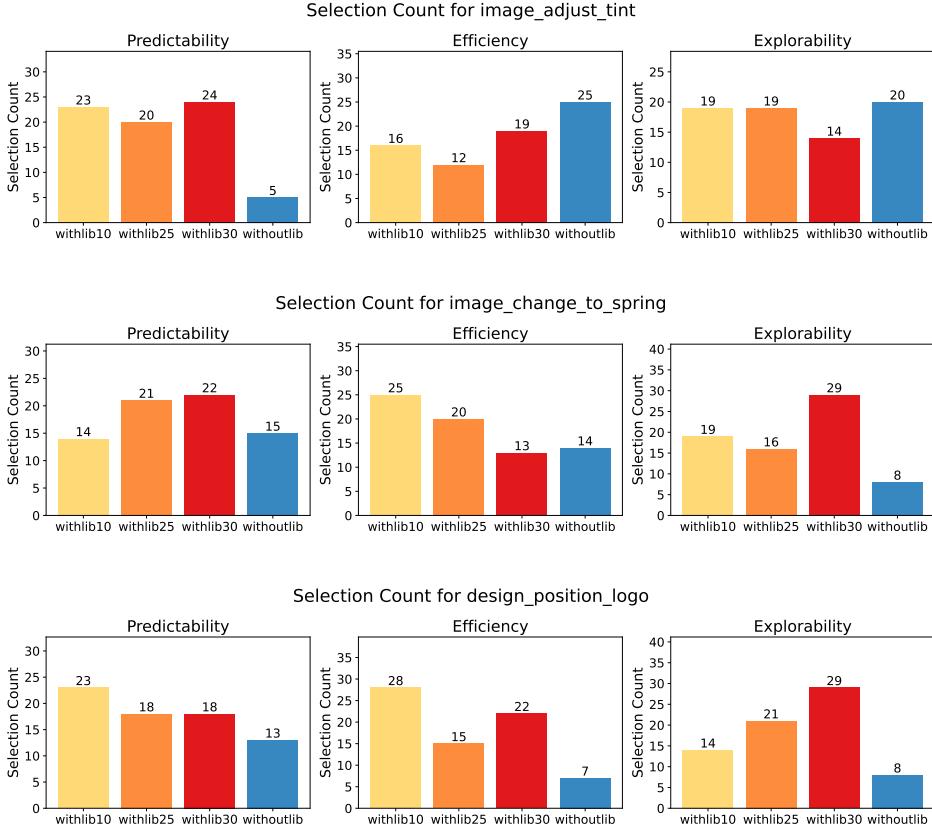


Fig. 11. User preference selection counts of tasks in task set 2.

“the slider is the most instantly understandable widget... the color wheel and color picker require a bit of deduction from the user on how to utilize them properly”. They also remarked on sliders’ sensitivity to value changes, such as *“I like the sensibility of sliders in general, and I feel they are a very efficient means of getting a good result by eye, just by dragging your mouse left or right, so this is the best”*. However, widgets reasoned with libraries were favored for explorability. For instance, participants preferred the color picker reasoned by withlib10 because *“you are given 3 RGB values each ranging from 0 to 255, giving you 255^3 possibilities”*, as opposed to the slider reasoned by withoutlib since *“you are only given 100 options as the slider moves in intervals of 0.01”*.

With regard to design_align_text (see Figure 17 for LLM-reasoned widgets), widgets reasoned with libraries were again more favored for predictability, and trends similar to image_adjust_temperature were observed for efficiency and explorability.

In task set 2, for image_adjust_tint (see Figure 8 or 14 for LLM-reasoned widgets), widgets reasoned with libraries remained more favored for predictability compared to those without a library. For efficiency, withoutlib was the most preferred option, indicating that the preset buttons were favored over the color picker and color wheel reasoned with libraries. Regarding explorability, withlib10, withlib25, and withoutlib were nearly equally favored, showing that participants preferred the color wheel and color picker for exploration over sliders reasoned by withlib30.

In terms of `image_change_to_spring` (see Figure 16 for LLM-reasoned widgets), `withlib25` and `withlib30` were almost equally preferred, both more favored than `withlib10` and `withoutlib` for predictability. In terms of efficiency, `withlib10` and `withlib25` were preferred over `withlib30` and `withoutlib`. For explorability, widgets reasoned with libraries were consistently more favored than those without any library.

Regarding `design_position_logo` (see Figure 18 for LLM-reasoned widgets), widgets reasoned with libraries were more favored across all three aspects. `Withlib10` was the most preferred for predictability and efficiency, while `withlib30` was the most favored for explorability. For predictability and efficiency, `withlib10` reasoned similar widgets (click on image) to those reasoned by other libraries, but it was preferred over `withlib25` and `withlib30` largely due to its reasoning. For instance, participants commented, “*Option 1 [withlib10] made more sense because it said how it takes minimal effort, instant feedback, and it’s quick and precise. Whereas option 2 [withlib30] said it’s just simple and takes minimal effort*”.

7.4.3 Reflection on Research Questions. For **RQ1**, the results reveal that the user preferences collected from crowdsourcing can effectively generalize to tasks beyond the original crowdsourcing tasks by enhancing the LLM’s reasoning capabilities. Broadly speaking, across all user evaluation tasks, widgets reasoned with the library are largely favored over those reasoned without a library. This highlights the adaptability and relevance of user preferences captured through crowdsourcing, suggesting that these insights can serve as a foundation to inform widget reasoning across a range of related tasks. While the overall trend favors library-informed reasoning, some exceptions emerge for specific tasks. For instance, for predictability in `image_adjust_temperature` and both efficiency and explorability in `image_adjust_tint`, widgets reasoned without the library are preferred. These results suggest that certain tasks may require additional consideration in the reasoning process.

For **RQ2**, the results demonstrate that incorporating the crowdsourced library enhances the ability of the LLM to reason widgets that align with user preferences regarding predictability, efficiency, and explorability. In terms of predictability, users’ preferences captured from the crowdsourcing study lean towards preset buttons for predefined value selection, color wheels, and color pickers for color adjustments, and click on image for position changes (recall Section 6.3.2). These preferences have directly influenced the widgets reasoned for the user evaluation tasks, where widgets reasoned with the library adhered more closely to these favored widget types. In contrast, when the library is not used, the LLM reasons sliders for most tasks regardless of the task categories. For efficiency, similar patterns emerge—preset buttons, color wheels, color pickers, and click on image are repeatedly preferred for their ease and speed in achieving task goals. The LLM, when equipped with the crowdsourced library, is able to reason toward these efficient widgets, whereas, without the library, the reasoning leans more heavily on sliders and preset buttons for most tasks without too much differentiation for task categories. Regarding explorability, the user preference trends show that sliders are favored for continuous value adjustments, color wheels are preferred for color changes, and click on image is liked for position adjustments. Without the library, however, the LLM tends to reason sliders to allow exploration for continuous value adjustments, color pickers for color changes, and sliders and text fields for position adjustments. The incorporation of the crowdsourced library enables the LLM to capture the need for exploration in tasks like color change and position adjustment by effectively aligning the widget choices with user preferences.

For **RQ3**, we observe that `withlib30`, containing the most crowdsourced data, emerges as the most preferred library across participants, particularly for predictability and explorability, while performing almost equally with `withlib25` in terms of efficiency. This pattern, supported by the distribution of user preferences across all tasks, as shown in Figure 12, highlights `withlib30` as the most reliable for all three aspects due to the richness of its crowdsourced data. This consistency

of user preferences underscores the value of extensive, diverse data in reasoning well-aligned widget options for users. On the other hand, *withoutlib* shows more significant variability in user preferences, particularly in predictability and efficiency, suggesting that when no library is used, LLM reasoning lacks the consistency that is otherwise afforded by crowdsourced data. While *withlib10* and *withlib25* are moderately favored for predictability and efficiency, they do not perform as well in explorability as *withlib30*, further emphasizing the benefit of larger datasets in supporting widget reasoning for tasks requiring exploration. However, examining each task individually reveals significant variation in preferred libraries for different preference aspects. This indicates that while larger libraries lead to consistent preference-aligned widget reasoning across multiple tasks in general, for individual tasks, the utility of LLM-reasoned widgets for specific preference aspects may need additional evaluation.

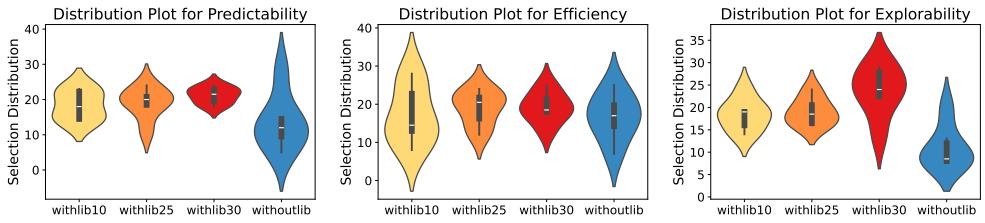


Fig. 12. Distribution of user preference selection counts of three preference aspects across all tasks.

8 Discussion

Based on the analysis of user evaluation results, this section discusses the implications of these results from three perspectives: the balance between task-generic and task-specific LLM widget generation, the alignment with user preferences for user-centered UI design, and the use of user preference data to create UIs that adapt to personalized use cases and evolve over time.

8.1 Generalizability of User Preference Data to Support Various Tasks

The user evaluation findings suggest that crowdsourced UI widget preference libraries provide an effective foundation to improve LLM reasoning across tasks beyond the original scope of crowdsourcing. From the tasks tested in the user evaluation, we have shown the task generalizability not only in the image editing domain but also in the design domain (recall tasks 5 and 6 in Table 2). This generalization ability can reduce the need for task-specific widget customization using automatic LLM widget generation, streamlining the UI design process. To further enhance this generalizability, the scope of crowdsourced tasks can be expanded by incorporating a more comprehensive array of task categories beyond continuous value adjustment, discrete value selection, and color and position changes. In addition, we can deepen the exploration within specific task categories to involve more complex aspects of data manipulation. For instance, our study emphasized tasks that required exploration and precision within the three task categories, and this can be extended to tasks on content editing or value adjustment that emphasize simplicity, speed, or flexibility. Also, while we focused on single-value adjustment, this setup can be extended to multi-dimensional data manipulation or compound parameter adjustments. By carefully curating the tasks for crowdsourcing, we envision that LLMs can be equipped with crowdsourced insights to reason UI widgets in both task-generic and task-specific contexts, adapting to diverse user needs and task scenarios.

Moreover, combining task-generic and task-specific UI widget reasoning could provide a solution for cases where generalized user preferences alone are insufficient. In our user evaluation, preference

aspects of a few tasks showed that user preferences were better supported without the library (recall Sections 7.4.2 and 7.4.3). These findings highlight that crowdsourced preferences do not universally apply to all tasks, and some require more refined, context-sensitive reasoning. To address this, an adaptive approach that integrates task-specific customization with the general crowdsourced library can potentially better accommodate unique user interaction demands to ensure the LLM delivers more tailored and effective UI widget suggestions across a wide range of tasks. For example, for the efficiency aspect of `image_adjust_tint`, users showed a preference for preset buttons generated by `withoutlib` over the color wheel and color picker reasoned with the library. Incorporating this insight into the LLM reasoning process by assigning higher weights to preset buttons for tasks involving image tint adjustments and related operations can lead to more preference-aligned UI generation than directly using the library. This approach leverages the flexibility of LLM-based widget generation by taking updated user insights to adapt to both generic and specific task requirements.

8.2 Towards User-Centered UI Widget Selection and Design

The crowdsourcing results reveal significant variation in user preferences for UI widgets based on different data manipulation needs, emphasizing the importance of a user-centered approach to UI widget selection. This approach contrasts with the traditional designer-dominant method, where UI widgets are often pre-selected by software designers. For instance, tools like Photoshop provide a slider and a text field for adjusting image lightness, saturation, and hue, which offer a one-size-fits-all solution. However, our crowdsourced data shows that users prefer distinct widgets for various data manipulation requirements, such as preset buttons for efficient adjustment of lightness and saturation and a color wheel to explore multiple hue options (recall Section 6.3.2). This divergence reveals a gap between user-preferred and designer-selected UI widgets. This gap highlights that static, predefined UI widgets in most software may not always meet the diverse needs of users. LLM-supported UI generation offers a promising solution to bridge this gap by generating dynamic widgets. With the crowdsourced preference library, we have extended the existing LLMs' ability to reuse user-preferred widgets based on user preference insights and, therefore, generate preference-aligned widgets for a broad range of tasks. This framework can be used to enable the creation of user-centered interfaces that align with the preferences collected from users to augment the traditional UI widget selection method that heavily relies on designers.

Furthermore, the user-centered UI design approach can be expanded to accommodate a broader spectrum of preference aspects and cater to more diverse user groups. Our study focuses on three aspects of user preferences, demonstrating the prototype's effectiveness in guiding LLM reasoning, but future research could incorporate additional dimensions of UI design. For example, prior work on UI design principles [14, 32, 37], which covers a number of guidelines, can be used to inform crowdsourcing efforts. Among them, accessibility is a particularly important dimension that can be addressed by collecting user preferences from individuals with disabilities. Crowdsourced data from these users could inform the design of accessible UI widgets, ensuring that interfaces meet their specific needs. This method would not only aid LLMs in generating accessible widgets that benefit users but also provide valuable insights for software designers. For example, LLMs can be equipped with crowdsourced data to suggest widget sizes optimized for low-vision users or alternative interaction techniques for those with motor impairments. They can also be prompted to offer reasonings for their suggestions, which could serve as references for design rationale to assist designers. Thus, the LLM-reasoned widgets can potentially augment the ideas proposed in prior research, such as SUPPLE [26] and SUPPLE++ [27], to develop dynamic UIs that accommodate motor and vision capabilities.

8.3 Opportunities for Adaptive UIs

The comparison between different library sizes reveals that larger libraries tend to produce more consistent preference-aligned widget reasoning across various tasks; meanwhile, it highlights the potential of smaller libraries for cost-effective and adaptive UI design. Our study shows that even with a library of just 10 user responses, i.e., `withlib10`, key insights into user preferences can be effectively captured. In some tasks and aspects, such as the predictability of adjusting image exposure and tint, the efficiency of changing colors to spring tones and positioning a logo, and the explorability of aligning text and adjusting tint, smaller libraries performed as well or even better than larger ones. Small-scale libraries are quick to collect and easy to evolve, providing efficient UI design suggestions for designers to reference in the design process. Even at the extreme, i.e., using no library at all, LLMs can reason UI widgets that, in some instances, outperformed those based on larger libraries. This demonstrates that LLMs can aid in UI widget reasoning and selection without requiring extensive human input, which offers designers a low-cost way to access user preferences during UI design. Such adaptability of small libraries can significantly reduce the time and resources required to gather large datasets while still yielding meaningful user-centered insights.

The adaptability enabled by small-scale libraries for UI design can be extended to deeper UI personalization, which reflects individual preferences and supports long-term UI evolution. By collecting a few samples from individual users, these libraries can be tailored to represent personalized preferences for UI widget generation. Such personalization can allow users to build workflows that align with their specific needs. Moreover, personalized libraries have the potential to evolve alongside users as their skills and expertise grow. For example, the LLM could initially generate simple, beginner-friendly widgets and, over time, adapt to offer more advanced, feature-rich widgets as the user becomes more proficient by using the user's updated personal library it uses for widget reasoning. This dynamic evolution enables users to progressively engage with more complex tools to support an adaptive and long-term learning curve. This kind of UI evolution not only streamlines immediate tasks but also fosters continuous improvement in tool use, helping users to gradually master more sophisticated functions without overwhelming them at the start.

9 Limitations and Future Work

The first limitation lies in the range of preferences covered during crowdsourcing. Our study focused on three key aspects, but there are other significant UI design principles, such as accessibility, cognitive load, and adaptability [32], that were not explicitly addressed here. However, our framework is highly extendable and can be enhanced by incorporating a broader set of preference dimensions with customizable aspects to account for user preferences from different perspectives.

Next, we mainly considered software-based UI widget generation in this work. While these widgets suited our tasks, other potential options, like gesture-based or tangible widgets, were not included. This may limit our understanding of how different widgets, like those supporting spatial interactions in XR [7] or physical interactions in tangible interfaces [12], could meet specific task needs or user preferences. Expanding the widget types in future research could offer comprehensive insights into user preference across a broader spectrum of tasks and interaction styles.

Lastly, our system implementation is currently focused on UI generation within the image editing domain. In the future, we aim to collect data from multiple domains and various tasks to build a more extensive user preference library. This would allow us to evaluate how a cross-domain library can improve widget generation, both for in-domain tasks and for tasks outside the collected domains. Such a system would also introduce challenges, such as how to effectively manage large datasets and ensure the LLM fully leverages all the available knowledge.

10 Conclusion

In this paper, we introduced a UI widget preference library collected from a crowdsourcing study and a framework for LLM-driven widget generation enhanced by this library. Our evaluation of the framework demonstrated its ability to extend widget generation across diverse but related tasks, align generated widgets with user preferences captured through crowdsourcing, and use large libraries for consistent, preference-aligned widget reasoning while leveraging smaller libraries for cost-effective preference capture. We further discussed these results to explore the extendability of crowdsourcing tasks, preference aspects, and library sizes in supporting UI generation tailored to task-generic and task-specific use cases, diverse user preferences and needs, and adaptive workflows. We envision our framework to benefit both users and designers by simplifying the development of user-centered interfaces that enhance task efficiency and user satisfaction.

References

- [1] Adobe. 2024. *Adobe Firefly*. Adobe. <https://www.adobe.com/products/firefly.html>
- [2] Adobe. 2024. *Adobe Illustrator*. Adobe. <https://www.adobe.com/products/illustrator.html>
- [3] Adobe. 2024. *Adobe Photoshop*. Adobe. <https://www.adobe.com/products/photoshop.html>
- [4] Adobe. 2024. *Adobe Photoshop Generative Fill*. Adobe. <https://www.adobe.com/products/photoshop/generative-fill.html>
- [5] Adobe. 2024. *Adobe Premiere Pro*. Adobe. <https://www.adobe.com/products/premiere.html>
- [6] Adobe. 2024. *Adobe Premiere Pro with Sensei AI*. Adobe. <https://www.adobe.com/products/premiere/ai-video-editing.html>
- [7] Alejandro Aponte, Arthur Caetano, Yunhao Luo, and Misha Sra. 2024. GraV: Grasp Volume Data for the Design of One-Handed XR Interfaces. In *Proceedings of the 2024 ACM Designing Interactive Systems Conference*. 151–167.
- [8] Shraddha Barke, Michael B James, and Nadia Polikarpova. 2023. Grounded copilot: How programmers interact with code-generating models. *Proceedings of the ACM on Programming Languages* 7, OOPSLA1 (2023), 85–111.
- [9] Blender. 2024. *Blender*. Blender. <https://www.blender.org/>
- [10] React Bootstrap. 2024. *React Bootstrap*. React Bootstrap. <https://react-bootstrap.github.io>
- [11] Tim Brooks, Aleksander Holynski, and Alexei A. Efros. 2023. InstructPix2Pix: Learning to Follow Image Editing Instructions. arXiv:2211.09800 [cs.CV] <https://arxiv.org/abs/2211.09800>
- [12] Daniel Campos Zamora, Mustafa Doga Dogan, Alexa F Siu, Eunyee Koh, and Chang Xiao. 2024. MoiréWidgets: High-Precision, Passive Tangible Interfaces via Moiré Effect. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*. 1–10.
- [13] Angel X. Chang, Mihail Eric, Manolis Savva, and Christopher D. Manning. 2017. SceneSeer: 3D Scene Design with Natural Language. arXiv:1703.00050 [cs.GR] <https://arxiv.org/abs/1703.00050>
- [14] Qiuyuan Chen, Chunyang Chen, Safwat Hassan, Zhengchang Xing, Xin Xia, and Ahmed E Hassan. 2021. How should i improve the ui of my app? a study of user reviews of popular apps in the google play. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 30, 3 (2021), 1–38.
- [15] Xin Chen, Biao Jiang, Wen Liu, Zilong Huang, Bin Fu, Tao Chen, and Gang Yu. 2023. Executing your Commands via Motion Diffusion in Latent Space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 18000–18010.
- [16] Yiru Chen. 2020. Monte carlo tree search for generating interactive data analysis interfaces. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2837–2839.
- [17] Yiru Chen, Ryan Li, Austin Mac, Tianbao Xie, Tao Yu, and Eugene Wu. 2022. NL2INTERFACE: Interactive Visualization Interface Generation from Natural Language Queries. arXiv:2209.08834 [cs.HC] <https://arxiv.org/abs/2209.08834>
- [18] Yiru Chen and Eugene Wu. 2022. Pi2: End-to-end interactive visualization interface generation from queries. In *Proceedings of the 2022 International Conference on Management of Data*. 1711–1725.
- [19] Ruijia Cheng, Titus Barik, Alan Leung, Fred Hohman, and Jeffrey Nichols. 2024. BISCUIT: Scaffolding LLM-Generated Code with Ephemeral UIs in Computational Notebooks. *arXiv preprint arXiv:2404.07387* (2024).
- [20] Charles J Colbourn. 2010. *CRC handbook of combinatorial designs*. CRC press.
- [21] Descript. 2024. *Descript*. Descript. <https://www.descript.com/>
- [22] Victor Dibia. 2023. LIDA: A Tool for Automatic Generation of Grammar-Agnostic Visualizations and Infographics using Large Language Models. arXiv:2303.02927 [cs.AI] <https://arxiv.org/abs/2303.02927>
- [23] Eureka Foong, Darren Gergle, and Elizabeth M Gerber. 2017. Novice and expert sensemaking of crowdsourced design feedback. *Proceedings of the ACM on Human-Computer Interaction* 1, CSCW (2017), 1–18.

- [24] Tsu-Jui Fu, Wenhe Hu, Xianzhi Du, William Yang Wang, Yinfei Yang, and Zhe Gan. 2024. Guiding Instruction-based Image Editing via Multimodal Large Language Models. arXiv:2309.17102 [cs.CV] <https://arxiv.org/abs/2309.17102>
- [25] Tsu-Jui Fu, Xin Eric Wang, Scott T Grafton, Miguel P Eckstein, and William Yang Wang. 2022. M3l: Language-based video editing via multi-modal multi-level transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 10513–10522.
- [26] Krzysztof Gajos and Daniel S Weld. 2004. SUPPLE: automatically generating user interfaces. In *Proceedings of the 9th international conference on Intelligent user interfaces*. 93–100.
- [27] Krzysztof Z Gajos, Jacob O Wobbrock, and Daniel S Weld. 2007. Automatically generating user interfaces adapted to users' motor and vision capabilities. In *Proceedings of the 20th annual ACM symposium on User interface software and technology*. 231–240.
- [28] Jeffrey Heer, Maneesh Agrawala, and Wesley Willett. 2008. Generalized selection via interactive query relaxation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 959–968.
- [29] Jupyter. 2024. *Interactive Canvas in Jupyter*. Jupyter. <https://ipyccanvas.readthedocs.io/en/latest/>
- [30] Jupyter. 2024. *Jupyter Widgets*. Jupyter. <https://ipywidgets.readthedocs.io/en/stable/>
- [31] Sang Won Lee, Rebecca Krosnick, Sun Young Park, Brandon Keelean, Sach Vaidya, Stephanie D O'Keefe, and Walter S Lasecki. 2018. Exploring real-time collaboration in crowd-powered systems through a ui design tool. *Proceedings of the ACM on Human-Computer Interaction* 2, CSCW (2018), 1–23.
- [32] William Lidwell, Kritina Holden, and Jill Butler. 2010. *Universal principles of design, revised and updated: 125 ways to enhance usability, influence perception, increase appeal, make better design decisions, and teach through design*. Rockport Pub.
- [33] Yimeng Liu and Misha Sra. 2024. DanceGen: Supporting Choreography Ideation and Prototyping with Generative AI. In *Proceedings of the 2024 ACM Designing Interactive Systems Conference*. 920–938.
- [34] Kurt Luther, Jari-Lee Tolentino, Wei Wu, Amy Pavel, Brian P Bailey, Maneesh Agrawala, Björn Hartmann, and Steven P Dow. 2015. Structuring, aggregating, and evaluating crowdsourced design critique. In *Proceedings of the 18th ACM conference on computer supported cooperative work & social computing*. 473–485.
- [35] Paula Maddigan and Teo Susnjak. 2023. Chat2VIS: generating data visualizations via natural language using ChatGPT, codex and GPT-3 large language models. *Ieee Access* 11 (2023), 45181–45193.
- [36] Angular Material. 2024. *Angular Material*. Angular Material. <https://material.angular.io>
- [37] Everett N McKay. 2013. *UI is communication: How to design intuitive, user centered interfaces by focusing on effective communication*. Newnes.
- [38] OpenAI. 2024. *DALL-E 3*. OpenAI. <https://openai.com/index/dall-e-3/>
- [39] OpenAI. 2024. *OpenAI GPT-4o*. OpenAI. <https://openai.com/index/hello-gpt-4o/>
- [40] Jonas Oppenlaender, Thanassis Tiropanis, and Simo Hosio. 2020. CrowdUI: Supporting web design with the crowd. *Proceedings of the ACM on Human-Computer Interaction* 4, EICS (2020), 1–28.
- [41] Ben Poole, Ajay Jain, Jonathan T. Barron, and Ben Mildenhall. 2022. DreamFusion: Text-to-3D using 2D Diffusion. *arXiv* (2022).
- [42] Prolific. 2024. *Prolific*. Prolific. <https://www.prolific.com/>
- [43] Bosheng Qin, Juncheng Li, Siliang Tang, Tat-Seng Chua, and Yuetong Zhuang. 2024. Instructvid2vid: Controllable video editing with natural language instructions. In *2024 IEEE International Conference on Multimedia and Expo (ICME)*. IEEE, 1–6.
- [44] Sigal Raab, Inbal Leibovitch, Guy Tevet, Moab Arar, Amit H. Bermano, and Daniel Cohen-Or. 2023. Single Motion Diffusion. arXiv:2302.05905 [cs.CV]
- [45] Runaway. 2024. *Runaway*. Runaway. <https://runwayml.com/>
- [46] Subham Sah, Rishab Mitra, Arpit Narechania, Alex Endert, John Stasko, and Wenwen Dou. 2024. Generating Analytic Specifications for Data Visualization from Natural Language Queries using Large Language Models. *arXiv preprint arXiv:2408.13391* (2024).
- [47] Debbie Stone, Caroline Jarrett, Mark Woodroffe, and Shailey Minocha. 2005. *User interface design and evaluation*. Elsevier.
- [48] Guy Tevet, Sigal Raab, Brian Gordon, Yonatan Shafir, Daniel Cohen-Or, and Amit H. Bermano. 2022. Human Motion Diffusion Model. arXiv:2209.14916 [cs.CV]
- [49] Priyan Vaithilingam, Elena L Glassman, Jeevana Priya Inala, and Chenglong Wang. 2024. DynaVis: Dynamically Synthesized UI Widgets for Visualization Editing. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*. 1–17.
- [50] Priyan Vaithilingam and Philip J Guo. 2019. Bespoke: Interactively synthesizing custom GUIs from command-line applications by demonstration. In *Proceedings of the 32nd annual ACM symposium on user interface software and technology*. 563–576.

- [51] Lea Verou, Amy X Zhang, and David R Karger. 2016. Mavo: creating interactive data-driven web applications by authoring HTML. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*. 483–496.
- [52] Vuetify. 2024. *Vue Component Framework*. Vuetify. <https://vuetifyjs.com/en/>
- [53] Chenglong Wang, Bongshin Lee, Steven Drucker, Dan Marshall, and Jianfeng Gao. 2024. Data Formulator 2: Iteratively Creating Rich Visualizations with AI. arXiv:2408.16119 [cs.HC] <https://arxiv.org/abs/2408.16119>
- [54] Chenglong Wang, John Thompson, and Bongshin Lee. 2023. Data Formulator: AI-powered Concept-driven Visualization Authoring. *IEEE Transactions on Visualization and Computer Graphics* (2023).
- [55] Chauncey Wilson. 2013. *User interface inspection methods: a user-centered design method*. Newnes.
- [56] Anbang Xu, Shih-Wen Huang, and Brian Bailey. 2014. Voyant: generating structured feedback on visual designs using a crowd of non-experts. In *Proceedings of the 17th ACM conference on Computer supported cooperative work & social computing*. 1433–1444.
- [57] Anbang Xu, Huaming Rao, Steven P Dow, and Brian P Bailey. 2015. A classroom study of using crowd feedback in the iterative design process. In *Proceedings of the 18th ACM conference on computer supported cooperative work & social computing*. 1637–1648.
- [58] Lixiu Yu, Aniket Kittur, and Robert E Kraut. 2016. Encouraging “outside-the-box” thinking in crowd innovation through identifying domains of expertise. In *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing*. 1214–1222.
- [59] Haoci Zhang, Viraj Raj, Thibault Sellam, and Eugene Wu. 2018. Precision interfaces for different modalities. In *Proceedings of the 2018 International Conference on Management of Data*. 1777–1780.
- [60] Shu Zhang, Xinyi Yang, Yihao Feng, Can Qin, Chia-Chih Chen, Ning Yu, Zeyuan Chen, Huan Wang, Silvio Savarese, Stefano Ermon, Caiming Xiong, and Ran Xu. 2024. HIVE: Harnessing Human Feedback for Instructional Visual Editing. arXiv:2303.09618 [cs.CV] <https://arxiv.org/abs/2303.09618>

A LLM-Reasoned UI Widgets for User Evaluation Tasks

Figures 13 to 18 show the LLM-reasoned UI widgets with crowdsourced libraries and without any library for the user evaluation tasks.

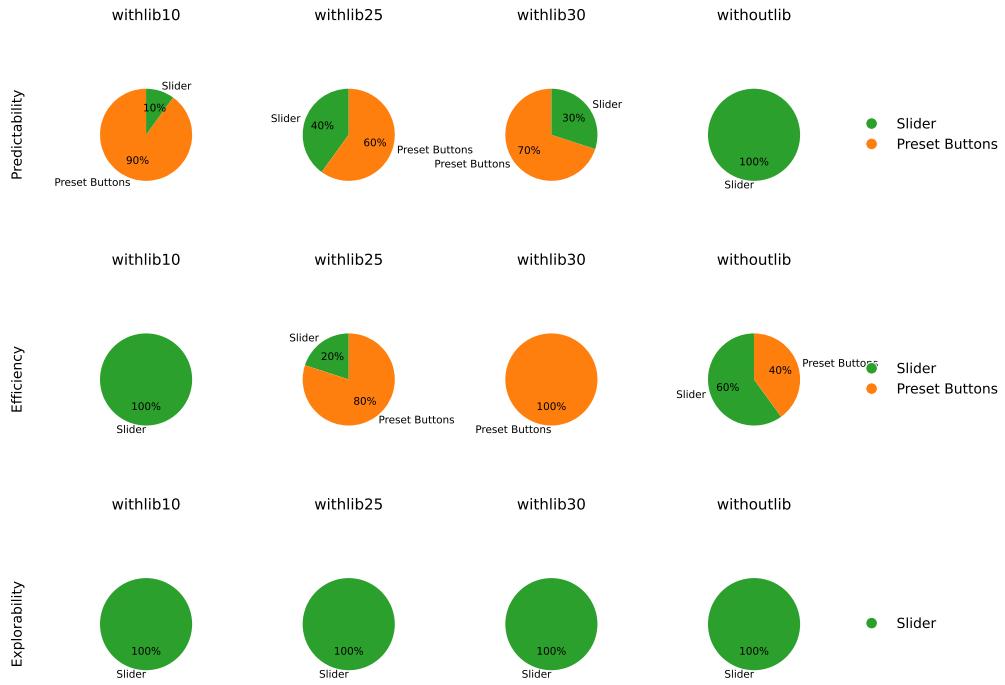


Fig. 13. LLM-reasoned UI widgets for `image_adjust_exposure` with three sizes of the crowdsourced libraries and without any library. Withlib10, withlib25, and withlib30 refer to using the crowdsourced libraries with 10, 25, and 30 user responses for all aspects and tasks, while withoutlib refers to using no library.

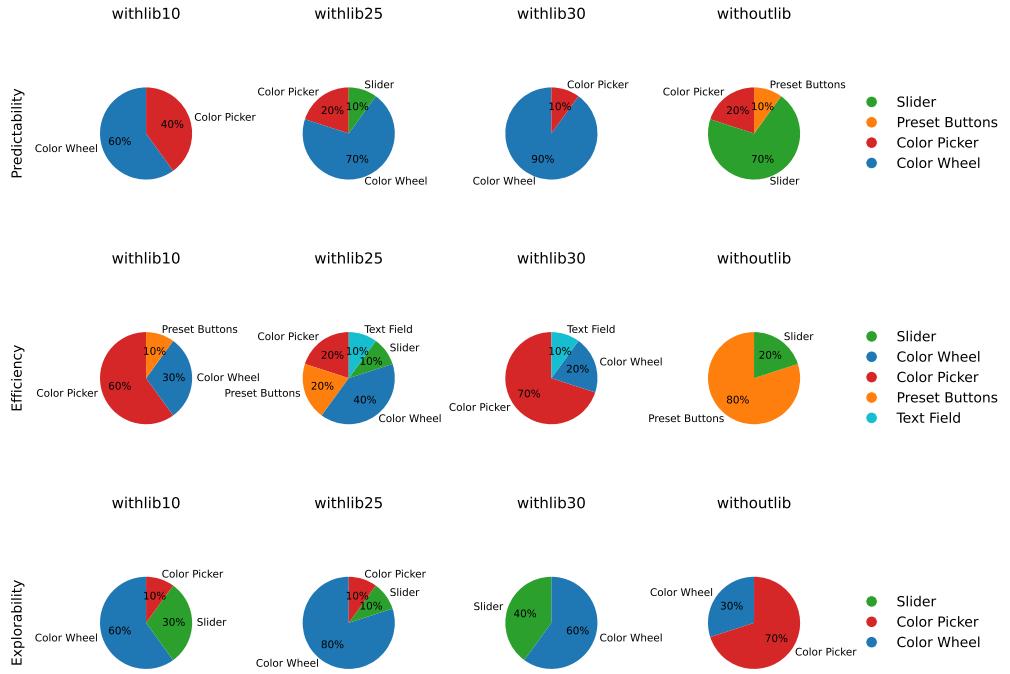


Fig. 14. LLM-reasoned UI widgets for `image_adjust_tint` with three sizes of the crowdsourced libraries and without any library. Withlib10, withlib25, and withlib30 refer to using the crowdsourced libraries with 10, 25, and 30 user responses for all aspects and tasks, while withoutlib refers to using no library.

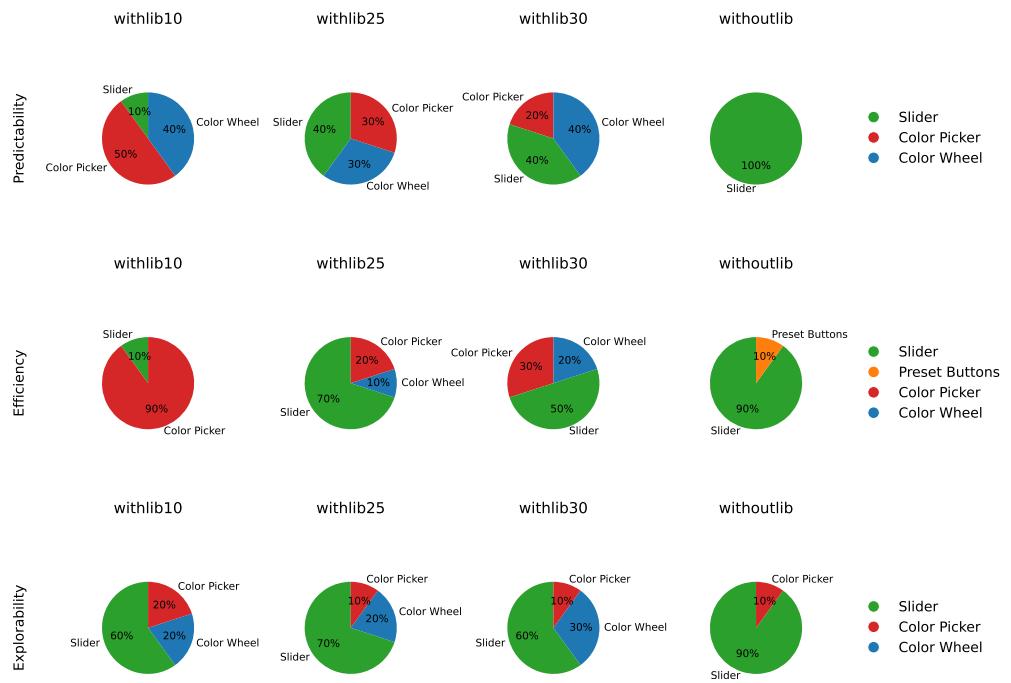


Fig. 15. LLM-reasoned UI widgets for `image_adjust_temperature` with three sizes of the crowdsourced libraries and without any library. Withlib10, withlib25, and withlib30 refer to using the crowdsourced libraries with 10, 25, and 30 user responses for all aspects and tasks, while withoutlib refers to using no library.

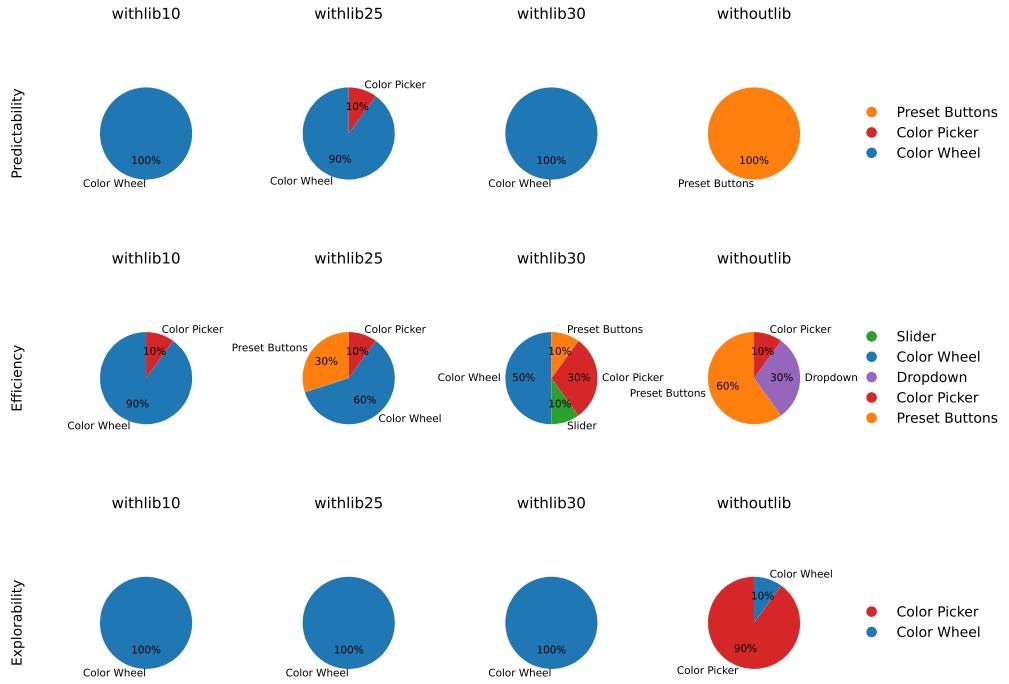


Fig. 16. LLM-reasoned UI widgets for `image_change_to_spring` with three sizes of the crowdsourced libraries and without any library. Withlib10, withlib25, and withlib30 refer to using the crowdsourced libraries with 10, 25, and 30 user responses for all aspects and tasks, while withoutlib refers to using no library.

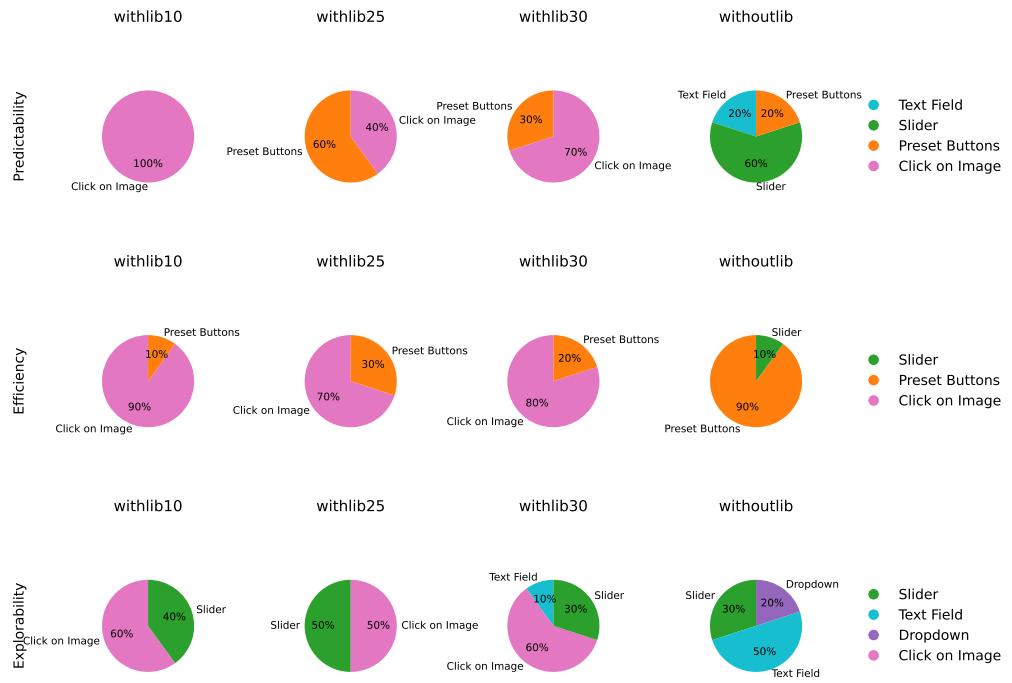


Fig. 17. LLM-reasoned UI widgets for `design_align_text` with three sizes of the crowdsourced libraries and without any library. `Withlib10`, `Withlib25`, and `Withlib30` refer to using the crowdsourced libraries with 10, 25, and 30 user responses for all aspects and tasks, while `Withoutlib` refers to using no library.

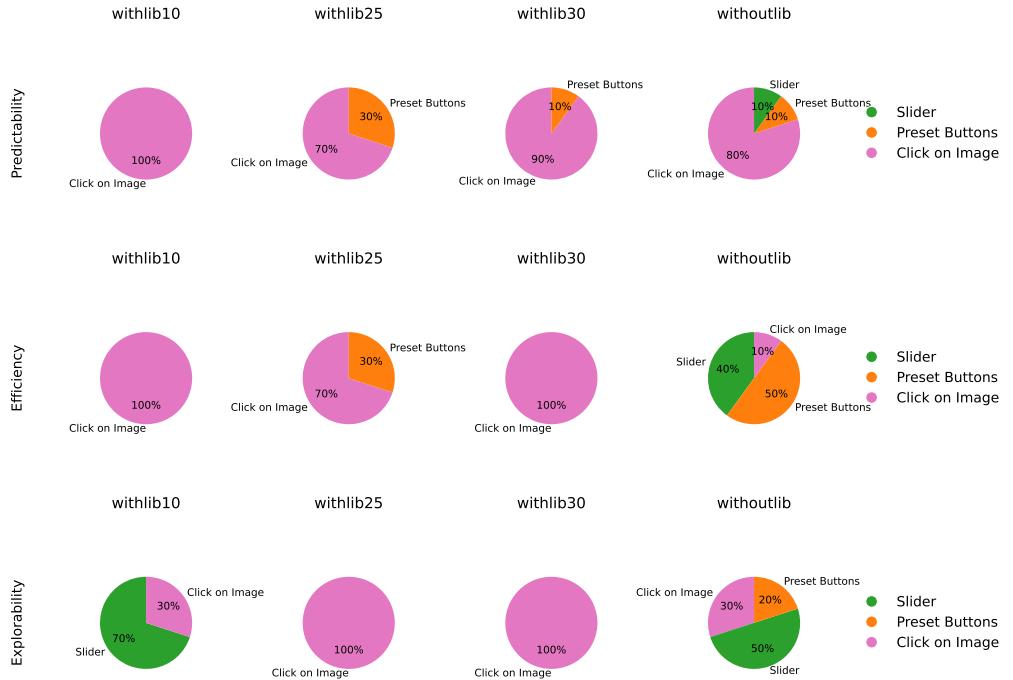


Fig. 18. LLM-reasoned UI widgets for `design_position_logo` with three sizes of the crowdsourced libraries and without any library. `Withlib10`, `Withlib25`, and `Withlib30` refer to using the crowdsourced libraries with 10, 25, and 30 user responses for all aspects and tasks, while `Withoutlib` refers to using no library.

B Prompts for LLM UI Widget Reasoning and Code Generation

B.1 Prompts for UI Widget Reasoning without Crowdsourced UI Preference Library

Reason UI widget type for the user task. You should follow the steps below for the reasoning.

First, please take the definitions below:

- Predictability: allows users to obtain results with no surprises or doesn't require users to deduce how to perform the interaction.
- Efficiency: allows users to perform tasks with a minimum amount of effort.
- Explorability: allows users to explore multiple possibilities and perform functions with high flexibility.

Second, reason the most proper UI widget for predictability, efficiency, and explorability.

- The UI widget you reason must from Slider, Preset Buttons, Dropdown, Radio Buttons, Text Field, Color Wheel, Color Picker, Click on Image.

Third, based on your reasoning, write down UI widgets for predictability, efficiency, and explorability in JSON format.

Refer to the example below to provide your response.

- Replace the placeholders marked by <> with your response. Do not include <> in your response.
- You must keep all the existing information from the example and only replace placeholders.
- The response must be in JSON format.

```
{
  "reasoning": {
    "task analysis": "<your reasoning>"

    "predictability_reasoning": {
      "<UI widget type>": "<your reasoning>"
    }

    "efficiency_reasoning": {
      "<UI widget type>": "<your reasoning>"
    }

    "explorability_reasoning": {
      "<UI widget type>": "<your reasoning>"
    }
  ...
}

"widget": {
  "<task_name>": {
    "predictability": "<UI widget type>",
    "efficiency": "<UI widget type>",
    "explorability": "<UI widget type>"
  }
}
}
```

B.2 Prompts for UI Widget Reasoning with Crowdsourced UI Preference Library

Based on the crowdsourced UI widget preference library, reason UI widget type for the user task. You should follow the steps below for the reasoning.

First, please take the definitions below:

- Predictability: allows users to obtain results with no surprises or doesn't require users to deduce how to perform the interaction.
- Efficiency: allows users to perform tasks with a minimum amount of effort.
- Explorability: allows users to explore multiple possibilities and perform functions with high flexibility.

Second, information on the crowdsourced UI widget preference library is in the prompt:

- Crowdsourced UI widget preference library task description: detailed descriptions of all the tasks.
- Crowdsourced UI widget preference library widget frequency: the frequency of user-preferred widgets.
Large numbers mean the corresponding widget is preferred by more people.
- Crowdsourced UI widget preference library widget reasons: the reasons for user-preferred widgets.

Third, search for the most relevant tasks from the crowdsourced UI widget preference library.

- You can compare the given task and the tasks names in the library, and refer to their task descriptions to help you to find the relevant tasks.
- Additional task information:
 - "image_adjust_lightness", "image_adjust_saturation", and "image_adjust_hue" represent tasks that allow continuous value adjustment and discrete value selection, and "image_adjust_hue" is also related to color adjustment.
 - "image_adjust_fall_color", "image_color_match", and "image_adjust_color_balance" are related to color adjustment.
 - "image_place_watermark" and "image_place_vignette" are related to object positioning and discrete value selection.

Fourth, reason the most proper UI widget for predictability, efficiency, and explorability.

- Your reasoning should be based on the relevant tasks you found in the library.
- After you find the relevant tasks, refer to the content of "Predictability", "Efficiency", or "Explorability" in the widget frequency and widget reasons.
- You must refer to the widgets of high frequencies of the relevant tasks in widget frequency.
- You must refer to widget reasons to help your reasoning.
- The UI widget you reason must come from the given library.

Lastly, based on your reasoning, write down UI widgets for predictability, efficiency, and explorability in JSON format.

Refer to the example below to provide your response.

- Replace the placeholders marked by <> with your response. Do not include <> in your response.
- You must keep all the existing information from the example and only replace placeholders.
- The response must be in JSON format.

```
{
  "reasoning": ```reasoning
    "relevant tasks from the library": "<your reasoning>"

    "predictability_reasoning": {
      "<UI widget type>": "<your reasoning>"
    }

    "efficiency_reasoning": {
      "<UI widget type>": "<your reasoning>"
    }

    "explorability_reasoning": {
      "<UI widget type>": "<your reasoning>"
    }
  ```

 "widget": ```widget
 "<task_name>": {
 "predictability": "<UI widget type>",
 "efficiency": "<UI widget type>",
 "explorability": "<UI widget type>"
 }
  ```
}
```

B.3 UI Preference Library Appended to the UI Widget Reasoning Prompt

Crowdsourced UI widget preference library task description:

```
{
  "image_adjust_lightness": "Adjust the lightness of the image.",
  "image_adjust_saturation": "Adjust the saturation of the image.",
  "image_adjust_hue": "Adjust the hue of the image.",
  "image_adjust_fall_color": "Change the image to fall colors.",
  "image_color_match": "Match the color of the image to a given target color.",
  "image_adjust_color_balance": "Adjust the color balance of the image by adjusting the RGB channels.",
  "image_place_watermark": "Place a watermark on the image to balance between visibility and subtlety.",
  "image_place_vignette": "Darken the background area using the vignette effect except for a certain image area."
}
```

Crowdsourced UI widget preference library widget frequency:

```
{
```

```
"image_adjust_lightness": {
    "Predictability": {
        "Slider": 1,
        "Preset Buttons": 4
    },
    "Efficiency": {
        "Slider": 3,
        "Text Field": 1,
        "Radio Buttons": 1
    },
    "Explorability": {
        "Slider": 5
    }
},
"image_adjust_saturation": {
    "Predictability": {
        "Slider": 2,
        "Preset Buttons": 3
    },
    "Efficiency": {
        "Slider": 1,
        "Preset Buttons": 4
    },
    "Explorability": {
        "Text Field": 1,
        "Slider": 4
    }
},
"image_adjust_hue": {
    "Predictability": {
        "Color Wheel": 4,
        "Preset Buttons": 1
    },
    "Efficiency": {
        "Slider": 1,
        "Color Wheel": 1,
        "Preset Buttons": 2,
        "Radio Buttons": 1
    },
    "Explorability": {
        "Color Wheel": 4,
        "Slider": 1
    }
},
"image_adjust_fall_color": {
    "Predictability": {
        "Color Wheel": 4,
        "Preset Buttons": 1
    },
    "Efficiency": {
        "Preset Buttons": 2,
        "Slider": 1,
        "Radio Buttons": 1,
        "Color Wheel": 1
    },
    "Explorability": {
        "Color Wheel": 4,
        "Slider": 1
    }
},
"image_color_match": {
    "Predictability": {
        "Text Field": 2,
        "Color Picker": 3
    },
    "Efficiency": {
        "Text Field": 1,
        "Radio Buttons": 1,
        "Color Picker": 1,
        "Preset Buttons": 2
    },
}
```

```

    "Explorability": {
      "Color Wheel": 4,
      "Color Picker": 1
    }
  },
  "image_adjust_color_balance": {
    "Predictability": {
      "Color Picker": 4,
      "Slider": 1
    },
    "Efficiency": {
      "Color Picker": 3,
      "Text Field": 1,
      "Slider": 1
    },
    "Explorability": {
      "Slider": 3,
      "Color Picker": 2
    }
  },
  "image_place_watermark": {
    "Predictability": {
      "Click on Image": 3,
      "Preset Buttons": 1,
      "Dropdown": 1
    },
    "Efficiency": {
      "Click on Image": 2,
      "Preset Buttons": 3
    },
    "Explorability": {
      "Preset Buttons": 1,
      "Click on Image": 2,
      "Slider": 2
    }
  },
  "image_place_vignette": {
    "Predictability": {
      "Preset Buttons": 2,
      "Click on Image": 2,
      "Slider": 1
    },
    "Efficiency": {
      "Click on Image": 3,
      "Slider": 2
    },
    "Explorability": {
      "Click on Image": 2,
      "Preset Buttons": 1,
      "Text Field": 1,
      "Slider": 1
    }
  }
}

```

Crowdsourced UI widget preference library widget reasons:

```

{
  "image_adjust_lightness": {
    "Predictability": {
      "Slider": [
        "It's easy to understand that higher values will produce lighter images and viceversa"
      ],
      "Preset Buttons": [
        "The preset button's background allows for users to figure out the lightness relatively quickly, and when clicking on the buttons, the button updates the lightness to how it looks on the buttons, allowing for users to need minimal deduction to understand what this button does.",
        "It is like to have an example of what you getting as a result",
        "The numbers show how each setting may look.",
        "the preset buttons provided an easy adjustment to the picture."
      ]
    }
  }
}

```

```
        ],
        "Efficiency": {
            "Slider": [
                "I can move it to the area I like the most",
                "Easier to adjust and more intuitive",
                "the slider felt like the most efficient method to adjust the exposure. This method being something I am familiar with changing in Adobe programs."
            ],
            "Text Field": [
                "If a user were more experienced in photo editing, they can quickly adjust the lightness precisely to what they want it to be by simply typing in the value, making it quick and easy."
            ],
            "Radio Buttons": [
                "The values are displayed clearly and you just click one."
            ],
            "Explorability": [
                "Slider": [
                    "Fast and contains options to the valid range so I don't have to figure that out.",
                    "It allows a greater range with more fluid motion, it provides the result on the spot, and it feels natural. Can select more ranges from a wide variety to see different types of results.",
                    "Allows for a more precise lighting value, and quicker",
                    "There's a bigger range of options",
                    "I find there is a lot of sensitivity in the slider, which has greater impact."
                ],
                "image_adjust_saturation": [
                    "Predictability": [
                        "Slider": [
                            "Saturation is usually set through a slider in image editing software",
                            "using the slider was the easiest method to adjust the saturation."
                        ],
                        "Preset Buttons": [
                            "The preset buttons include a preview of how the saturation would look like. For example, -1.0 is fully gray while 1.0 is a bright rainbow. This allows for users who may not know what saturation means to understand what each button does, it changes how \"colorful\" the image is.",
                            "It is easy to see the level of saturation that you want",
                            "It shows what the colors are like and then I go from there."
                        ],
                    ],
                    "Efficiency": [
                        "Slider": [
                            "Fast and easy to set it to a value I like."
                        ],
                        "Preset Buttons": [
                            "The ability to view gradients and brightness on the preset buttons really gave me an advantage. I had an idea of the color spectrums brightness and what to expect, while the other options was almost like choosing in the dark and waiting for a surprise.",
                            "Easier to use, and provides a glimpse into the saturation levels that radio buttons do not.",
                            "You can just click the button and see the result assuming it's a simple task like this one to just make colors pop",
                            "This makes editing easier, as the values are predetermined."
                        ],
                    ],
                    "Explorability": [
                        "Text Field": [
                            "I can enter specific values and see their results"
                        ],
                        "Slider": [
                            "The slider allows users to slowly go through each increment and explore what effect they have on the image. They can go anywhere within the range to understand the effect of saturation on the image and precisely pinpoint which exact value of saturation they want with a slider."
                        ],
                        "It is intuitive to increase the level of saturation just with the slider bar",
                        "It discretely lists all possible colors and lets me just pick one while I scroll.."
                    ],
                ],
            ],
        }
    ],
}
```

```

        "the slider also contributed to the explorability."
    ],
},
},
"image_adjust_hue": {
  "Predictability": [
    "Color Wheel": [
      "Colors don't feel like they should map to an order, and the color wheel is the only option that doesn't try to order the colors.",
      "For selecting the hue in this widget the color wheel was the most efficient choice in my opinion. It covers all the colors and gives me a clear area to select, the others I don't have much of guidance until I chose the options and see the results.",
      "There's a lot more options and you know exactly what you're getting when you press an option .",
      "The color wheel provides more flexibility with color hues than the preset values."
    ],
    "Preset Buttons": [
      "The backgrounds of the preset buttons allow the user to already have a baseline as to what that hue value would then look like"
    ],
},
"Efficiency": [
  "Slider": [
    "I'm more familiar with using a slider to change the hue of an image"
  ],
  "Color Wheel": [
    "If the task were to match the image to the same hue as another image, the user could look at the reference image and quickly pinpoint it to which hue it matches closely to on the color wheel, without needing to explore additional options."
  ],
  "Preset Buttons": [
    "You obtain the colour that you choose",
    "the preset buttons provided the most efficient way to change the hue."
  ],
  "Radio Buttons": [
    "It gives a simple numeric representation which I can just click on."
  ],
},
"Explorability": [
  "Color Wheel": [
    "I can select different colours and check the results",
    "There are more colour options",
    "The color wheel shows how each color hue looks so I can see all combinations from the outside.",
    "the color wheel provided the most options to adjust the hue and explore other possibilities ."
  ],
  "Slider": [
    "The slider allows for users to see the exact hue value as opposed to the color wheel, and see what even a small 0.01 increment in hue can do for an image. A slider for this task allows for more exploration as users scrub through the slider and see the different hue changes."
  ],
},
},
"image_adjust_fall_color": {
  "Predictability": [
    "Color Wheel": [
      "Like before, colors don't have an order, so the wheel lets me know what I will get before clicking it.",
      "By being able to select a color on the wheel and it produce the widget result with that color I will say it's the most appropriate option for predictability. I wouldn't have to deduce much knowing the style of orange or yellow I chose is what will be applied.",
      "Color wheel makes it easier to pick the colour you want, allowing for a more direct choice as you see the colours at once.",
      "I find it easier to navigate the wheel due to be able to seeing the colours beforehand."
    ],
    "Preset Buttons": [
      "Simple interface that has preset values and you know what to expect"
    ],
},
}
,
```

```

    "Efficiency": {
        "Preset Buttons": [
            "I can quickly choose an option",
            "You can choose exactly the colour you want for the picture just with one click"
        ],
        "Slider": [
            "For this specific task, the slider is the most efficient option, because the other preset
            options do not create a autumnal atmosphere. The preset buttons only allow for red,
            where autumn is more orange or brown. The slider allowed for quickly scrubbing through
            each value to see what was most warm and orange, and then fine tuning the smaller 0.01
            values for the exact atmosphere vibes."
        ],
        "Radio Buttons": [
            "You have few choices and can just click a simple choice."
        ],
        "Color Wheel": [
            "The color wheel was the most efficient way to perform the task. I was able to click on which
            color I thought I would like."
        ]
    },
    "Explorability": {
        "Color Wheel": [
            "It's easier to test different possibilities and check the results",
            "The color wheel allows the user to explore different atmospheres for the image without
            having to figure out what 0-1.0 means in this context. The color wheel allows for the
            overall image's atmosphere to be adjusted and explored in different seasonal contexts
            like pink for spring, and blue for winter.",
            "It lets you see the whole color spectrum and choose a particular hue that is right for the
            situation.",
            "The color wheel also offered more visual help to identify the colors and make the task more
            accessible."
        ],
        "Slider": [
            "As you can choose different values and see how it is affecting on the picture"
        ]
    }
},
"image_color_match": {
    "Predictability": {
        "Text Field": [
            "Since I know what my target it, just typing it gives me what I want.",
            "Can enter the hex number and see the desired result"
        ],
        "Color Picker": [
            "The color picker would accurately allow me to pinpoint the target color green and not have a
            doubt that it is correct.",
            "As you know what colour you are picking there are no surprises",
            "You can match the color visually before selecting it"
        ]
    },
    "Efficiency": {
        "Text Field": [
            "Being able to directly type what I want is the faster than trying to use the slider or color
            picker or wheel."
        ],
        "Radio Buttons": [
            "It would be the easiest way to click the options while producing the results under minimal
            effort."
        ],
        "Color Picker": [
            "Allows you to directly choose the target colour, with minimal effort/attempts"
        ],
        "Preset Buttons": [
            "Simple interface with preset options. Little to no effort required",
            "This is much quicker to select, as the colors are preset"
        ]
    },
    "Explorability": {
        "Color Wheel": [
            "Easy to try out options by just clicking."
        ]
    }
}

```

```

    "I feel like the wheel would have the most amount of options due to the color palette
        available to choose from.",
    "Allows for many options to be explored quicker than any of the other options",
    "This allows you to experiment more with the colors"
],
"Color Picker": [
    "More explorability options and plenty of options to choose from."
]
},
},
"image_adjust_color_balance": {
    "Predictability": [
        "Color Picker": [
            "I'm familiar with using a colour picker to change the colours",
            "The color picker widget adjusts the overall image's color harmony to the picked color
                quickly, and does not make the process confusing for the user to figure out how much red
                , green, and blue they need for the image to turn out a certain color.",
            "You need only to click the colour that you want to",
            "The widget just lets you see a color and overlay it on the image."
        ],
        "Slider": [
            "I had no surprises using the slider to adjust the color balance."
        ]
],
"Efficiency": [
    "Color Picker": [
        "It feels more intuitive",
        "It is easy to choose a colour, however there are less options than for the slider",
        "You can see what colors you want and just go with them without altering numbers."
    ],
    "Text Field": [
        "For more experienced users, the text field allows for them to efficiently type out the exact
            values they want to adjust without additional interaction. If the image needed just a
            little more blue, they could add .08 to it quickly, instead of fiddling with the slider
            ."
    ],
    "Slider": [
        "The slider allowed me to perform the task with minimum effort. The slider felt more natural
            to use."
    ]
],
"Explorability": [
    "Slider": [
        "Fastest way to just try different options.",
        "With multiple colors, the sliders makes it the most efficient and easy to use. It allows me
            to switch from colors easily and change their RGB values easily.",
        "Is the quickest out of all the options to explore many values other than the colour picker,
            however, the colour picker makes it harder to change individual colour channels."
    ],
    "Color Picker": [
        "Allows you click into colors and enter manually the RGB values.",
        "This allows users to explore more with the color settings"
    ]
],
},
"image_place_watermark": {
    "Predictability": [
        "Click on Image": [
            "Clicking is both intuitive and I know what I'll get. It did take two clicks at least,
                because I thought the text would be centered on the click, but how it worked was quickly
                apparent so I could then adjust things as I wanted.",
            "As the user directly clicks on the image, they know exactly where the watermark will be
                placed.",
            "Text is there and you can just click on image and it will go exactly where you pressed."
        ],
        "Preset Buttons": [
            "The presets give an indication of what to expect and wouldn't confuse me in this task. I
                could choose the proper preset and it's identifiable where the water mark would go."
        ],
        "Dropdown": [
            "It is easier to understand where the cursor will be positioned"
        ]
    ]
}
}

```

```
        ],
    },
    "Efficiency": {
        "Click on Image": [
            "I can simply click where I want the watermark to be",
            "It is easier to choose the right position"
        ],
        "Preset Buttons": [
            "The typical locations a watermark would be placed in are listed in the preset buttons widget
             . In terms of efficiency, a user can quickly click on which location they want the
             watermark to appear and be done. This widget is also efficient in allowing the user to
             click through each option and explore where they want the watermark to be located
             quickly.",
            "They just give you a few options set out for you to pick from.",
            "The preset buttons are the most efficient way of adjusting the text."
        ],
    },
    "Explorability": {
        "Preset Buttons": [
            "I can test different variations easily"
        ],
        "Click on Image": [
            "If a user wants more control over where their watermark is placed, the 'Click on image'
             widget option is the best for this, because the user can click on exactly where they
             want the watermark to be, then fine tune it by clicking slightly, ultimately adjusting
             their watermark to their preferred location.",
            "You can totally visually discern where you want the watermarks and/or image to go to as it
             is continuous."
        ],
        "Slider": [
            "You can have more control by watching the numbers ",
            "The slider made the explorability more accessible to adjust the text."
        ],
    }
},
"image_place_vignette": {
    "Predictability": {
        "Preset Buttons": [
            "It's easier to understand where the circle will move to",
            "The preset buttons for this task allows for a user to quickly move the circle to common
             positions without any surprises. Users less familiar with photo editing can simply click
             on a button and know what it should be doing, and this widget does just that."
        ],
        "Click on Image": [
            "Very intuitive ",
            "I just click on a location and it goes where I want."
        ],
        "Slider": [
            "I feel that the slider came with no surprises. It was the easiest way to adjust the position
             of the circle."
        ],
    },
    "Efficiency": {
        "Click on Image": [
            "The only option that really works well.",
            "Lets the user quickly determine where the circle will be placed",
            "You can point to exactly you need t be"
        ],
        "Slider": [
            "The easiest method would be the slider because it will produce the results with the least
             amount of effort. It's not the most powerful method but it is still efficient to a
             degree.",
            "It was the easiest to get the best outcome without taking a long time."
        ],
    },
    "Explorability": {
        "Click on Image": [
            "Easy to just click in slightly different places as needed to get the circle where I want it
             .",
            "As it is the quickest method it also allows the user to check many positions more quickly
             than the other options."
        ]
    }
}
```

```
        ],
        "Preset Buttons": [
            "The presets give me the most ranges with options to choose from that are already covering a wide variety. It's flexible and also easy to use."
        ],
        "Text Field": [
            "Allows you enter exact co-ordinates giving more flexibility."
        ],
        "Slider": [
            "This provides more flexibility with the positioning"
        ]
    }
}
```

B.4 Prompts for UI Widget Code Generation

Generate code for the UI widgets to perform the specified task. You should follow the steps below for the coding.

First, the UI widget code you provide must allow users to perform the task specified in the content of User task.

Second, the UI widget code you provide must follow the examples offered below for the implementation of Slider, Dropdown, Radio Buttons, Text Field, Preset Buttons, Color Wheel, Color Picker, Click on Image.

- The UI widget types are in `widget_type`.
- Find the example code for the relevant UI widget types.
- When coding, you must only change the task to the specified task, implement the specified widgets, and keep the remaining code format the same as the example code.

Third, write your response in JSON format following the example responses below.

```
"task": "Adjust image hue",
"widget_type": "Slider, Dropdown, Radio Buttons, Text Field, Preset Buttons, Color Wheel, Color Picker",
"widget_code": [```
import numpy as np
import matplotlib.pyplot as plt
import ipywidgets as widgets
from IPython.display import display, clear_output
from PIL import Image
from skimage import data, img_as_ubyte
from matplotlib.patches import Wedge
import matplotlib.colors as mcolors

image = data.astronaut()
image = Image.fromarray(img_as_ubyte(image))

# Function to allow performing the task
def adjust_hue(image, hue):
    img_hsv = image.convert('HSV')
    np_img = np.array(img_hsv)

    hue_shift = int(hue * 255)

    np_img = np_img.astype(np.int32)
    np_img[..., 0] = (np_img[..., 0] + hue_shift) % 256

    np_img = np.clip(np_img, 0, 255).astype(np.uint8)

    adjusted_img = Image.fromarray(np_img, mode='HSV').convert('RGB')
    return adjusted_img

# Function to create widgets
def create_hue_widgets():
    # Slider
    slider_label = widgets.Label(value='Slider:')
```]
```

```
slider = widgets.FloatSlider(value=0.0, min=0.0, max=1.0, step=0.01)

Dropdown
dropdown_label = widgets.Label(value='Dropdown:')
dropdown = widgets.Dropdown(options=[0.0, 0.2, 0.4, 0.6, 0.8], value=0.0)

Radio Buttons
radio_buttons_label = widgets.Label(value='Radio Buttons:')
radio_buttons = widgets.RadioButtons(options=[0.0, 0.2, 0.4, 0.6, 0.8], value=0.0)

Text Field
text_field_label = widgets.Label(value='Text Field:')
text_field = widgets.BoundedFloatText(value=0.0, min=0.0, max=1.0, step=0.01)

Preset Buttons
preset_label = widgets.Label(value='Preset buttons:')

preset_hues = [
 (0.0, 'red'),
 (0.2, 'green'),
 (0.4, 'cyan'),
 (0.6, 'blue'),
 (0.8, 'magenta')
]

hue_mapping = {f"({hue})": hue for hue, color in preset_hues}

preset_buttons = [
 widgets.Button(
 description=f"({hue})",
 layout=widgets.Layout(width='75px', height='30px'),
 style={'button_color': color}
)
 for hue, color in preset_hues
]

preset_buttons_box = widgets.HBox(preset_buttons)

Color Wheel
color_wheel_label = widgets.Label(value='Color Wheel:')
color_wheel = widgets.Output()
create_color_wheel(color_wheel)

Color Picker
color_picker_label = widgets.Label(value='Color Picker:')
color_picker = widgets.ColorPicker(concise=True, value="#ffffff", disabled=False)

Layout
layout = widgets.Layout(align_items='flex-start')
spacer = widgets.Box(value='', layout=widgets.Layout(height='20px'))

Combine widgets into a vertical box
widgets_box = widgets.VBox([slider_label, slider, spacer,
 dropdown_label, dropdown, spacer,
 radio_buttons_label, radio_buttons, spacer,
 text_field_label, text_field, spacer,
 preset_label, preset_buttons_box, spacer,
 color_wheel_label, color_wheel, spacer,
 color_picker_label, color_picker, spacer], layout=layout)

return widgets_box, slider, dropdown, radio_buttons, text_field, preset_buttons, hue_mapping, color_wheel, color_picker

Function to create and display the color wheel
def create_color_wheel(output):
 with output:
 clear_output(wait=True)

 fig, ax = plt.subplots(figsize=(1.5, 1.5))

 num_colors = 360
```

```

theta = np.linspace(0, 2 * np.pi, num_colors, endpoint=False)
colors = plt.cm.hsv(theta / (2 * np.pi))

for i in range(num_colors):
 wedge = Wedge(center=(0, 0), r=1, theta1=(i * 360 / num_colors),
 theta2=((i + 1) * 360 / num_colors), color=colors[i],
 transform=ax.transData._b, clip_on=False)
 ax.add_patch(wedge)

ax.set_aspect('equal')
ax.set_xlim(-1.1, 1.1)
ax.set_ylim(-1.1, 1.1)
ax.axis('off')

fig.canvas.mpl_connect('button_press_event', on_color_wheel_click)

plt.show()

Function to handle color wheel click
def on_color_wheel_click(event):
 if event.inaxes:
 x, y = event.xdata, event.ydata
 theta = np.arctan2(y, x) % (2 * np.pi)
 hue = theta / (2 * np.pi)
 hue = round(hue, 2)

 update_plot(None, image, output, slider, dropdown, radio_buttons, text_field, preset_buttons,
 hue_mapping, hue=hue)

Function to convert hex color to hue value
def hex_to_hue(hex_color):
 rgb = np.array([int(hex_color[i:i+2], 16) for i in (1, 3, 5)]) / 255.0
 hsv = mcolors.rgb_to_hsv(rgb.reshape(1, 1, 3))
 hue = hsv[0, 0, 0]
 return hue

Function to update the image display based on widget values
def update_plot(change, image, output, slider, dropdown, radio_buttons, text_field, preset_buttons,
 hue_mapping, hue=None):
 with output:
 clear_output(wait=True)

 if hue is None:
 if change and change['owner'] in preset_buttons:
 clicked_button = change['owner']
 hue = hue_mapping[clicked_button.description]
 elif change and change['owner'] == slider:
 hue = slider.value
 elif change and change['owner'] == dropdown:
 hue = dropdown.value
 elif change and change['owner'] == radio_buttons:
 hue = radio_buttons.value
 elif change and change['owner'] == text_field:
 hue = text_field.value
 elif change and change['owner'] == color_picker:
 hue = hex_to_hue(change['new'])
 else:
 hue = 0.0

 adjusted_image = adjust_hue(image, hue)
 plt.figure(figsize=(4, 4))
 plt.imshow(adjusted_image)
 plt.axis('off')
 plt.title(f'Hue Adjustment: {hue:.2f}')
 plt.show()

Function to link widgets to the update function
def link_widgets_to_update(image, output, slider, dropdown, radio_buttons, text_field, preset_buttons,
 hue_mapping, color_picker):
 def callback(change):

```

```
update_plot(change, image, output, slider, dropdown, radio_buttons, text_field, preset_buttons,
hue_mapping)

slider.observe(callback, names='value')
dropdown.observe(callback, names='value')
radio_buttons.observe(callback, names='value')
text_field.observe(callback, names='value')
color_picker.observe(callback, names='value')
for btn in preset_buttons:
 btn.on_click(lambda btn: update_plot({'owner': btn}, image, output, slider, dropdown, radio_buttons,
 text_field, preset_buttons, hue_mapping))

Function to display the widgets and output plot
def display_hue_widgets():
 global output, slider, dropdown, radio_buttons, text_field, preset_buttons, hue_mapping, color_picker
 output = widgets.Output()

 widgets_box, slider, dropdown, radio_buttons, text_field, preset_buttons, hue_mapping, color_wheel,
 color_picker = create_hue_widgets()

 layout = widgets.Layout(width='100%', display='flex', justify_content='space-between')
 ui_and_image_box = widgets.HBox([widgets_box, output], layout=layout)

 display(ui_and_image_box)

 link_widgets_to_update(image, output, slider, dropdown, radio_buttons, text_field, preset_buttons,
 hue_mapping, color_picker)

 update_plot(None, image, output, slider, dropdown, radio_buttons, text_field, preset_buttons, hue_mapping
)

display_hue_widgets()

 ...}

}
```

## C User Evaluation Briefing and Instructions

Welcome!

This study evaluates user preferences for different UI widgets.

Your participation is voluntary. You are free to quit the study at any time. You can participate in this study only once.

All your responses (including demographic info) will be kept private and only used for research purposes.

### # Study Instructions

- You will be working on 3 tasks. Each task is followed by 6 questions asking your UI widget preference by selecting one of two provided options for each question.
- For each task, please follow the task description and interact with **all the provided UI widgets**.
- After interacting with the UI widgets, you will choose the preferences that make the most sense to you regarding the widgets' **predictability**, **efficiency**, or **explorability**.
  - Predictability: Allows you to obtain results with no surprises or doesn't require you to deduce how to perform the interaction.
  - Efficiency: Allows you to perform tasks with a minimum amount of effort.
  - Explorability: Allows you to explore multiple possibilities and perform functions with high flexibility
- When making your selection, please consider the following **comprehensive factors**:
  - **Widget types**: Whether the widget types make sense to you regarding predictability, efficiency, or explorability.
  - **Scores**: How much each widget is recommended in terms of predictability, efficiency, or explorability.
    - Each widget type has a score out of 10.
    - In each option, a widget with higher scores indicates a stronger recommendation.
    - The total score of all widgets in each option adds up to 10.
  - **Reasons**: Why each widget is a good choice to allow predictability, efficiency, or explorability.
    - Some reasons may refer to users' UI widget preferences when performing other relevant tasks. These reasons are based on a library. Assume that such a library exists.
    - If a reason is based on the library, you will need to assess how well the information from this library is used for reasoning.
- **Do NOT** base your preference selection on:
  - The number of widget types, e.g., choosing an option simply because it offers more widget types.
  - The length of reasoning, e.g., choosing an option simply because the reasons are longer.
- Remember, your preference selection should consider widget types, scores, and reasons **collectively**. You can go back and interact with the UI widgets if you want.
  - If widget types are the same, consider the scores and reasons to make your selection.
  - If widget types and scores are the same, consider the reasons to make your selection.
- All questions are required, including preference selection and an explanation of your reason.