석 사 학 위 논 문
Master's Thesis

도구 기반 사용자 인터페이스 설계를 위한 시각 언어 모델 벤치마크

A Benchmark for Vision-Language Models on Tool-Based User Interface Design

2026

정 대 헌 (鄭 大憲 Jeong, Daeheon)

한 국 과 학 기 술 원

Korea Advanced Institute of Science and Technology

석 사 학 위 논 문

도구 기반 사용자 인터페이스 설계를 위한 시각 언어 모델 벤치마크

2026

정 대 헌

한 국 과 학 기 술 원

김재철AI대학원

도구 기반 사용자 인터페이스 설계를 위한 시각 언어 모델 벤치마크

정 대 헌

위 논문은 한국과학기술원 석사학위논문으로
학위논문 심사위원회의 심사를 통과하였음

2026년 6월 11일

심사위원장 김 주 호 (인)

심 사 위 원 오 혜 연 (인)

심 사 위 원 김 현 우 (인)

A Benchmark for Vision-Language Models on Tool-Based User Interface Design

Daeheon Jeong

Advisor: Juho Kim

A dissertation submitted to the faculty of
Korea Advanced Institute of Science and Technology in
partial fulfillment of the requirements for the degree of
Master of Science in AI

Daejeon, Korea
June 11, 2026

Approved by

Juho Kim
Professor of Computer Science

The study was conducted in accordance with Code of Research Ethics[1].

[1] Declaration of Ethical Conduct in Research: I, as a graduate student of Korea Advanced Institute of Science and Technology, hereby declare that I have not committed any act that may damage the credibility of my research. This includes, but is not limited to, falsification, thesis written by someone else, distortion of research findings, and plagiarism. I confirm that my thesis contains honest conclusions based on my own careful research under the guidance of my advisor.

MAI 정대헌. 도구 기반 사용자 인터페이스 설계를 위한 시각 언어 모델 벤치마크. 김재철AI대학원 . 2026년. 54+v 쪽. 지도교수: 김주호. (영문 논문)
Daeheon Jeong. A Benchmark for Vision-Language Models on Tool-Based User Interface Design. Kim Jaechul Graduate School of AI . 2026. 54+v pages. Advisor: Juho Kim. (Text in English)

초 록

사용자 인터페이스 설계는 디자이너가 피그마와 같은 디자인 도구에서 요소를 배치하고 수정하며 결과를 반복적으로 다듬는 과정이다. 최근 시각언어 모델은 도구 호출을 통해 이러한 디자인 도구를 제어할 수 있게 되었지만, 실제로 도구를 이용해 사용자 인터페이스 설계 능력을 체계적으로 평가하는 기준은 부족하다. 이 논문은 디자이너들로부터 수집한 사용자 인터페이스들을 바탕으로 도구 기반 사용자 인터페이스 설계 벤치마크를 구성한다. 벤치마크는 (가) 전체 화면을 재현하는 디자인 복제와 (나) 기존 화면의 특정 부분을 바꾸는 디자인 수정으로 나뉜다. 모델은 상황에 맞는 도구를 여러 차례 호출해 목표 화면에 가까운 결과를 만든다. 평가는 시각적 구조, 어탠션 맵, 의미, 구성 요소 속성을 함께 고려한다. 본 연구는 실험 결과를 통해 높은 성능의 모델은 더 전략적으로 도구를 사용한다는 점을 확인하고, 반복적 설계에서 나타나는 주요 오류 유형을 검증하였다.

핵 심 낱 말 사용자 인터페이스 설계, 시각 언어 모델, 도구 호출, 벤치마크, 디자인 자동화

Abstract

User interface (UI) design is an iterative process in which designers progressively refine their work with design software such as Figma or Sketch. Recent advances in vision language models (VLMs) with tool invocation suggest these models can operate design software to edit a UI design through iteration. Understanding and enhancing this capacity is important, as it highlights VLMs' potential to collaborate with designers within conventional software. However, as no existing benchmark evaluates tool-based design performance, the capacity remains unknown. To address this, we introduce a benchmark for VLMs on tool-based user interface design. Our benchmark contains 598 tool-based design tasks paired with ground-truth references sampled from 3.3K mobile UI designs across 30 function-based categories (e.g., onboarding, messaging). In each task, a VLM updates the design step-by-step through context-based tool invocations (e.g., create a rectangle as a button background), linked to design software. Specifically, the benchmark incorporates two task types: (i) *design replication* evaluates the ability to reproduce a whole UI screen; (ii) *design modification* evaluates the ability to modify a specific part of an existing screen. Results suggest that leading models exhibit more strategic tool invocations, improving design quality. Furthermore, we identify common error patterns models exhibit, guiding future work in enhancing tool-based design capabilities.

Keywords User interface design, vision-language model, tool invocation, benchmark, design automation

Contents

List of Tables

List of Figures

Chapter 1. Introduction

User interface (UI) design is a nonlinear process in which designers experiment with ideas through back-and-forth actions and observations [8, 3]. In this process, designers continuously adjust, group, and arrange each design component, such as buttons and text, using design software (e.g., Figma and Sketch). Working with design software is essential for designers, as they provide familiar controls over design components for efficient iterations [33, 37, 36].

Recent vision-language models (VLMs) have demonstrated agentic tool invocation capabilities; a model can complete tasks by invoking tools over multiple turns [11, 27]. This capability suggests that VLMs could create and edit UI designs by interacting with design software using tool invocations (Fig. 1.1A). Yet, existing benchmarks solely focus on *code-based* capabilities for the implementation stage, such as generating HTML and CSS [18, 34, 22]. The earlier *tool-based* design stage, where designers iteratively refine interfaces using design software, remains underexplored. Understanding VLMs' tool-based capabilities is crucial for enabling collaboration during this design phase [23, 5].

To address this gap, we introduce a benchmark for evaluating VLMs' performance in tool-based UI design. The benchmark comprises 598 UI design tasks selected to evaluate the model's performance in two task scenarios: (i) *design replication* (Fig. 1.1B), where the model generates a design that replicates a given UI image, and (ii) *design modification* (Fig. 1.1C), where the model modifies an existing human-made design by applying a sequence of edits such as resizing a component. In each task, a VLM invokes context-based tool commands (e.g., *"create a rectangle as a button background"*) to interact with the design software [9], and iteratively construct a complete UI design. Each task is paired with ground-truth designs sampled from 3,327 human-crafted mobile UI designs across 30 functions (e.g., onboarding, messaging) collected from an online design community [10].

To evaluate model performance, the benchmark measures the similarity between generated and ground-truth designs across three hierarchical levels of visual perception, modeling how humans progressively interpret visual information: features, patterns, and objects. The evaluation includes structural similarity at the feature level (SSIM) [40], compositional similarity at the pattern level via saliency maps [15], and semantic interpretation at the object level using BLIP captions [21]. To capture fine-grained editing performance, we additionally evaluate component-level attributes [34].

In our experiments, we evaluate state-of-the-art VLMs capable of tool invocation using the benchmark. Our analysis reveals two key findings: (i) in replication tasks, model performance is closely linked to diverse and strategic tool usage; and (ii) in modification tasks, precise tool selection is critical, as a single incorrect action can cause large metric score shifts due to the granularity of the task. Through the two task configurations, the benchmark captures the model's capability to operate the design strategically, while precisely determining accurate tools. Furthermore, we identify the limitations of current VLMs in tool-based design through error analysis and discuss directions for future improvement. In summary, our contributions are:

- We introduce a benchmark to evaluate VLMs' ability to perform tool-based UI generation in an interactive, multi-turn design environment.
- We construct a dataset of 598 tool-driven design tasks, comprising replication and modification tasks, sampled from 3,327 UIs across 30 categories.

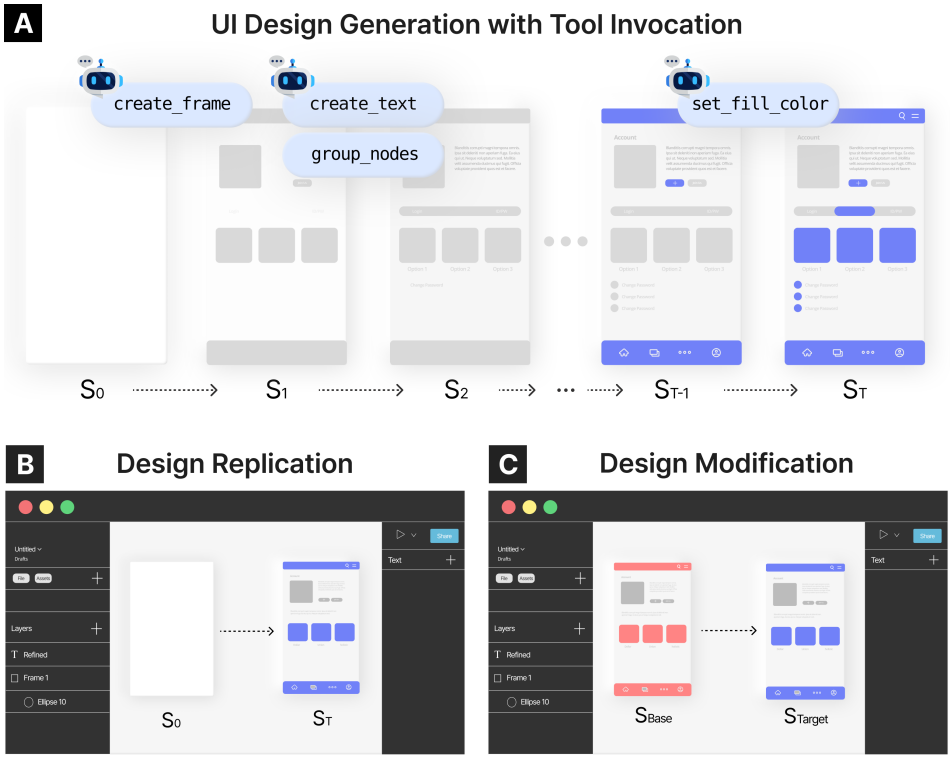


Figure 1.1: **Overview.** The benchmark evaluates a VLM's capability to (A) generate a UI design with tool invocations in two tasks: (B) design replication and (C) design modification.

- We conduct a comprehensive evaluation of five state-of-the-art VLMs and discuss key insights into their tool use behaviors through quantitative and qualitative analysis.

Chapter 2. Related Work

2.1 UI Design Generation

Research on UI design with generative models follows two approaches: code-based generation, which generates the code that renders a design, and image-based generation, which synthesizes the design image directly. Beltramelli's pix2code provides an early proof-of-concept for code-based UI design generation, translating UI screenshots into code using neural network models [2]. Subsequent research advances the models to condition on more abstract representations, such as sketches, wireframes, and text prompts [26, 16, 22]. Recent work has increased instruction compliance by incorporating human and self-generated feedback into a generation process [47, 14], training on the layout structure [38], and setting incremental steps [39]. Another line of research explored directly generating UI designs as images, including generative adversarial networks [19, 46] and diffusion models [4, 12]. These approaches are unconstrained by code-specific syntax and render design and layout images through a denoising process. Nevertheless, the generated result remains less editable than designs created with design software, restricting its practical applicability; the problem persists in code-based generation [44]. Contrary to previous approaches, the benchmark evaluates tool-based design generation: models produce designs with tool invocations in design software, creating directly editable outputs.

2.2 UI Design Datasets and Benchmark

Existing datasets and benchmarks on UI generation primarily focus on code-based generation. A common dataset structure pairs UI screenshots with corresponding source code (e.g., HTML, XML) to support code-based design generation [34, 45]. The datasets have increased the scale, starting from RICO on mobile UI design [7] to large-scale real-world datasets, such as WebCode2M [13] and WebSight [17]. These datasets enable training UI generation models and benchmarking by comparing generated images and source code with ground-truth designs. Evaluation metrics include pixel-level similarity, SSIM for structural comparison, CLIP for semantic alignment, and FID [46, 42, 22, 14]. Studies also utilize the source code to measure component-level attributes (e.g., text, position, color) [34, 14] and structural relationships [13]. Still, existing metrics have limited applicability to tool-based UI design, as design software structures data representation around designer-centric concepts such as masks and layers. The benchmark addresses these limitations by proposing metrics adapted to the tool-based design context.

Chapter 3. Benchmark Design

3.1 Task Design

To reflect real-world user scenarios, the benchmark consists of two types of user interface (UI) design tasks: (i) *design replication* and (ii) *design modification*. In *design replication*, the model completes a design based on a reference image and a build instruction, which provides the task context and the overall design goal. In *design modification*, the model refines an existing design based on a reference image and an edit instruction, which describes the target component and the expected changes in plain language without numeric values. These task settings reflect the designers' practical expectations towards AI assistance, ranging from automating repetitive tasks to assisting with micro-level design edits [24, 20].

The benchmark represents a UI design as a state composed of components (e.g., buttons, text) and their attributes (e.g., position, width, color), mirroring how design software organizes design components. In each task, a UI design is formalized as a state s made up of n component–attribute pairs.

$$s \;=\; \{\,(c_i,\, a_{ij},\, v) \mid i = 1, \ldots, n,\; j = 1, \ldots, m_i\,\}.$$

where each component c_i is associated with a set of attribute-value pairs (a_{ij}, v). The benchmark evaluates the model-generated design $\hat{s}$ by comparing it with a ground-truth design s_{GT}, using these attribute values and visual characteristics.

3.1.1 Design Replication

The replication task measures a VLM's ability to map out a sequence of tool invocations to reconstruct a target UI design s_{GT}. Starting from an empty canvas $s_0 = \varnothing$, the model iteratively adds or edits components, producing a design trajectory

$$s_0 \;\rightarrow\; s_1 \;\rightarrow\; \ldots \;\rightarrow\; s_t.$$

The process stops when the model determines $s_t \approx s_{\mathrm{GT}}$ and ends tool invocations.

3.1.2 Design Modification

The design modification task (Fig. 3.1) measures the VLM's ability to apply a fine-grained tool invocation over turns to complete specified changes. Starting from a pre-existing design state s_{old}, the model is instructed to modify the design into a specified target state s_{GT} based on (i) a task instruction and (ii) a ground truth image. The model needs to conduct a list of changes $\Delta = [\ldots]$ to reach the target design state

$$s_{old} \xrightarrow{\Delta} s_{GT}$$

The design modification task consists of three sub-tasks:

- *Attribute Update* (Fig. 3.1B-(A)): Modify attributes a in different components c to target values v' in a design. The attributes include fill color, size, text content, corner radius, and position.

$$\Delta_{\mathrm{attr}} = \{\,(c_k,\, a_k,\, v'_k) \mid k = 1, \ldots, K\,\}.$$

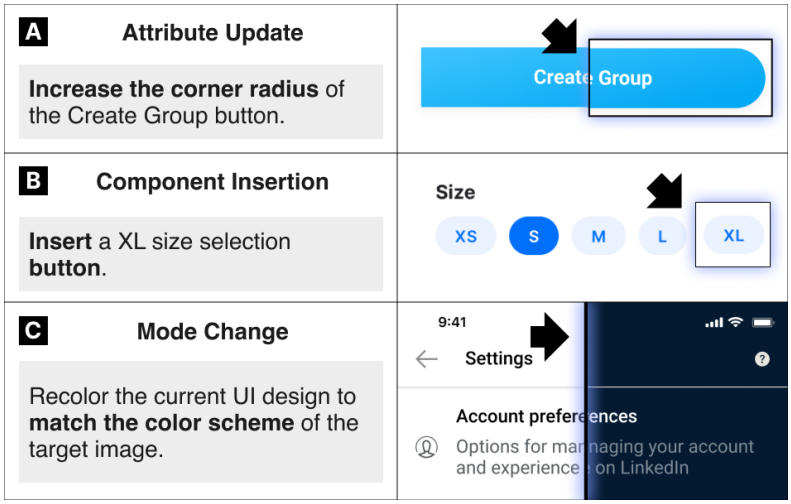


Figure 3.1: **Design Modification Tasks.** Design modification task measures a model's capacity to perform targeted edits, requiring it to (A) adjust a component's attributes, (B) insert a new component, or (C) switch the overall color scheme.

- *Component Insertion* (Fig. 3.1B-(B)): Create and insert a list of new component-attribute pairs into a design.

$$\Delta_{\text{add}} = \{\, (c_k,\, a_k,\, v_k) \mid k = 1, \ldots, K \,\}.$$

- *Mode Change* (Fig. 3.1B-(C)): Change a list of colors to a target value to convert the design between light and dark themes.

$$\Delta_{\text{col}} = \{\, (\text{rgb}_k^{\text{old}},\, \text{rgb}_k^{\text{new}}) \mid k = 1, \ldots, K \,\}.$$

These operations cover the primitive modification patterns in UI design workflows: property adjustments, component additions, and systematic style changes.

3.2 Dataset Creation

We collected UI designs from an online community and refined them through a designer review.

3.2.1 Data Source

Our dataset comprises UI designs from public projects on the Figma Community platform [10], a widely used online repository. We selected Figma as a data source for its popularity and standardized format [1].

3.2.2 Data Collection

We created the dataset through selection, categorization, sampling, and annotation to ensure UI type diversity (see Appendix for details).

- (i) *Data Selection.* We manually selected projects with editable UI designs, excluding non-UI resources, e.g., prototyping kits. Selection stopped when additional designs showed minimal variation. Each design was exported as SVG, PNG, and JSON via Figma's REST API.

[1] All designs are licensed under Creative Commons (CC BY 4.0), and the dataset includes a link to the original project.

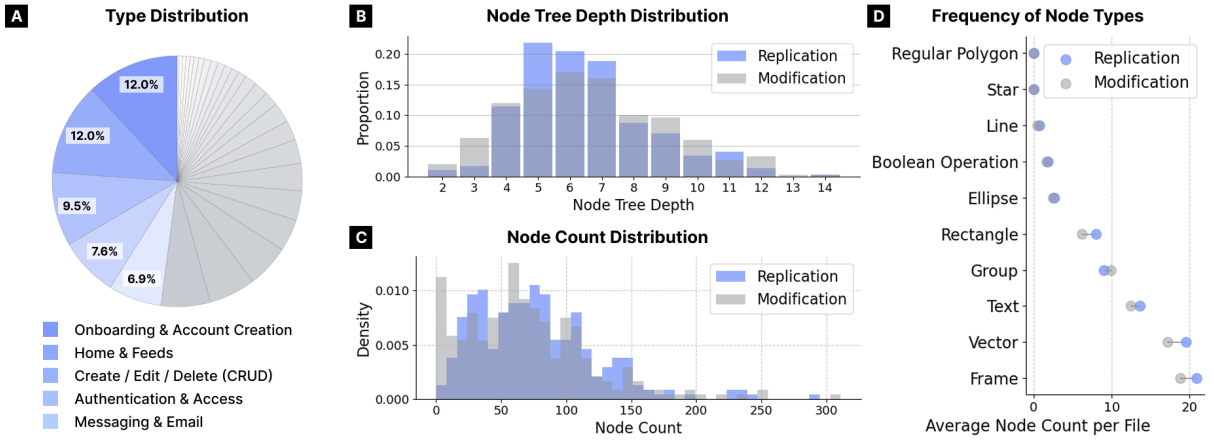


Figure 3.2: **Benchmark Data Statistics**: (A) Frequency of the five most frequent UI design types, with other categories grouped (see Appendix for full distribution). (B) The distribution of node tree depth is similar across the replication and modification sets with a Gaussian-like pattern. (C) The node count distribution is also similar across both sets. (D) The skewed frequency of node types per design indicates common patterns in component usage.

- (ii) *Data Sampling.* The designs were categorized into 30 UI types by GPT-4.1-Mini [29] with zero-shot prompting; one of the authors defined the types, adapting Mobbin's categorization with GPT-4.5 [25, 30]. The replication set ($n = 298$) was sampled using stratified sampling from the original pool ($n = 3,327$), covering all UI categories (10 per category, except one with 8). The modification set ($n = 300$) was manually sampled from the same pool based on task-relevant attributes (e.g., round borders).
- (iii) *Data Annotation.* For attribute update and component insertion tasks, we generated instructions using GPT-4.1-Mini based on visual differences between manually created "base" and "target" states. These states were manually created by editing the original design in Figma (e.g., deleting a component).

3.2.3 Data Refinement

To improve dataset quality, we reviewed and refined the dataset through designer review of designs and annotations. (see Appendix for details).

- (i) *Data Analysis.* We cleaned the designs through automated pre-processing (e.g., placeholder replacement, font standardization) and defined common issues, including occlusions, illustrations, and visual clutter (misleading layout composition).
- (ii) *Manual Revision.* Four experienced designers revised 598 designs by fixing issues, correcting flawed instructions from GPT-4.1-Mini, and annotating required tools. This process resolved occlusions in 33.4% of the designs, illustration errors in 15.5%, and clutter in 29.0%.

3.3 Data Statistics

3.3.1 Overview

The benchmark contains 598 design tasks collected from 3,327 mobile UI designs across 30 categories collected through stratified sampling (Fig.3.2A). Each design includes an image and a JSON file representing Figma's node structure, where components (e.g., vector, text, rectangle) are organized in hierarchical trees with attributes such as position, size, and color (see Appendix for an example). Vector, text, and rectangle nodes are most common, while group and frame nodes also appear frequently (Fig.3.2D). Through the data refinement process, we remove outliers with an excessive number of nodes, reducing variance within the dataset.

3.3.2 Structural Characteristics

We analyzed the structural properties of our benchmark tasks to ensure they offer an appropriate level of complexity for evaluating tool-based design capabilities. First, the node tree depth is slightly shallower in both replication ($M = 6.48,\ SD = 2.04$) and modification ($M = 6.62,\ SD = 2.40$) tasks than in the source dataset ($M = 7.06,\ SD = 2.32$) (Fig. 3.2B); the node count is also lower (replication: $M = 76.30$, modification: $M = 69.11$, source: $M = 126.25$; Fig. 3.2C), achieving a more efficient representation of the design while preserving core structure. Despite this reduction, the normalized *Shannon entropy* of the node types remains comparable, with $J = 0.763$ ($K = 10$) for replication, $J = 0.760$ ($K = 10$) for modification, and $J = 0.726$ ($K = 13$) for source dataset, retaining component type diversity in the design. The distribution of component types remains consistent across tasks, with nodes such as `frame`, `text`, and `vector` most frequent (Fig. 3.2D). These patterns suggest that our benchmark tasks simplify structural complexity while maintaining component-level information, enabling consistent evaluation of model performance.

3.4 Evaluation Metrics

Our evaluation framework measures perceptual and semantic alignment between generated and reference designs based on human visual processing. The framework decomposes visual similarity into three hierarchical levels that approximate the bottom-up stages of human visual perception: *features* (low-level properties such as edges and colors), *patterns* (mid-level compositions such as shapes and groupings), and *semantic objects* (high-level interpretations such as buttons or input fields) [41]. This hierarchy reflects how humans sequentially extract meaning from a design, progressively assembling basic visual features into recognizable patterns and eventually inferring functional design elements. Additionally, to capture model performance on component-level attribute editing (position, color, text), we include component-wise similarity, which evaluates how accurately individual UI elements are reproduced (see Appendix for implementation).

Structural Similarity Index Measure (*Feature-level*): SSIM compares local image statistics (mean, variance, and covariance) to assess how well low-level features such as size and shape are preserved [40].

Saliency Similarity (*Pattern-level*): Saliency similarity measures predicted human attention patterns within a design. We compute histogram intersection between normalized saliency maps of the ground

Table 3.1: **Benchmark scores on design replication and modification task.** Each score indicates the average similarity scores between completed designs and the ground truth. In replication, GPT-4.1 and Gemini-2.5-Pro exhibit leading performance in the replication task, while GPT-4.1 consistently exhibits robust performance in the modification task.

Replication

Model	SSIM	Saliency	BLIP	Comp. Wise
GPT-4o	0.739 (±0.172)	0.478 (±0.136)	0.495 (±0.250)	0.671 (±0.087)
GPT-4.1	0.767 (±0.129)	0.612 (±0.137)	**0.655 (±0.251)**	**0.716 (±0.075)**
Claude-3.5-Sonnet	0.725 (±0.180)	0.483 (±0.151)	0.518 (±0.272)	0.666 (±0.089)
Gemini-2.5-Flash	0.736 (±0.184)	0.619 (±0.149)	0.571 (±0.270)	0.702 (±0.100)
Gemini-2.5-Pro	**0.774 (±0.117)**	**0.630 (±0.162)**	0.620 (±0.273)	0.694 (±0.094)

Modification

Model	SSIM	Saliency	BLIP	Comp. Wise
GPT-4o	0.843 (Δ0.136)	0.845 (−Δ0.009)	0.740 (−Δ0.021)	0.943 (Δ0.015)
GPT-4.1	**0.890** (Δ0.183)	**0.861** (Δ0.007)	**0.806** (Δ0.044)	**0.951** (Δ0.024)
Claude-3.5-Sonnet	0.816 (Δ0.109)	0.858 (Δ0.004)	0.775 (Δ0.013)	0.946 (Δ0.018)
Gemini-2.5-Flash	0.874 (Δ0.167)	0.857 (Δ0.003)	0.784 (Δ0.023)	0.948 (Δ0.020)
Gemini-2.5-Pro	0.867 (Δ0.159)	0.851 (−Δ0.003)	0.804 (Δ0.043)	0.935 (Δ0.007)

± indicates the standard deviation. Δ indicates the average score increase from the base design. Shaded cells mark the best score in each column.

truth and generated design using UEyes [15], trained on eye-tracking data from UI screens.

BLIP Caption Similarity (*Object-level*): We generate captions for both ground-truth and generated designs using BLIP-2 and compute cosine similarity between their embeddings using SentenceTransformer [32], approximating semantic-level similarity.

3.4.1 Component-wise Similarity

Following Design2Code [34] and related work [22], we perform one-to-one component matching using the Hungarian algorithm (IoU for visual elements, text-position similarity for text), then compute similarity across four attributes: component match rate, position (Euclidean distance), text content (F1 score), and fill color (RGB distance).

3.4.2 Metric Aggregation

For both tasks, we evaluate the similarity between the generated design s_t and the reference design s_{GT}, reporting the final score as the average similarity values across all benchmark cases. In the modification task, we measure how much the model's edits improve similarity to the ground truth s_{GT} by comparing scores before and after editing.

Chapter 4. Experiments

4.1 Configuration

4.1.1 Pipeline

The benchmark implements a tool-based UI design generation pipeline based on the Model Context Protocol (MCP)[1]. Models have access to 50 predefined tools for design operations, including creation, deletion, layout adjustment, styling, and content modification, which are relayed to Figma via MCP (See Appendix for details). Models initiate each task by receiving the target screen UI image and task instructions, employing predefined tools; for modification tasks specifically, the pipeline initializes existing designs on the Figma canvas from a structured JSON representation. In each task, models operate within an agentic loop based on the ReAct framework [43], which involves cycles of thought, action (tool invocation), and observation.

4.1.2 Model Setup

We evaluate five state-of-the-art vision-language models (VLMs) with visual understanding and tool-use capabilities, including GPT-4o [28], GPT-4.1 [31], Claude-3.5-Sonnet [1], Gemini-2.5-Flash and Pro [6]. All models are instructed using standardized instructions with the temperature level at 0 for reproducibility (See Appendix for the instructions). We excluded open-source models due to their high failure rates in multi-turn tool invocation. These models commonly terminated after one or two turns, resulting in blank or incomplete outputs.

4.2 Main Results

Table 3.1 shows average similarity scores between completed designs and ground truth across the replication and modification tasks. The score with ($\pm$) represents the standard deviation in the replication task, while the score with (Δ) indicates score changes due to model actions from the initial state s_{old} to the target state s_{GT} in the modification task.

4.2.1 Replication

As shown in Table 3.1, Gemini-2.5-Pro scored highest in SSIM and saliency, indicating strength in feature-to-pattern-level similarity (contours and shape compositions). GPT-4.1 achieved the best scores for BLIP and component-wise similarity, indicating strength in replicating design semantics and preserving component attributes (position, color, text)

4.2.2 Modification

GPT-4.1 achieved the highest scores across all four metrics. Importantly, several models demonstrate small or negative Δ values, indicating score stagnation or degradation due to model actions. We explore these results in the analysis section.

[1] Architecture building upon https://github.com/grab/cursor-talk-to-figma-mcp, MIT License

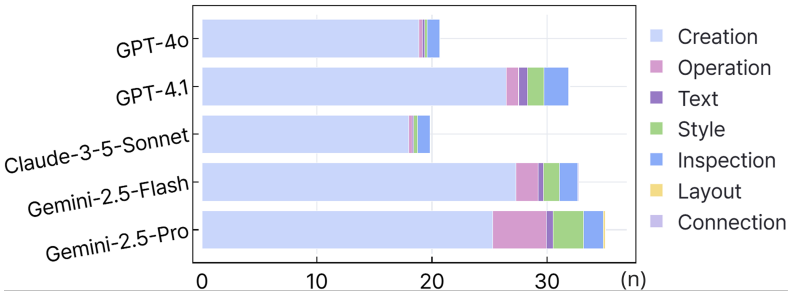


Figure 4.1: **Tool Invocation Frequency**: Average tool invocations per task *x axis* (n). Colored blocks show tool types (e.g., creation includes `create_rectangle`). Higher-performing models exhibit greater tool diversity.

4.3 Analysis

To understand performance differences across models, we analyzed tool invocation histories at each turn. The histories directly capture model actions and their influence on overall scores. Our analysis reveals two primary observations: (i) models achieving higher scores in the replication task demonstrate diverse tool invocations over increased turns; and (ii) models performing well in the modification task exhibit precise, targeted tool invocations within fewer turns.

4.3.1 Replication task demands diverse invocation of tools.

We identify that in the replication task, high-scoring models exhibit more diverse tool patterns. Figure 4.1 shows average tool invocation frequency and tool type diversity by model on each task. The higher-performing models, namely GPT-4.1 and Gemini-2.5-Pro, display increased tool diversity, indicating more strategic behaviors. For example, we observe a modular approach, where a model creates a component (e.g., a buy button) once and propagates it across the design using the `copy_node` action, saving turns. In contrast, lower-performing models, such as GPT-4o and Claude-3.5-Sonnet, complete the design by simply creating components in order. These observations suggest that intrinsic rewards encouraging exploration could help models learn strategic, diverse tool patterns.

4.3.2 Modification task requires precise selection of tools.

We posit that the modification task poses a unique challenge for models due to its high granularity. It requires precise tool invocations, where slight inaccuracies lead to non-uniform impacts across similarity metrics. For instance, adding a line break may minimally affect textual content but can cause a large shift in a saliency map. This phenomenon is reflected in the small or negative Δ values in Table 3.1 (Modification); due to non-uniform shifts, the score changes converge towards zero on average.

For a more in-depth understanding, we introduce Pos@K, a metric counting the number of cases where the number (K) of similarity scores increases after an edit. For instance, A Pos@3 indicates the number of cases where the model edit introduced a positive increase in *three* metrics and a decrease in the remaining metrics. Table 4.1 shows the distribution of Pos@K values for each model in the modification Task 1, *attribute update*, in ratio. Notably, in the table, models that underperformed elsewhere — Claude-3.5-Sonnet and Gemini-2.5-Flash — yielded a higher proportion of Pos@4 than other models.

To investigate this performance inversion, we compared tools invoked by models against human-

Table 4.1: Distribution of Pos@K cases in ratio for each model in Task 1: Attribute Update.

Model	Pos@4	Pos@3	Pos@2	Pos@1	Pos@0	F
GPT-4o	24.0%	**41.0%**	18.0%	13.0%	2.0%	2.0%
GPT-4.1	25.0%	31.0%	**23.0%**	14.0%	6.0%	1.0%
Claude-3.5-Sonnet	30.0%	38.0%	17.0%	10.0%	4.0%	1.0%
Gemini-2.5-Flash	**31.0%**	37.0%	20.0%	9.0%	3.0%	0.0%
Gemini-2.5-Pro	27.0%	36.0%	12.0%	**16.0%**	**8.0%**	1.0%

Shaded cells mark the highest value in each column. F: the ratio of failed cases.

Table 4.2: Tool invocation statistics per model.

Model	Tool Precision	Tool Recall	Tool Diversity
GPT-4o	0.4195	0.9967	4.3100
GPT-4.1	0.5434	1.0000	5.4433
Claude-3.5-Sonnet	0.5677	0.9900	3.6300
Gemini-2.5-Flash	0.5943	0.9767	3.4067
Gemini-2.5-Pro	0.5659	0.9633	4.1400

Tool Precision and Recall: normalized intersection with human-annotated tools. Tool Diversity: average unique tools per case.

annotated tools for each task. Three designers annotated the required tools for each of the 300 modification cases, serving as reference annotations (see Appendix for details). As shown in Table 4.2, Gemini-2.5-Pro and GPT-4.1 exhibited lower tool precision but higher diversity compared to Claude-3.5-Sonnet and Gemini-2.5-Flash. This suggests that excessive tool diversity impaired performance in this task. The finding generalizes across all modification tasks: treating Pos@K as an ordinal score, we found a positive correlation between tool precision and the Pos@K score ($\rho = 0.149, p < 0.01$) and a negative correlation between tool diversity and Pos@K ($\rho = -0.365, p < 0.01$). These results indicate that precise tool selection is critical for the modification task. To improve precision, methods that teach accurate skills, such as imitation learning, are necessary.

4.3.3 Our metrics align with design experts.

A study with human annotators reveals that our metrics closely align with human preference. Table 4.3 demonstrates that three metrics, saliency, BLIP, and component-wise similarity, are statistically significant predictors of pairwise human judgments on design similarity.

Following the methodology of Design2Code [34], we collected pairwise human preference data from design experts on Prolific. We used 100 design cases, chosen from our replication results via stratified sampling (weighted by UI type and complexity). Each pair received "win," "lose," or "tie" labels from three annotators, with final outcomes determined by majority vote, excluding cases without a majority. Subsequently, we trained a logistic regression model to predict the binary preference (win=1, lose=0) using the difference in our metric scores between two designs in a pair as the independent variable. On a 50/50 training/test split of the 363 non-tie samples, the model achieved 75% prediction accuracy.

Table 4.3: Logistic regression on human pairwise preferences using similarity metrics as features.

	coef	std err	z	p
SSIM	-0.0500	0.2241	-0.223	0.82336
Saliency	0.9808	0.2758	3.557	0.00038
BLIP	0.7323	0.2357	3.106	0.00189
Comp. Wise	0.5836	0.2806	2.080	0.03506

4.4 Ablation Study

We conducted an ablation study to isolate the effects of multi-turn iteration and tool-based interaction. (i) The baseline setting is tool-based multi-turn, which is identical to the setting in the benchmark. To control the effect of the turns, we include the (ii) tool-based single-turn condition, where all tool invocations are generated at once without intermediate feedback. This enables a direct comparison against our third condition, (iii) a code-based single-turn adapted from prior work [34], thereby isolating the impact of using tools.

Table 4.4: Ablation study results for each condition.

Method	**Model**	**Replication**		
		SSIM	**Saliency**	**BLIP**
Code	GPT-4.1	0.773 (±0.110)	0.649 (±0.129)	**0.689 (±0.231)**
(Single-Turn)	Gemini-2.5-pro	0.771 (±0.108)	0.696 (±0.120)	0.685 (±0.233)
Tool	GPT-4.1	0.704 (±0.219)	0.510 (±0.156)	0.534 (±0.281)
(Single-Turn)	Gemini-2.5-pro	**0.775 (±0.118)**	**0.699 (±0.121)**	0.660 (±0.259)
Tool	GPT-4.1	0.767 (±0.129)	0.612 (±0.137)	0.655 (±0.251)
(Agent)	Gemini-2.5-pro	0.774 (±0.117)	0.630 (±0.162)	0.620 (±0.273)

Shaded cells mark the *best overall* in each column.

Table 4.4 shows that Gemini-2.5-Pro achieves its highest scores in the single-turn tool-based setting, while GPT-4.1 performs better under the multi-turn condition. The single-turn and multi-turn settings induce distinct model behaviors: the multi-turn setting encourages strategic, non-linear operations (e.g., layer reordering), while the single-turn setting favors more uniform, linear action sequences. This divergence explains the performance results in Table 4.4: the riskier strategic operations (e.g., copying incorrectly designed buttons) can degrade output quality, leading to lower scores in some design cases. Nevertheless, we propose that this multi-turn, agentic configuration holds important potential for complex, collaborative design tasks.

4.5 Error Cases

We conducted an error analysis to identify common failure cases of current models and their underlying causes.

4.5.1 Geometric Operation Errors

Models frequently fail to control the geometric properties of UI elements, leading to errors in count, directionality, and spatial arrangement. For instance, in the map user interface shown in Fig. 4.2, the model produces (A) an incorrect number of location markers, (B) paths with incoherent directions, and (C) a spatially inconsistent arrangement of components.

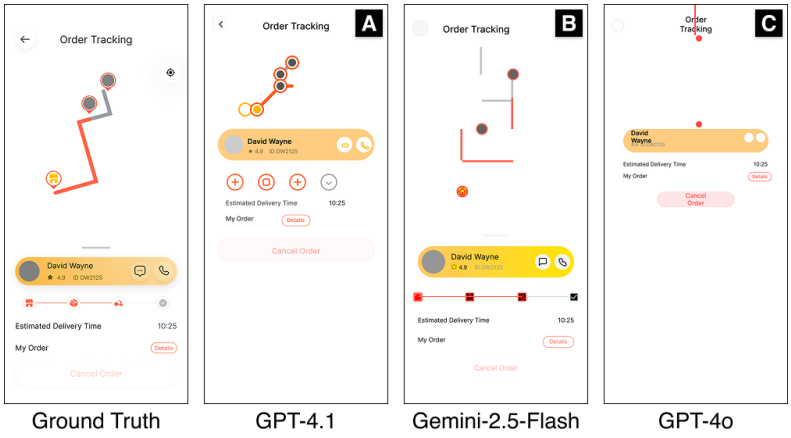


Figure 4.2: **Error case 1**. The models (A) miscount the markers, (B) draw irregular lines, and (C) create an inconsistent layout.

4.5.2 Layout Operation Errors

Models reveal erroneous reasoning about screen auto-layout. In Figma, this feature automatically realigns and resizes child elements when their parent resizes; similar mechanisms appear in other editors [35]. Because any modification to the layout parameters can propagate through an entire hierarchy, the task demands prediction of changes without dictating their values through commands. As shown in Fig. 4.3, the models' adjustment to auto-layout disintegrates the selector button and its placeholder (A) and forces the placeholder below the viewport (B).

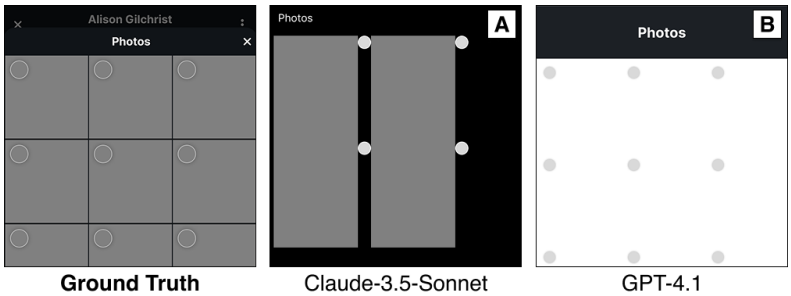


Figure 4.3: **Error case 2**. Updates to auto-layout settings trigger (A) disintegration between selector and placeholder or (B) push the placeholder downwards beyond the screen.

4.5.3 Text Operation Errors

Models commonly fail to infer appropriate dimensions from text attributes (e.g., font, size, and spacing), leading to narrowly sized text components and subsequent text overflow. Figure 4.4 illustrates two typical failures: (A) the price label extends beyond its bounding box, breaking visual alignment, and (B) the button label overruns its frame, blending into the background.

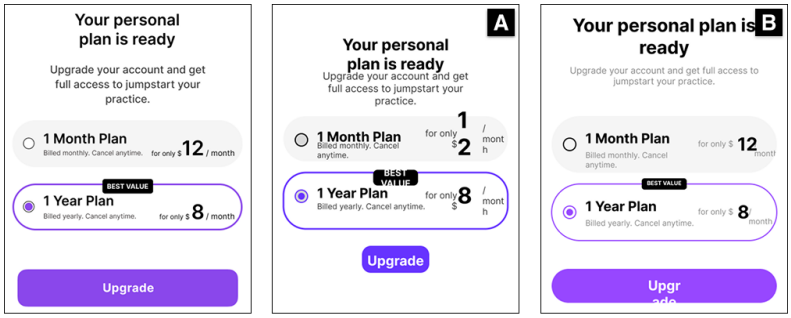


Figure 4.4: **Error case 3**. Models (A, B) fail to assign sufficient span to the text component and trigger text overflow.

Chapter 5. Conclusion

We introduce a benchmark designed to evaluate vision-language models (VLMs) on tool-based UI design tasks. Our findings suggest that current state-of-the-art models exhibit promising capabilities in replicating and modifying interface designs through tool invocation with design software. Particularly, high-performing models exhibit more diverse tool use and strategic behaviors. Nonetheless, adverse tool selection and design errors highlight the limitations in the implementation framework and the model capacity. By presenting an initial evaluation of VLMs performing design tasks within conventional software, this benchmark offers valuable insights toward achieving human-aligned design automation with VLMs.

Chapter A. Appendix

A.1 Data Collection Procedure

To create a consistent dataset representative of the current design styles, we selected the Figma Community as our data source. Given that Figma does not offer an officially documented API on the Figma Community, we devised a data collection procedure to collect and sample data relevant to the evaluation. Additionally, we implemented a task instruction generation step to provide instruction prompts for a model in the two design modification tasks: *attribute update* and *component insertion*, guiding the model to attain a narrowly scoped task goal.

A.1.1 Data Selection

We chose a manual selection process to identify Figma projects adherent to the criteria: each project was required to have at least two high-fidelity, editable UI designs. Each project is accessed through the "Design resources" tab under the category "Mobile app design templates." Resources unrelated to UI design, such as prototyping kits, static images, device mockups, or design systems, were excluded. The design selection terminated when further designs offered smaller variations relative to previously collected samples. Finally, the selected UI screens were exported through the Figma REST API, producing a set of SVG, PNG, and JSON data for each design.

A.1.2 Data Sampling

To analyze diversity within the dataset, we first categorized the dataset into 30 distinct UI types based on visual characteristics. The initial UI design categories were adapted from Mobbin's web UI classifications. Due to the extensive nature of these categories, one of the authors used GPT-4.5 to generate initial clusters and manually refined the categories to minimize overlap and increase clarity. Finally, we used GPT-4.1-Mini with zero-shot prompting to automatically assign these categories to design images based on the predefined prompt.

The benchmark addresses two primary tasks: UI design replication and modification, each requiring task-specific data selection methodologies. Modification tasks specifically demand manual selection based on clearly defined criteria. For example, tasks involving border radius require UI elements with rounded borders.

- *Design Replication*: We conducted stratified sampling based on the function-based UI type (e.g., login screen) to select 298 designs from the total of 3,327 categorized designs, ensuring balanced representation of different UI designs. This resulted in 10 examples per category, except for one category, limited to 8 examples due to insufficient data.
- *Design Modification*: Authors manually selected 100 designs per modification type (attribute modification, component insertion, mode change) from the dataset. Algorithmic selection was inadequate, as it could not reliably ensure the presence of necessary UI features (e.g., matching dark and light mode designs). Authors manually generated target states of the modification by directly editing selected designs in Figma, such as removing a button.

A.1.3 Data Annotation

Each modification task includes a ground-truth design paired with a case-specific instruction. These instructions are essential for precisely guiding the model to select and modify specific design elements (e.g., adjusting the corner radius of an individual list element). Without explicit instructions, the model must infer the necessary edits by visually comparing the original and target states, introducing a significant perceptual bottleneck due to fine-grained visual differences. Specifically, instructions for *attribute update* and *component insertion* tasks are generated by comparing differences between the "base" and "target" states. GPT-4.1-Mini is employed for generating an instruction for each case. In contrast, instructions for the mode-change task are uniformly defined due to its general characteristic of applying color changes across multiple components.

A.1.4 Model Collection Prompts

Figure A.1: **Modification Task-1 Instruction Generation Prompt:** A prompt for generating task instructions for the modification Task-1 (Attribute Update). The keyword alternates based on the target components.

```
** Instruction **
Write a human-like prompt instructing the model to transform the UI in the first image into the UI
shown in the second image.
Specifically, focus on changing only the {colors|sizes|text contents|corner radius|positions} of UI
elements from the first image to match those in the second image.
Then, explicitly identify and describe the required {color|size|text content|corner radius|position}
changes for each UI element.
Start by requesting the modification on the image: "Make changes to the {colors|sizes|text contents|
corner radius|positions} of the following UI elements:".
Then outline the changes in a structured format strictly using bullet points.

** Rules **
- Avoid specifying exact color codes, font names, dimensions, or other numerical values.
- You don't need to mention the image, just say "Make changes to the colors of the following UI
elements:".
- You don't mention elements that are not changed.
```

A.2 Data Refinement Procedure

To improve the reliability of our dataset, we performed systematic refinement steps. The procedure aims to align the visual appearance with the underlying structure, ensuring that the design is reproducible via tool invocation, solely based on the image. Initially, we identified significant noise in the dataset, manifesting as inconsistent patterns in the structure of common UI elements (e.g., a large gradient background hidden under button elements that extends over the parent frame). This inconsistency poses a critical challenge for evaluation, as the structure of the ground truth design exhibits an arbitrary pattern. Therefore, our refinement procedure focuses on matching the visual appearance to an underlying structure and ensuring the design is reproducible solely via tool invocation. We expect the outlined refinement procedure to help future researchers plan similar data refinement procedure using design platforms and software.

Figure A.2: **Modification Task-2 Instruction Generation Prompt:** A prompt for generating a task instruction for the modification Task-2 (Component Insertion).

```
** Instruction **
Write a human-like prompt instructing the model to add a component to the UI in the first image to
make the UI shown in the second image.
Specifically, you are provided with a third image that illustrates the component to be added.
Comprehensively describe the component to be added using the third image.
If the third image contains text, include the text in the description.
Then suggest where to place the component in the first image to match the second image.
Start by requesting the insertion of the component. Start with "Insert the following component into
the UI:"."
Then outline the component in a structured format, strictly using bullet points.
Next, describe the placement of the component in the first image. Start with "Place the component in
the following location:"."
Then describe the location in a structured format, strictly using bullet points.

** Rules **
- Avoid specifying exact color codes, font names, dimensions, or other numerical values.
- Explicity mention any text in the component. Make sure you do not miss any text.
- You should not mention the image in the instruction. Just elaborate on how to add the component.
- You should not mention elements that are not changed.
```

A.2.1 Data Analysis

With the initial review of the dataset, we conducted automated data cleansing procedures. Specifically, (i) all original images, such as profile photos, were replaced with grey placeholders due to limitations in reproducing these images within the design software. (ii) Locked design elements were unlocked to enable manipulation. (iii) All component instances in Figma, which are copies of reusable components synchronized with their originals, were detached to allow independent editing. (iv) Hidden elements, which are visually irrelevant, were removed entirely. (v) Fonts were standardized uniformly to the "Inter" typeface while preserving original stylistic attributes, including weight and style. This was due to accessibility constraints associated with certain font families in different environments.

After automated cleansing, we performed a detailed analysis of the remaining issues that affect the model's capability to generate designs from input images successfully. Our evaluation identified three primary error categories that degrade data quality: (i) component occlusions, (ii) complex illustrations, and (iii) visual clutter. Component occlusions describe cases where components are obscured by overlaying objects or positioned beyond the visible boundaries, such as modal dialogues with hidden background layers. Complex illustrations indicate highly detailed visuals that require a complex layering of multiple vectors, which falls outside the current user interface generation scope. Visual clutter refers to irregular layouts in which groups or components are slightly offset by small amounts (e.g., 2 pixels) while looking aligned. Additionally, our review revealed inaccuracies within the task modification instructions generated by GPT-4.1-mini.

A.2.2 Manual Revision

To enhance dataset reliability, we performed a manual review of each design involving four participants who (i) are currently majoring in a design or related field and (ii) have experience with the Figma tool. Participants were recruited through snowball sampling and compensated approximately USD 11 per hour, contributing a total of 7 hours each to the review process. Participants systematically checked for the three predefined issue categories and made necessary corrections. In modification tasks

Figure A.3: Data categorization prompt for User Interface (UI) type classification

```
###  SYSTEM  ###
You are an expert product-designer-turned-taxonomist.
Your task is to inspect one UI screenshot and decide which single category below best describes the *
purpose of the entire screen* (not individual widgets).

Available categories
(only one may be returned):

1. **Onboarding & Account Creation** -- Account Setup, Splash Screen, Welcome/Get-Started, Sign-up,
Walk-through
2. **Authentication & Access** -- Login, Forgot-Password, Verification, Delete/Deactivate Account

(...categories 3-29 omitted for brevity)

30. **Others** -- Any screen that clearly doesn't fit categories 1-29

###  INSTRUCTIONS  ###
1. Look at the screenshot holistically: What is the user trying to do here?
2. Pick the single *most appropriate* category (or **30** if none fit).
```

specifically, they validated the task instruction prompts and annotated the corresponding required tools. Additionally, each design was re-reviewed by the author, who applied further corrections when necessary. Out of 598 design tasks, our revision addressed component occlusions (200 cases, 33.4%), complex illustrations (93 cases, 15.5%), and visual clutter (174 cases, 29.0%).

Data Revision Instruction

During the revision, we asked the four participants to conduct the manual revision. Figure A.4 shows the revision instruction that participants followed during the manual revision process.

Design Issue Examples

We identify three major issues regarding the collected Figma design examples: component occlusions (Fig. A.5), complex illustrations (Fig. A.6), and visual clutter (Fig. A.7).

A.3 Data Characteristics

A.3.1 Dataset Overview

This appendix provides a detailed overview of the dataset compiled for our experiments. The dataset was collected from the Figma community, and each sample is composed of metadata, a JSON structure, a PNG image, and an SVG image. The metadata, summarized in the Table A.1 below, includes details such as the source URL, license, design type, and structural properties. All designs are distributed under the CC BY 4.0 license.

The data is organized into three distinct subsets for different evaluation tasks: (i) the Source Dataset, (ii) the Replication Task Set, and (iii) the Modification Task Set. The specific characteristics of each are detailed below.

Source Dataset The Source Dataset contains 3,327 original design examples parsed directly from projects in the Figma Community, encompassing 30 different UI design types. Table A.2 shows the

Figure A.4: **Data Revision Instruction** An instruction given to expert designers for revising each task case.

```
1. Remove Invisible Elements from the Frame (label: invisible)

    Removes elements that are hidden behind other objects or are otherwise not visible within the
    Figma frame.

    [Example Image]
    Example 1: The card element hidden beneath the keyboard needs to be removed.

2. Remove Complex Illustrations (label: illustration)

    Removes illustrations used for purely aesthetic purposes (e.g., pictures, drawings) as opposed to
     visual elements that convey information (e.g., icons).

    [Example Image]
    Example 1: The aesthetic illustration located in the center is removed.

3. Remove Element and Layer Clutters (label: clutter)

    Addresses design errors such as elements that are severely misaligned with others or extend
    beyond the boundaries of the Figma frame.

    [Example Image]
    Example 1: A case where individual elements are subtly shifted and not correctly arranged along
    the alignment axis.

    [Example Image]
    Example 2: A case where an element protrudes beyond its parent frame due to incorrect grouping.
```

distribution of the type in the Source Dataset. We observed the gradual decline pattern in the frequency per UI type, with the largest group, "Onboarding & Account Creation," revealing 398 examples and "Pricing & Paywall" showing 8 examples only.

Replication Task Set The Replication Task Set is designed to evaluate the model's ability to reproduce existing designs. It contains 298 cases created by applying stratified sampling to the Source Dataset. We sampled 10 examples for each of the 30 UI types, except for the "Pricing & Paywall category," where all 8 available source examples were used. Each sample in this set includes an additional metadata field, "difficulty," which is classified as either standard or hard.

Modification Task Set The Modification Task Set is designed to assess the model's capacity for targeted design editing. It consists of 300 cases, with 10 examples for each of the 30 UI types. The metadata for each case includes an instruction field that provides a task-specific description of the required change. The data provided per case varies by task type:

- For attribute update and mode change tasks, each case includes a `base_state` and a `target_state`.
- For component insertion tasks, each case includes a `base_state`, the `component` to be inserted, and the `target_state`.

A.3.2 Node Structure

Each design case is represented by a JSON file that encodes its hierarchical node structure. This representation is derived from the Figma REST API, which provides access to the node structure in a

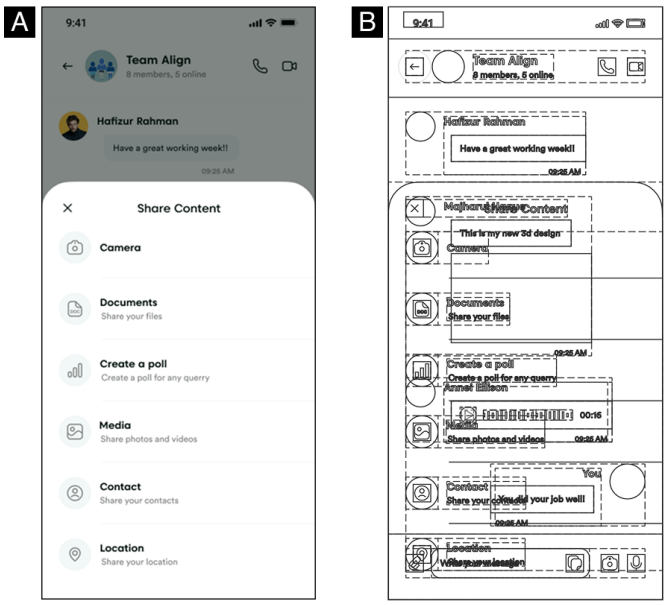


Figure A.5: **Case: Component Occlusion**: (A) A UI design image showing a partial message list above a bottom sheet. (B) The underlying structure of the design reveals the components hidden beneath the bottom sheet.

specific design, with minor modifications to ensure consistency with the parser based on the Figma Plugin API. The JSON mirrors the layered nature of the design process; designers often construct interfaces by stacking, nesting, and grouping components. Consequently, the data follows a tree-like structure where child nodes are recursively accessed through a children list.

Although this hierarchical structure resembles markup languages like HTML or SVG, its semantic purpose is fundamentally different. While markup is optimized for browser rendering, this JSON representation is intended to capture the design's structure as a designer perceives it. This distinction is evident in the prevalence of design-specific properties such as multiple fills and blend modes, which appear more frequently and prominently here than in typical web-oriented markup.

Node as a Complexity Measure

Given that each node corresponds to a design component, the total node count serves as a practical proxy for measuring design complexity. Higher node count indicates greater structural complexity, which primarily equates to greater visual complexity.

For instance, Figure A.8 contrasts two designs from the bottom and top 15th percentiles of node counts in our dataset. Design (A), with a low node count ($n = 25$), features a simple layout with a few primary components, such as icons in a top bar and buttons in a bottom bar. Design (B) has a significantly higher node count ($n = 148$), largely due to numerous fine-grained components, such as the individual keys of a virtual keyboard, which illustrates its greater structural complexity.

A.3.3 Structural Analysis of Datasets

As shown in Table A.3, the Replication Task Set and the Modification Task Set exhibit different patterns compared to the Source Dataset. The Source Dataset is considerably more complex than these task sets. Its mean node count is approximately 65% and 83% greater than that of the replication and

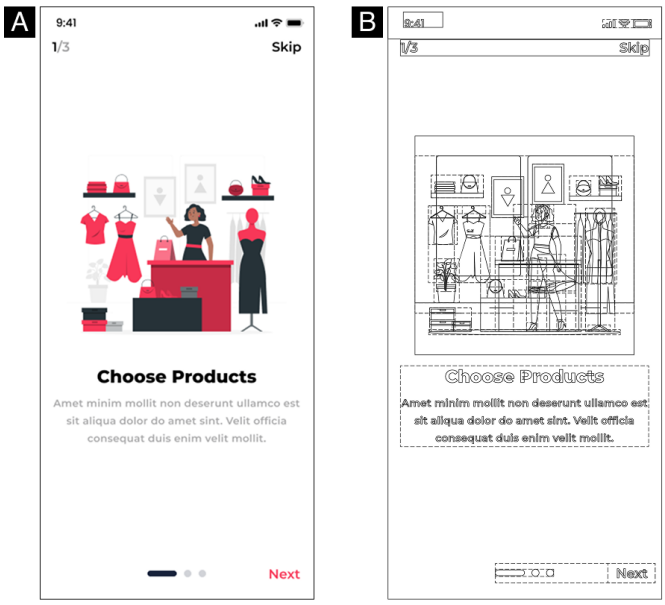


Figure A.6: **Case: Complex Illustration**: (A) A UI design image with complex vector-based illustration at the center of the screen; (B) The underlying structure of the design reveals the complex layers and vector paths.

modification sets, respectively. Furthermore, its designs feature a deeper average hierarchy, with a mean node depth of 7.06±2.32, compared to approximately 6.5 for the other two sets. The numbers reveal a progressive simplification of designs from the originally collected to the more refined examples in the replication and modification test sets.

Furthermore, we analyze the complexity of the designs in our datasets by examining the distributions of node count, depth, and type. This analysis helps to characterize the impact of our data revision process.

Figure A.9 compares the distribution of node counts for the Source Dataset alongside the Replication Task Set before (sampled) and after (revised) our data revision process. Initially, the distribution of the sampled dataset closely mirrors that of the source dataset, confirming the effectiveness of our stratified sampling strategy. However, after the revision, the distribution for the revised data exhibits a noticeable leftward shift. This shift indicates that our revision process successfully reduced the overall complexity of the designs by decreasing the number of nodes per design.

In terms of node depth, the distributions for all three sets (source, sampled, and revised) are largely similar, as shown in Figure A.10. All distributions are centered around a depth of 5 to 7 and resemble a Gaussian distribution. Unlike the node count, the revision process appears to have had a less pronounced effect on node depth. This suggests that the complexity reduction was achieved by simplifying the node structure rather than by reducing their maximum depth.

Figure A.11 details the average count of different node types across the datasets. The source and sampled datasets exhibit similar node type distributions. In contrast, the revised dataset displays a distinct pattern resulting from our revision protocol. Specifically, the counts of `Instance` and `Vector` nodes are significantly reduced. This directly corresponds to our targeted revision goals: to detach instance components for direct manipulation and to simplify complex vector graphics. Concurrently, we observe an increase in `Frame` nodes and a decrease in `Group` nodes, reflecting the detached instances (which become a frame) and the removal of the visual clutter.

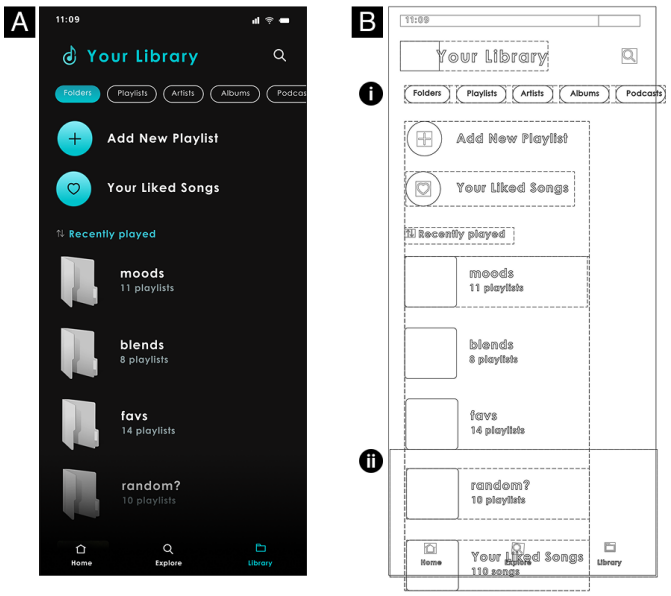


Figure A.7: **Case: Visual Clutters**: (A) A UI design image with visually aligned components; (B) The underlying structure of the design shows (i) an element overflowing its container and (ii) child components that are misaligned with their parent by few pixels.

A.3.4 Node Structure Example

Figure A.12 illustrates the JSON format used to represent the design structure. The data is organized hierarchically, where each node is nested as a child of its parent. Every node contains a unique ID and a type attribute. Visual characteristics are specified through properties such as `blendMode` and `fills`, which define the node's appearance.

Table A.1: Common Metadata Attributes

Field	Description
url	Link to the original Figma file or prototype.
license	Concise statement of the file's usage license.
node_id	Unique identifier of the parent frame in Figma.
type_id	Numeric code denoting the design category.
type_name	Human-readable name of that design category.
node_depth	Maximum nesting depth in the structure JSON.
node_count	Total number of nodes in the structure JSON.
node_type_count	Per-type node counts in the JSON.

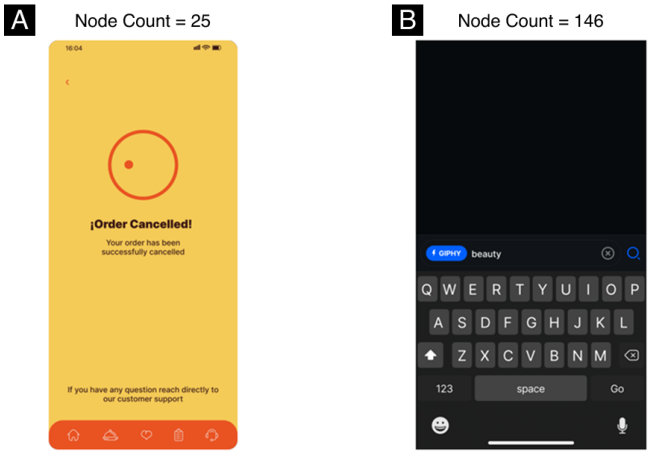


Figure A.8: **Node Count Examples** Two representative designs randomly sampled from the (A) bottom 15% and (B) top 15% of node counts, respectively.

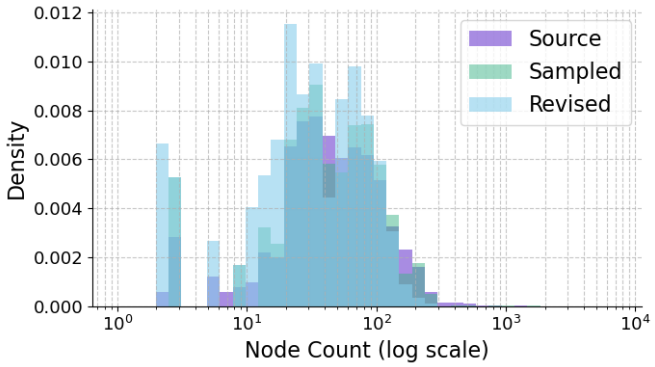


Figure A.9: **Node Count Distribution** A distribution of the node count for the Source Dataset and Replication Task Set (two states: sampled and revised).

Table A.2: UI Categories and Frequency for Source Dataset

Category	**Count**	**Exemplary components**
Onboarding & Account Creation	398	Account setup, splash screen, welcome/get-started, sign-up, walk-through
Home & Feeds	397	Home, news/social feed, browse/discover hub
CRUD (Create–Update–Delete)	316	Add/create, edit form, delete confirmation, draw/annotate
Authentication & Access	254	Login, forgot-password, verification, delete/deactivate account
Messaging & Email	230	Chat thread, inbox, bot conversation
Permissions & Security	211	OS permission request, two-factor verification prompt, privacy policy
Content Detail	201	Article/product/movie/note detail
Profile & Personal Info	183	My account/profile, user/group profile, followers/following
Feedback & Success States	137	Success confirmation, acknowledgement banner, "Done!" dialogs
Cart & Checkout	128	Cart/bag, checkout flow, order confirmation
Search & Explore	117	Search results, suggestions, discover tab
Orders & Billing	103	Order history, billing, wallet/balance, payment method
Settings & Preferences	85	Global settings, dark-mode toggle, language selector
Notifications & Alerts	85	Notification feed, in-app badge list, pull-to-refresh alert
Loading & Progress	62	Spinner, skeleton loader, progress bar
Dashboards & Analytics	60	Dashboard, charts, metric cards
Select & Input	52	Pickers (date, time, list...), set-value dialog
Guided Tours & Walk-throughs	32	Coach marks, feature tutorial, step-by-step guide
Community & Groups	32	Groups, invite teammates, leaderboard, achievements
Promotions & Rewards	32	Coupons, rewards center, gamified rewards
Filter & Sort	30	Filter panel, sort dropdown, advanced filter sheet
Error & Empty States	30	404/error page, empty state, permission denied
Social Engagement	24	Like/up-vote, comments, share sheet, follow/subscribe
Learning & Tasks	20	Quiz/lesson detail, stories, goal/task tracker
Utilities & Productivity	17	Calendar, reminder, timer, schedule manager
Others	17	Components that do not fit the above groups
Media Playback	36	Audio/video player, now-playing screen
Maps & Location	15	Map view, address form, nearby results
Media Capture & Editing	15	Camera/scanner, media editor, recorder, filters
Pricing & Paywall	8	Pricing table, paywall, promotions/rewards upsell

Table A.3: Structural Statistics Across Datasets

Attribute	Original	Replication	Modification
Node Depth	7.06 (2.32)	6.48 (2.04)	6.62 (2.40)
Node Count	126.25 (194.30)	76.30 (46.81)	69.11 (49.40)

Each cell reports the mean and standard deviation (SD).

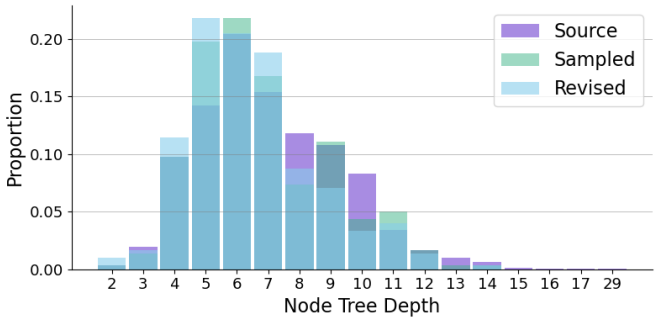


Figure A.10: **Node Depth Distribution** A distribution of the node depth for the Source Dataset and Replication Task Set (two states: sampled and revised).

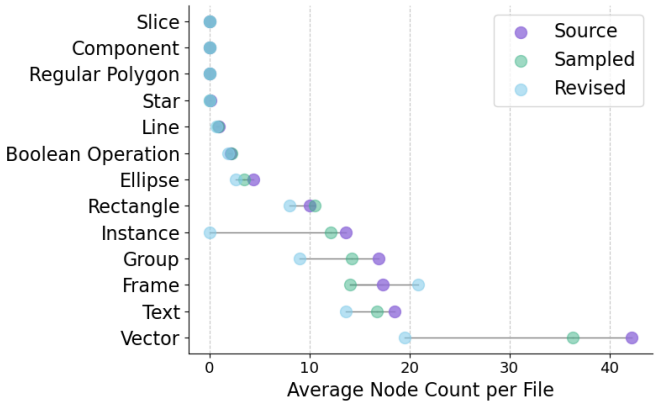


Figure A.11: **Node Type Distribution** A distribution of the node type for the Source Dataset and Replication Task Set (two states: sampled and revised).

Figure A.12: **Design Structure JSON Example** An example formatting for design structure JSON.

```
"children": [
  {"id": "I1:1383;11:211",
    "name": "Image",
    "type": "INSTANCE",
    "children": [
      {"id": "I1:1383;11:211;11:209",
        "name": "Shape",
        "type": "VECTOR",
        "blendMode": "PASS_THROUGH",
        "fills": [
          {"blendMode": "NORMAL",
            "type": "SOLID",
            "color": {
              "r": 0.7041666507720947,
              "g": 0.8579999804496765,
              "b": 1.0,
```

A.4 Benchmark Execution Setup

A.4.1 Experiment Configuration

We implement our experiment in three setups: tool-based multi-turn generation, tool-based single-turn generation, and code-based single-turn generation. Figure A.13 shows the pathways that each configuration produces a UI design image and the structural representation. Our main experimental method, which constitutes the default benchmark configuration, is tool-based multi-turn generation. For our ablation study, we evaluate two additional conditions: tool-based single-turn generation and code-based single-turn generation. This uniform framework ensures that the results from all three methods are directly comparable.

A.4.2 Execution Infrastructure

Pipeline Design

The benchmark adopts the Model-Context-Protocol (MCP) communication structure from an open-source project [1], enabling multi-turn tool invocation in Figma. While inspired by its architecture, we redesigned the pipeline for research use. Instead of using the Cursor IDE as the MCP client, we implemented a custom client for controlled evaluation and agent orchestration. We also extended the tool interface, such as vector operations with explicit parameter control (e.g., shape, position, and color). These enhancements support interpretable, fine-grained manipulation of the environment, which is essential for benchmarking agent behavior in iterative UI design. Our structure enables both execution and structured analysis of reasoning and action quality in tool-driven settings.

Specifically, we implemented the communication between the model and the Figma environment using four key components: a Figma Plugin, a Socket Server, an MCP Server, and an MCP Client that interfaces with the model. A model interaction is triggered by a tool invocation, which follows this sequence:

- The model's tool call is passed from the MCP Client to the MCP Server.
- The command is forwarded through the Socket Server to the Figma Plugin.
- The plugin executes the action on the Figma canvas.
- The action's outcome is returned to the model via the same path.

ReAct Agent Implementation

To enable iterative and multi-step design tasks, we employ an agentic process based on the ReAct framework. Figure A.14 shows the overview of the implementation. The agent initiates an action (a tool invocation) by seeing the task instruction and the ground truth image. Following each action, the model receives an observation (a response) from the Figma environment. It then generates an internal "thought" to reason about the state and produce another "action" in the form of a tool invocation. This observation-thought-action cycle repeats, allowing the model to iteratively complete the design. The process terminates when the model stops invoking a tool invocation, signifying that it views the task as complete.

[1] cursor-talk-to-figma-mcp, MIT license, https://github.com/grab/cursor-talk-to-figma-mcp

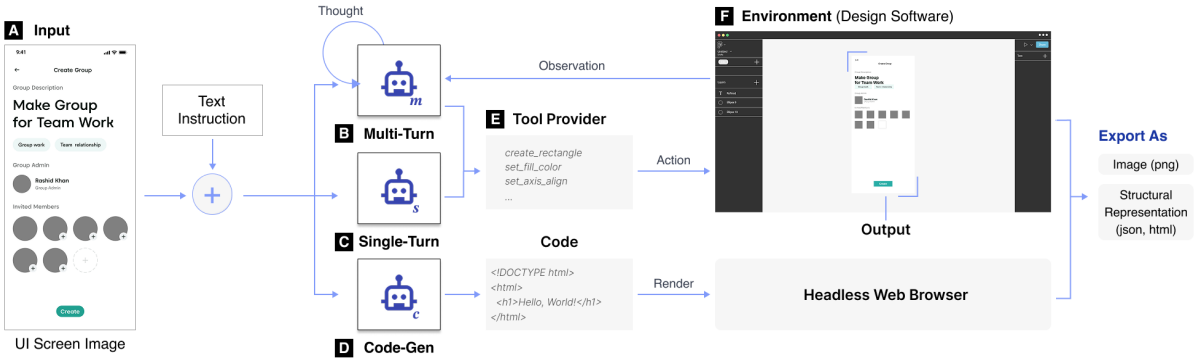


Figure A.13: **Benchmark Experiment Pipeline**: (A) The pipeline feeds the model a target UI image with a textual task instruction. (B) Based on the input, the model generates a design via multi-turn thought–action–observation cycles with tool invocation; (C) single-turn tool invocation; or (D) direct UI code generation. For the code generation, we use Puppeteer to render the code as an image. (E) For the tool-based generation, the pipeline provides a 50-tool list for tool-based configuration. (F) Each tool is linked to a specific operation in Figma.

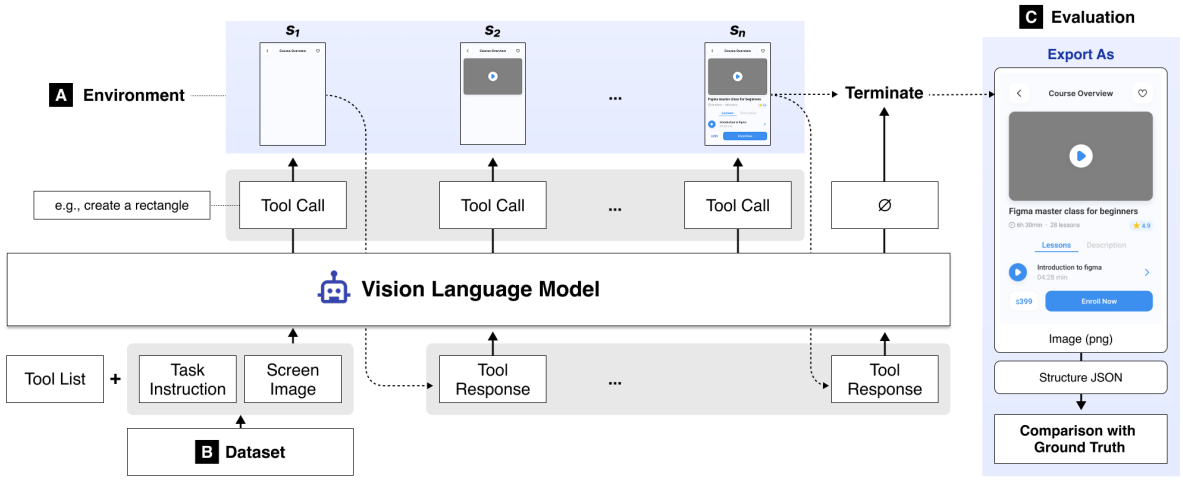


Figure A.14: **Detailed View of the ReAct Agentic Loop**: (A) The model iteratively invokes design tools as an agent. (B) Each task begins with an instruction and reference image. (C) The loop ends when the model stops making tool calls. (D) The final design is evaluated against the ground truth using rendered and structural outputs.

Table A.4: List of 50 Tools Included in the Benchmark

Category & Tool	Description
Connection	
get_channels	Return a communication channel array currently open for environment communication.
select_channel	Select a communication channel.
check_connection_status	Check the environment to verify whether the communication is live.
Creation	
create_rectangle	Create a rectangle with width, height, fill & stroke along with other attributes.
create_frame	Create an auto-layout frame with padding, spacing, resize rules, and style props.
create_text	Create a text node with content string, font family, size, alignment, and color token.
create_graphic	Create vector nodes based on the SVG markup code text.
create_ellipse	Create an ellipse with configurable radii, fill, stroke, and corner smoothing.
create_polygon	Create an N-sided polygon with adjustable corner radius, fill/stroke, and rotation.
create_star	Create a star with custom points and inner-radius.
create_line	Create a straight line between two absolute points with dash pattern and end-caps props.
create_mask	Create a mask node using group of nodes to create a clipping group.
Inspection	
get_page_info	Return list of top-level nodes on the page with names, IDs, types, and positions.
get_selection_info	Return full property sets (geometry, styles) for nodes in the active selection.
get_node_info	Return metadata, bounding boxes, style refs, and hierarchy path given the node IDs.
get_node_info_by_types	Traverse a parent and return children information that match the specified type.
get_result_image	Rasterise the visible design to PNG/JPEG and return an encoded image blob.
get_page_structure	Return a full node tree of the page, including order and absolute transforms.
export_json	Return the entire page as a structured JSON format.
import_json	Reconstruct a design on the page from a serialized JSON structure string.
Layout	
set_padding	Set per-side padding inside an auto-layout frame for internal spacing control.
set_axis_align	Set child alignment on primary/cross axes.
set_layout_sizing	Set width/height mode of the layout.
set_item_spacing	Set uniform gap distance between auto-layout children.

set_layout_mode	Set layout direction and optional wrapping.
Operation	
move_node	Translate a node to new x, y coordinates and optionally reparent it.
clone_node	Duplicate a node, retaining constraints and styles in the copy.
resize_node	Set width/height while preserving position or constraints.
delete_node	Remove one or more nodes from the canvas and history stack.
group_nodes	Group selected nodes into a group layer.
ungroup_nodes	Dissolve a group, restoring its children to the parent container.
rename_node	Change the name of a node for better organization.
rotate_node	Apply rotation in degrees based on the node centre.
boolean_nodes	Perform a boolean operation to create a composite node.
reorder_node	Move node forward/backward in sibling z-order.
Style	
set_fill_color	Set a solid color fill.
set_corner_radius	Set uniform or per-corner radii to round a node's corners.
get_styles	Return arrays of all shared text, color, and effect styles in the file.
set_opacity	Set overall opacity (0–100%) of the component.
set_stroke	Set stroke color, weight, alignment, and dash pattern.
set_fill_gradient	Set linear, radial, angular, or diamond gradient with multi-stop control.
set_drop_shadow	Add a drop shadow specifying color, blur radius, offset, and spread.
set_inner_shadow	Add an inner shadow confined within node bounds.
copy_style	Copy all visual properties from a source node to target nodes.
set_blend_mode	Set blend mode for compositing with layers beneath.
Text	
set_text_content	Replace the text string of one or many text nodes.
get_text_node_info	Return typography & content data for each text node in scope.
set_text_properties	Set font size, line height, letter spacing, alignment, and paragraphs.
set_text_decoration	Set underline, strikethrough, superscript, or case transformations.
set_text_font	Set font family and style weight; auto-loads missing fonts if present.

A.4.4 Figma Environment and Tool List

All tools available to the VLM are implemented using the official Figma Plugin API. When a tool is invoked, it executes a command on the Figma canvas and returns a result in a descriptive text and structured JSON format. This result reports the outcome of the operation, such as its success or failure, or provides any data queried from the environment.

The toolset is organized into seven comprehensive categories that cover the primary design operations available in Figma: `connection`, `creation`, `inspection`, `layout`, `operation`, `style`, and `text`.

To increase the reliability of our benchmark, we intentionally excluded features designated as "beta" in the Figma Plugin API documentation, such as advanced texture and blur effects. Table A.4 shows

Table A.5: High-level Tool Categories

Category	Purpose
`Connection`	Establishes and manages the connection to the Figma environment.
`Creation`	Creates new design elements, such as rectangles, frames, and other objects.
`Inspection`	Queries the current state of the design canvas, including the properties of existing elements.
`Layout`	Adjusts the alignment, distribution, and auto-layout properties of target components.
`Operation`	Manipulates the hierarchy, position, and grouping of design elements.
`Style`	Controls the visual appearance and aesthetic properties of components, including their fills, strokes, and effects.
`Text`	Edits textual content and modifies text-specific styling properties like font, size, and weight.

the full list of the tools included in the benchmark.

Prompt Design for the Replication and Modification Task

We employ a modular design for our task instruction prompts to ensure coherent context across all tasks. This approach involves two common instruction blocks, `Figma Tool Basics` and `Agency Principle`, which are added to each task-specific prompt as a variable.

The `Figma Tool Basics` block, shown in Figure A.15, outlines the VLM with basic knowledge about the Figma environment and general principles for structuring a design hierarchy. Based on findings from our early experiments, we identified that incorrect handling of layout and text elements was the most frequent source of task failure. Therefore, this block includes explicit guidelines on these topics to mitigate common errors and prevent significant performance degradation.

The `Agency Principle` instruction block, shown in Figure A.16, defines the operational protocol for the VLM to function as an autonomous agent. It instructs the model to complete the UI design task through an iterative process of reasoning and tool use. A primary goal of this prompt is to prevent the model from prematurely terminating the task, thereby encouraging persistent, multi-step problem-solving.

The instruction prompt for the Replication Task, shown in Figure A.17, directs the model to reproduce a target design from a ground-truth image. The prompt incorporates our standard `Agency Principle` and `Figma Basics` blocks to provide the necessary operational and environmental context. To further aid the models in spatial operations, the prompt also specifies the target frame dimensions (width and height), which define the coordinate system for the model.

For the reader's clarity when reviewing the figure, we note a minor terminological inconsistency in the prompt text: the `Figma Basics` block is referred to as `UI Design Principles`.

The instruction prompt for the Modification Task, shown in Figure A.18, is largely identical to that of the Replication Task. The key distinction is the inclusion of an additional `Text` field, which contains the specific natural language instruction detailing the required design modification.

Figure A.15: **Figma Tool Basics Prompt:** A prompt for guiding models on the use of the tools for producing a design in the Figma environment.

```
1. Figma Tool Basics
- In Figma tool calls, each design element appears as a node representing either a container (frame/
component) or a leaf (shape/text).
- Nodes provide uniform structural data exposing their unique properties.
- Coordinates are global: all nodes sit relative to the canvas origin (0, 0) at the top-left.

2. Node Hierarchy
- All nodes live in one rooted tree mirroring the layer list.
- Parent-child links create the hierarchy, and a node's index in its parent sets both z-order and
sidebar order.
- When child nodes (leaf) outgrow their parent nodes (container), they will be clipped.

3. Container Layout
- With auto layout applied to the frame, Figma automatically manages direction, gap, padding, and
resizing of the container and the children.
- So, manual layout property changes must account for these automatic adjustments of size and
position.
- Enable auto layout only when confident, as it can cause unexpected shifts.

4. Text Mechanics
- Text nodes expose font family, style, size, and other typography traits independent of layout.
- Resizing the text node doesn't scale the text; excess text simply overflows.
- Adequately set the text node size and alignment to avoid overflow.
```

Figure A.16: **Agency Principle Prompt:** A prompt for guiding models on the behavior as an agent when interacting with the environment.

```
1. Persistence
Keep iterating until the instruction is fully met and confirmed. Do not end the turn early.
2. Tool use
Interact with the canvas via the provided Figma-control tools.
3. Keen Examination
Carefully examine the instructions and image (if provided) and follow them accordingly.
```

Prompt Design for the Single-Turn Tool-based Generation and Code-based Generation Task

For our ablation study, we designed specialized prompts for two additional conditions: single-turn tool-based generation and code-based generation. Both of these conditions were evaluated exclusively on the Replication Task.

For the single-turn tool-based generation condition, we modified the standard prompt by replacing the `Agency Principle` block with a new `Tool Use Principle` block, shown in Table A.19. The new prompt instructs the model to forgo the iterative agentic loop. Instead, its objective is to generate a single, exhaustive tool invocation capable of constructing the entire target design in one turn.

For the code-based generation condition, our prompt is based on the methodology from Design2Code (Si et al., 2024) to closely approximate their experimental setup. The full prompt is shown in Table A.20. While we reproduced the core structure of the original prompt, we introduced two modifications to integrate it into our framework: We instructed the model to wrap the generated HTML/CSS code in triple backticks to ensure reliable parsing of the output. We explicitly predefined the output canvas dimensions (width and height in pixels). This modification ensures a fair comparison with our tool-based methods, where screen dimension information is also provided to assist with spatial operations.

Figure A.17: **Design Replication Prompt:** A prompt for replicating a UI design. The "${}" indicates a variable.

```
**Context**
You are a UI-design agent with access to Figma via tool calls.
Follow the **Instruction** to generate a UI design.
Refer to the **Agency Principles** and **UI Design Principles** for guidance.

**Agency Principles**
${agencyPrinciples}

**Figma Basics**
${figmaInstruction}

**Instruction**
Please analyze the following image and reproduce the UI design inside the existing "Main Screen"
frame in the Figma, exactly.
The frame size is ${width}x${height} pixels.
```

Figure A.18: **Design Modification Prompt:** A prompt for modfiying a UI design. The "${}" indicates a variable.

```
**Context**
You are a UI-design agent with access to Figma via tool calls.
Follow the **Instruction** to modify a UI design.
Refer to the **Agency Principles** and **Figma Basics** for guidance.

**Agency Principles**
${agencyPrinciples}

**Figma Basics**
${figmaInstruction}

**Instruction**
Please analyze the provided screen image and text instruction, then update the UI design within the
existing "Main Screen" frame in Figma to precisely match the image and the instruction.
The frame size is ${width}x${height} pixels.
Text: ${instruction}
```

Modification Task Prompt Examples

The modification task consists of three sub-tasks: `attribute update`, `component insertion`, and `mode change`. For `attribute update` and `component insertion`, each task instance is paired with a unique, case-specific instruction prompt. To illustrate these instructions, we include representative examples from the dataset below. Each example presents two images depicting the initial (base) state and the final (target) state, along with the corresponding text instruction. For every subtask, the target image is provided to the model during inference to ensure the model approximates the ground truth result.

- **Attribute Change:** This sub-task requires the model to modify a property of an existing element. As illustrated in Figure A.21, the model is provided with the base image and a case-specific text instruction. These instructions intentionally use qualitative, relational descriptions (e.g., resizing a "popup modal" to be "slightly below the bottom edge of the 'Go Back' button") rather than precise numerical values. This design choice aims to reflect real-world scenarios where designers communicate changes colloquially.
- **Component Insertion:** For this sub-task, the model must add a new element to the design. The

Figure A.19: **Tool-Use Principle Prompt:** A prompt for outlining the guidance for the single-turn tool-based generation.

```
Your task is to produce an array of tool (function) calls necessary to recreate the design in one
turn.
Include every necessary tool (function) calls and do not output any text other than the function
calls themselves.

1. Plan first
Briefly outline the key steps you will take.
2. Be exhaustive
Consider all parameters, options, ordering, and dependencies necessary to recreate the design in one
turn.
3. Deliver
Based on the plan, respond with the exact sequence of tool(function) calls it should run.
```

Figure A.20: **Code-Based Generation Prompt:** A prompt for replicating a UI design using code. The "${}" indicates a variable.

```
You are an expert web developer who specializes in HTML and CSS. A user will provide you with a
screenshot of a mobile app.
You need to return a single html file that uses HTML and CSS to reproduce the given mobile app.
Include all CSS code in the HTML file itself. If it involves any images, use "rick.jpg" as the
placeholder.
Some images on the webpage are replaced with a blue rectangle as the placeholder, use "rick.jpg" for
those as well.
Do not hallucinate any dependencies to external files.
You do not need to include JavaScript scripts for dynamic interactions. Pay attention to things like
size, text, position, and color of all the elements, as well as the overall layout.
Respond with the content of the HTML+CSS file. Wrap the code in backticks.
The page must be designed to match ${width}-pixel width and ${height}-pixel height.
```

instruction, also unique to each case (Fig. A.22), specifies the required visual properties, internal structure, and location of the new component relative to the existing design components.

- **Mode Change:** Unlike the other two sub-tasks, `mode change` utilizes a single, generic instruction for all instances in its set (Fig. A.23). This approach is used because the required transformation is uniform across all cases, which involves applying a color palette shift to toggle between light and dark modes.

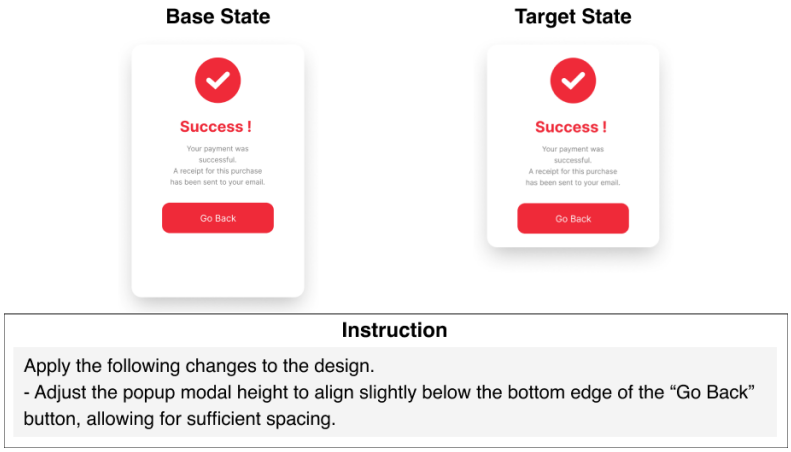


Figure A.21: **Modification Task-1 Instruction Example** An example task case from the Modification Task-1, *attribute update*, with the instruction.

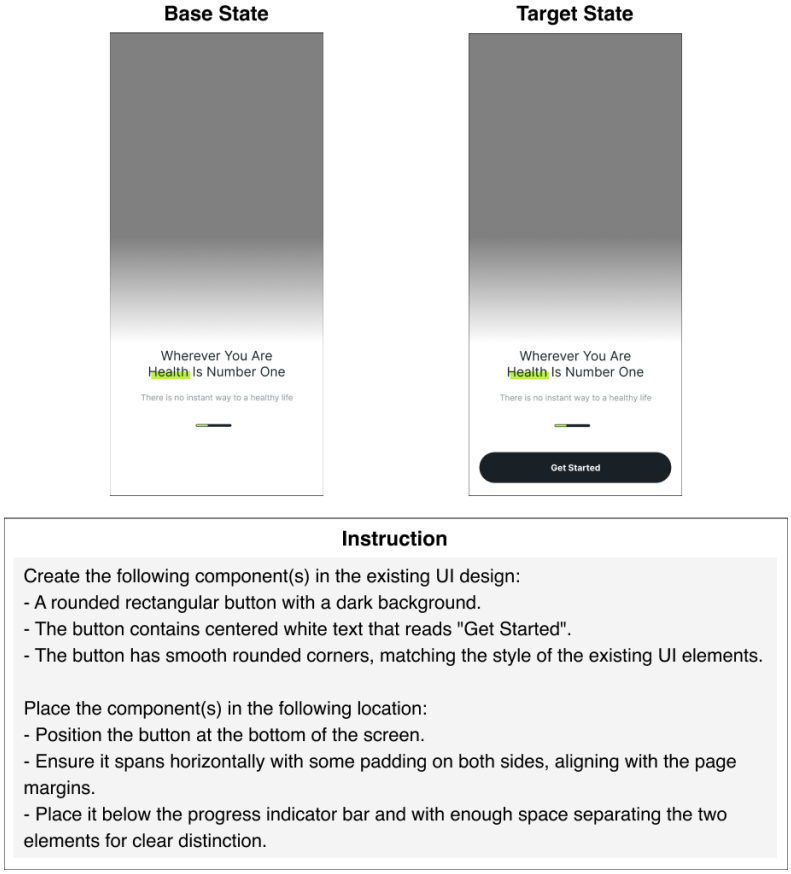


Figure A.22: **Modification Task-2 Instruction Example** An example task case from the Modification Task-2, *component insertion*, with the instruction.

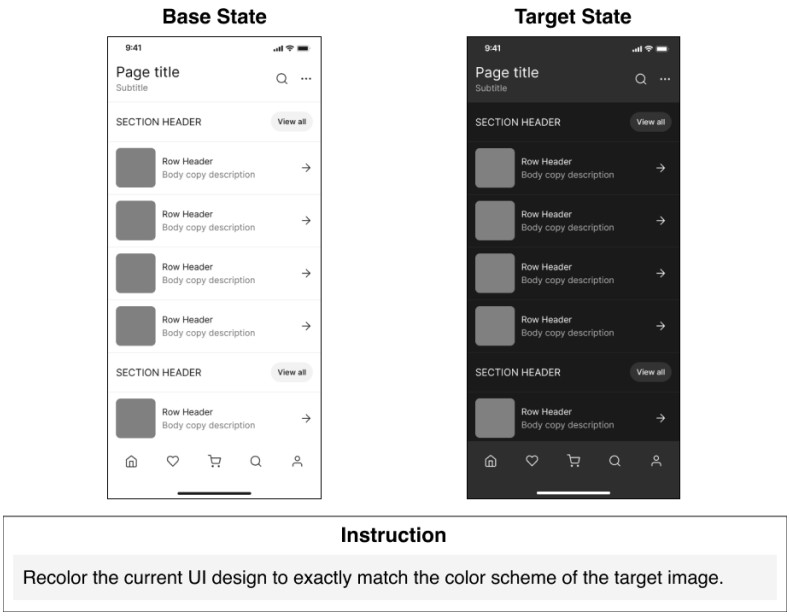


Figure A.23: **Modification Task-3 Instruction Example** An example task case from the Modification Task-3, *mode change*, with the instruction.

A.5 Metric Implementation Details

We evaluate model performance by adapting established metrics from code-based UI generation to tool-based generation, as both produce outputs with associated image and tree-based representations (HTML or JSON). However, as discussed in the *Related Work* section, structural mismatches between code-based outputs and tool-generated designs make direct reuse of existing metrics infeasible. To address this, we adapt the evaluation protocol to the newly suggested design environment, aligning the structured outputs (exported as JSON) from model generations with those of the ground-truth designs for comparison.

Drawing inspiration from the theory of human visual processing, we propose a hierarchical metric framework designed for human-centric interpretation of the evaluation results. The framework introduces two metric categories: (i) *perceptual similarity*, which measures similarity in the context of human visual perception; and (ii) *component-wise similarity*, which evaluates similarity at an individual component level.

A.5.1 Perceptual Similarity Metrics

The benchmark decomposes the visual similarity in design into three hierarchical levels based on the theory of the human bottom-up visual processing: features, patterns, and objects [41]. The hierarchy explains how a human sequentially extracts a meaning from an image, assembling visual features (e.g., edges) into patterns (e.g., a rectangular shape), and inferring a semantic object from it (e.g., a button). By mapping the metrics to three levels, the benchmark adds interpretability to contrasting trends between metrics within the same design. Precisely, we adopt algorithmic techniques designed to approximate visual characteristics that closely correspond to each stage of processing.

- ***Structural Similarity Index (SSIM)***: The SSIM captures mid-level structural patterns by comparing local means μ, variances σ^2, and cross–covariance σ_{xy} between two image patches. Hence, it exposes the low-level composition of pixels that build the structure of a shape (e.g., size and orientation) in design, corresponding to the feature. We calculate SSIM$(\cdot)$ using the `scikit-image` library with a 7×7 Gaussian window (default setting) and channel-wise averaging (`multichannel=True`).

  $$\mathrm{SIM}_{\mathrm{feat}}(s_t, s_{GT}) = \mathrm{SSIM}(I_t, I_{GT})$$

  where I_t and I_{GT} denote the rasterized images of s_t and s_{GT}, respectively.

- ***Saliency Map Similarity***: The saliency map estimates human attention based on the composition of low-level visual features, highlighting regions that may guide focus within a UI design. We generate saliency maps $\Phi(\cdot)$ using the pretrained model named `UMSI++` [15], resize both ground-truth and generated images to 320×240, and compute pattern-level similarity via the Pearson Correlation Coefficient (CC$(\cdot)$) between the two maps. Each saliency map is first min-max scaled to $[0, 1]$ and then standardized (zero mean, unit variance) before comparison. Inference is performed in `eval()` mode with all stochastic layers disabled and the random seed fixed to 42 for reproducibility.

  $$\mathrm{SIM}_{\mathrm{pat}}(s_t, s_{GT}) = \mathrm{CC}(\Phi(I_t), \Phi(I_{GT}))$$

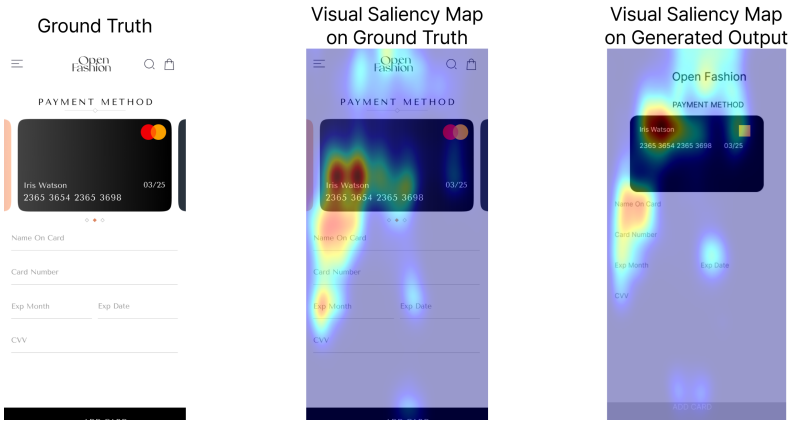


Figure A.24: **Visual saliency comparison** The middle and right panels show inferred saliency overlays on the ground truth and generated UIs, respectively. Similarity scores (CC = 0.5419) quantify the alignment between human and model attention.

- ***BLIPScore***: The BLIP caption approximates object-level semantics by generating textual descriptions of a design image, which reflect how humans may semantically interpret its content. We generate captions using the `Salesforce/blip2-opt-2.7b` model with greedy decoding (`num_beams=1`, `do_sample=False`). `sentence-transformers/all-MiniLM-L6-v2` is used to embed the resulting texts. All inference steps are conducted in `eval()` mode with the random seed fixed to 42 to ensure deterministic outputs. Semantic similarity is then computed as the cosine similarity between the caption embeddings:

$$\mathrm{SIM}_{\mathrm{obj}}(s_t, s_{GT}) = \frac{\langle\, \psi(I_t),\, \psi(I_{GT})\,\rangle}{\|\psi(I_t)\| \cdot \|\psi(I_{GT})\|}$$

  where $\psi(\cdot)$ denotes the embedding of the generated caption for the given image.

A.5.2 Component-wise Similarity Metrics

We adapt the component-level matching principles from prior works [34, 22], which evaluate how well individual UI elements (e.g., text, buttons, containers) are reproduced in the generated output. Unlike static input settings (e.g., sketches or images), our task involves tool-based, multi-turn generation within a design environment, where outputs are structured and semantically labeled. Accordingly, we reinterpret component-wise similarity in the context of tool-generated Figma designs by leveraging structured metadata (e.g., bounding boxes, fill styles, characters) for fine-grained element matching. This preserves the per-element evaluation spirit of Design2Code, while aligning with our scenario where models construct editable, design-native artifacts.

The evaluation comprises two stages: (i) Component Matching via bounding alignment and and (ii) Attribute Similarity calculation across matched pairs. The resulting similarity score reflects structural and semantic consistency between generated and ground-truth states.

Component Matching

Our matching strategy treats text and non-text components separately due to their distinct properties. Let $\mathcal{C}_{\mathrm{GT}}$ and $\mathcal{C}_{\mathrm{t}}$ denote the sets of ground-truth and generated components, respectively.

Non-text components We compute the Intersection over Union (IoU) between the ground-truth and generated boxes B. The cost C is defined as:

$$C_{ij} = 1 - \text{IoU}(B_t, B_{GT})$$

Only matches above an IoU threshold (e.g., 0.5) are retained.

Text components Text boxes are matched using a composite similarity score combining:

- **Text similarity**(S_{text}): computed using a mix of length ratio and character-level Jaccard similarity.
- **Position similarity**(S_{pos}): defined by the L-infinity distance between box centers.

The overall cost is computed as a weighted combination:

$$C_{ij} = \alpha \cdot (1 - S_{\text{text}}) + \beta \cdot (1 - S_{\text{pos}})$$

with equal weights $\alpha = \beta = 0.5$ by default.

Block Match The cost matrices C_{ij} for text and non-text components are used as input to the Hungarian algorithm, which computes one-to-one assignments between $\mathcal{C}_{\text{GT}}$ and $\mathcal{C}_{\text{t}}$ for each group. The resulting matched pairs are combined into a unified set $\mathcal{M}^\star$. The Block Match Score is then defined as:

$$\text{S}_{\text{match}} = \frac{|\mathcal{M}^\star|}{|\mathcal{C}_{\text{GT}}|}. \tag{1}$$

Similarity Metrics Calculation

We compute three attribute-level similarity metrics based on the matched component pairs $\mathcal{M}^\star$ between $\mathcal{C}_{\text{GT}}$ and $\mathcal{C}_{\text{t}}$. All metrics are averaged over the matched pairs and capture different aspects of visual similarity.

Position Similarity For each $(i, j) \in \mathcal{M}^\star$, we compute the Euclidean distance between the centers of the bounding boxes and normalize it by the larger diagonal of the two boxes:

$$\text{SIM}_{\text{pos}} = \frac{1}{|\mathcal{M}^\star|} \sum_{(i,j)\in\mathcal{M}^\star} \left(1 - \frac{\|c_i - c_j\|_2}{\max(\text{diag}_i, \text{diag}_j)}\right) \tag{2}$$

Color Similarity For matched pairs that both have solid fill colors, we compute the normalized Euclidean distance between their RGB values:

$$\text{SIM}_{\text{col}} = \frac{1}{|\mathcal{M}^\star_{\text{color}}|} \sum_{(i,j)\in\mathcal{M}^\star_{\text{color}}} \left(1 - \frac{\|\text{rgb}_i - \text{rgb}_j\|_2}{\sqrt{3} \cdot 255}\right) \tag{3}$$

where $\mathcal{M}^\star_{\text{color}} \subseteq \mathcal{M}^\star$ includes only pairs where both components define a solid color.

Text F1 Score Unlike the above metrics, text similarity is computed independently of component matching. We extract all text strings from s_{GT} and s_{new} into multisets $\mathcal{T}_{\text{GT}}$ and $\mathcal{T}_{\text{t}}$, respectively, and compute a global F1 score over string occurrences:

$$\text{SIM}_{\text{text}} = \text{F1}(\mathcal{T}_{\text{t}}, \mathcal{T}_{\text{GT}}) \tag{4}$$

This metric evaluates whether the correct text contents are present in the generated design, regardless of their positions or bounding boxes.

Interpretation Position and color similarity are computed only over successfully matched components in $\mathcal{M}^\star$, and thus reflect how well aligned or visually faithful each matched pair is. In contrast, text similarity assesses global content reproduction by comparing the sets of texts used, irrespective of spatial location or matching

Component-wise Similarity

We compute the overall component-wise similarity by averaging the four component-level scores introduced above: Block Match Score, Position Similarity, Color Similarity, and Text F1 Score. Formally, the final component-wise score is calculated as:

$$\begin{aligned}\mathrm{SIM}_{\mathrm{comp}}(s_t, s_{GT}) = \frac{1}{4}\big(&\mathrm{S}_{\mathrm{match}} + \mathrm{SIM}_{\mathrm{pos}} \\ &+ \mathrm{SIM}_{\mathrm{col}} + \mathrm{SIM}_{\mathrm{text}}\big)\end{aligned}$$

where all component-level metrics are computed between the generated design s_t and the ground-truth design s_{GT}. This unified score reflects both structural and content-level consistency between the generated and ground-truth UI designs.

A.5.3 Metric Aggregation Protocol

For each benchmark case, we compute similarity scores between the generated design s_t and the ground-truth design s_{GT} using a family of metric functions $\mathrm{SIM}_m(s_t, s_{GT})$. Here, m is selected from the metric set:

$$\mathcal{M}_{\mathrm{metric}} = \{\mathrm{SSIM}, \mathrm{Saliency}, \mathrm{BLIP}, \mathrm{Component\text{-}wise}\}.$$

Then, for each metric m, we report the model-level score as the average over all test cases:

$$\mathrm{Score}_m = \frac{1}{N}\sum_{i=1}^{N} \mathrm{SIM}_m\left(s_t^{(i)}, s_{GT}^{(i)}\right)$$

Replication

The model generates a design s_t from scratch using only the reference image s_{GT} in the replication task. We report Score_m along with its standard deviation across all examples, denoted by $\pm$.

Modification

The model transforms an initial state s_{old} into an updated design s_{new}, targeting the desired state s_{GT} in the modification task. We evaluate each metric by comparing s_{new} to s_{GT}, and further compute the score improvement relative to the base design s_{old}:

$$\Delta_m = \mathrm{SIM}_m(s_{new}, s_{GT}) - \mathrm{SIM}_m(s_{old}, s_{GT})$$

where the reported Score_m reflects the similarity between s_{new} and s_{GT}. Δ denotes the improvement from the base design.

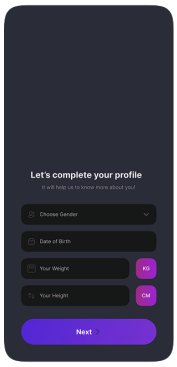
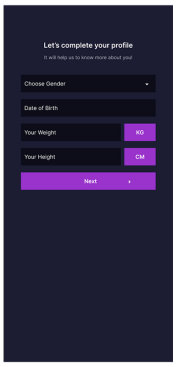
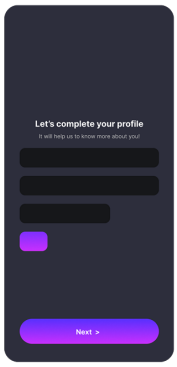
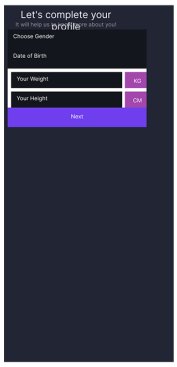
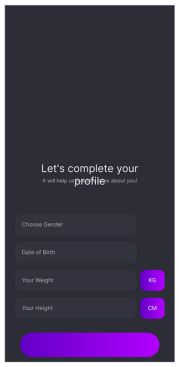
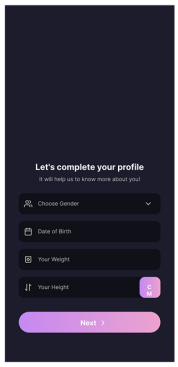
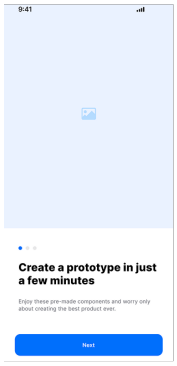
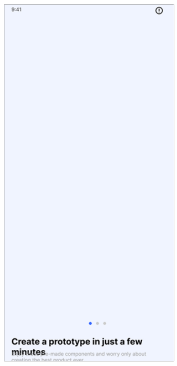
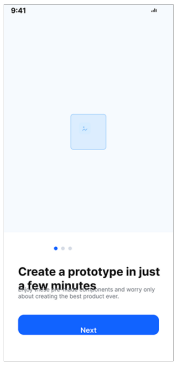
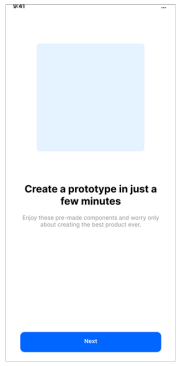
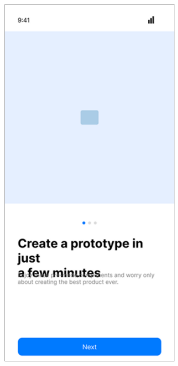
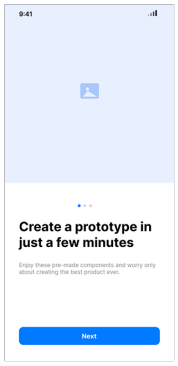
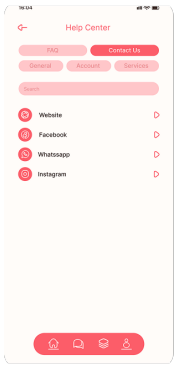
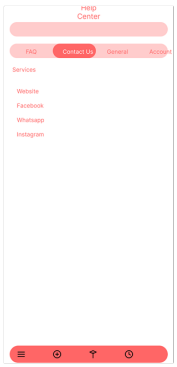
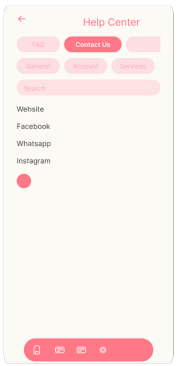
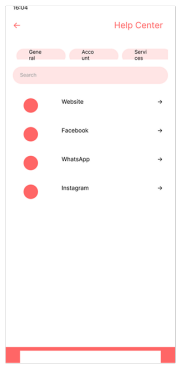
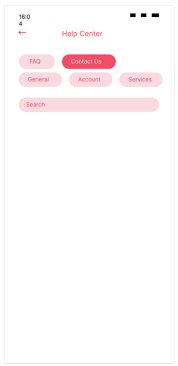
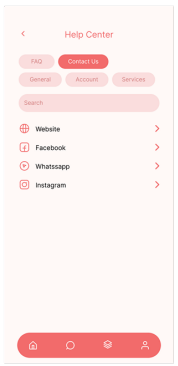
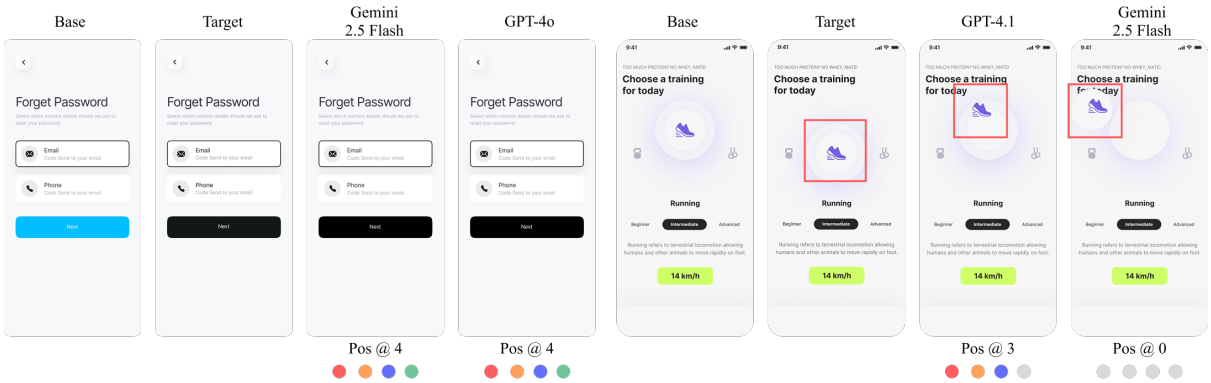
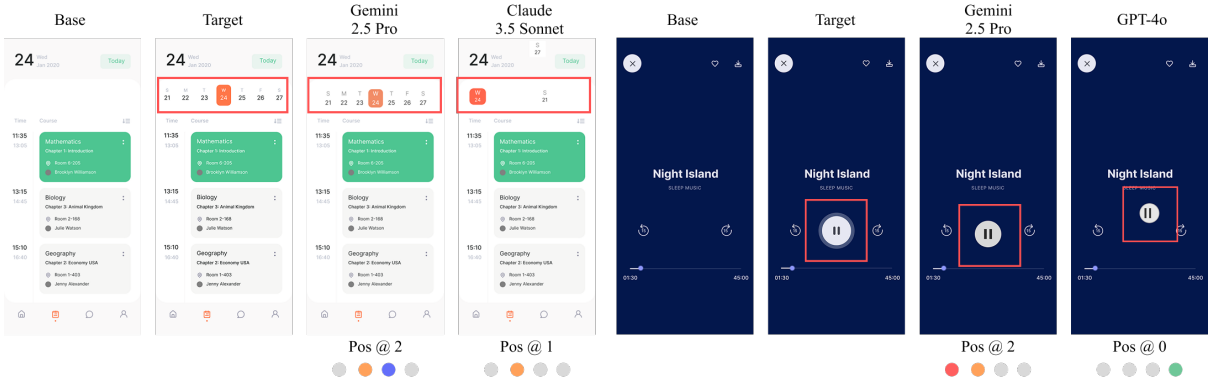
Ground Truth	GPT-4o	GPT-4.1	Claude 3.5 Sonnet	Gemini 2.5 Flash	Gemini 2.5 Pro
					
SSIM	0.740	0.392	0.756	0.798	0.816
Saliency Sim.	-0.101	0.309	-0.211	0.784	0.809
BLIPScore	0.553	0.154	0.432	0.433	0.675
Comp. Sim.	0.660	0.493	0.609	0.749	0.762
					
SSIM	0.889	0.855	0.871	0.886	0.898
Saliency Sim.	0.040	0.565	-0.125	0.756	0.930
BLIPScore	0.620	0.776	0.536	0.840	0.578
Comp. Sim.	0.776	0.744	0.769	0.801	0.818
					
SSIM	0.880	0.914	0.873	0.886	0.870
Saliency Sim.	0.416	0.184	0.309	0.311	0.754
BLIPScore	0.468	0.609	0.809	0.731	0.809
Comp. Sim.	0.635	0.513	0.667	0.677	0.744

Figure A.25: **Qualitative results with evaluation scores for the replication task.** Each row shows one example (`gid-51-43`, `gid-1-11`, `gid-78-11`), comparing model outputs to the ground truth. Reported metrics include SSIM, saliency map similarity (Saliency Sim.), BLIPScore, and component-wise similarity (Comp. Sim.). Higher values indicate greater similarity to the ground truth.



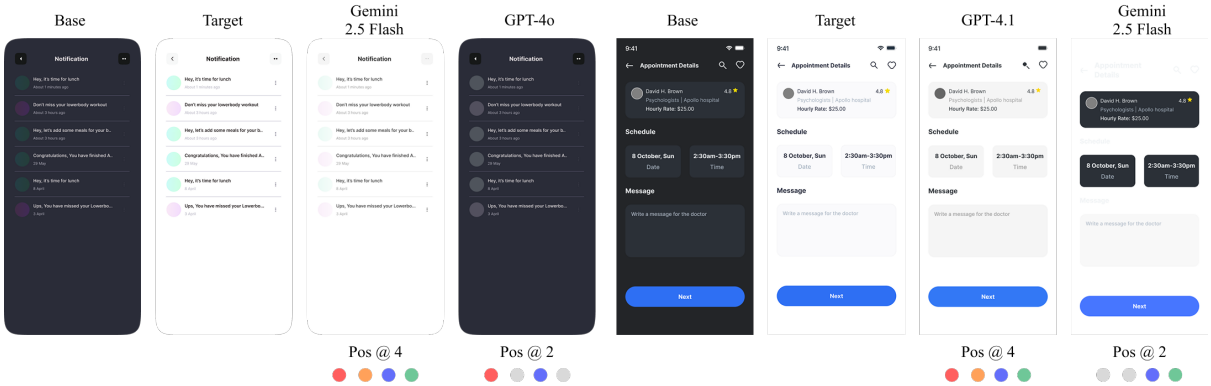
(a) Task-1, color adjustment, gid20

(b) Task-1, position adjustment, gid82



(c) Task-2, gid8

(d) Task-2, gid3



(e) Task-3, gid51-8

(f) Task-3, gid94-7

ΔSSIM ≥ 0 ΔSaliencySim ≥ 0 ΔBLIPScore ≥ 0 ΔComponentSim ≥ 0 : Indicates the corresponding Δ metric equals 0

Figure A.26: **Qualitative results with Pos@K annotations for the modification task.** Each example shows model outputs generated by editing an initial base state toward a given target design. The red bounding boxes are added for illustrative purposes and are not part of the actual UI. Below each output, the Pos@K label indicate which of the four evaluation metrics exhibit non-negative Δ improvements compared to the target. If all metric scores remain exactly zero despite valid operations, the case is considered as Pos@0, indicating no meaningful change. Cases with no valid operation detected are reported as failure cases.

A.6 Additional Experiments

A.6.1 Task Difficulty Effect

Table A.6 shows the variation in model performance based on the task difficulty. We define two difficulty levels based on the node count (a number of design elements) in the design: *standard* (bottom 85% percentile) and *hard* (top 15% percentile). Correlation analysis revealed negative associations between node count and both SSIM ($r = -0.274$, $p < 0.01$) and component-wise similarity ($r = -0.349$, $p < 0.01$). When evaluating performance separately on the hard tasks, we observed a general decrease in overall scores. Especially, Gemini-2.5-Pro demonstrated a stronger decrease in SSIM, whereas Gemini-2.5-Flash exhibited robustness in component-wise similarity scores.

Table A.6: Replication Task Scores Per Difficulty

Difficulty	Model	SSIM	Saliency	BLIP	Comp. Wise
Standard (n=1258)	GPT-4o	0.749	0.479	0.492	0.683
	GPT-4.1	0.776	0.613	**0.649**	**0.726**
	Claude-3.5-Sonnet	0.732	0.482	0.515	0.679
	Gemini-2.5-Flash	0.748	0.625	0.568	0.710
	Gemini-2.5-Pro	**0.788**	**0.634**	0.621	0.703
Hard (n=225)	GPT-4o	0.685	0.472	0.510	0.613
	GPT-4.1	**0.716**	0.605	**0.689**	0.665
	Claude-3.5-Sonnet	0.689	0.493	0.532	0.595
	Gemini-2.5-Flash	0.670	0.583	0.583	**0.665**
	Gemini-2.5-Pro	0.698	**0.606**	0.620	0.650

: the best score within *each difficulty*.

A.6.2 Human Preference Study

Following the methodology outlined in Design2Code [34], we collect a pairwise human preference on design with design experts. We recruited design experts through the Prolific platform, screening for participants who were fluent in English and had professional experience in relevant fields, such as UI/UX design, responsive design, or A/B testing. The instructions provided to the participants, shown in Figure A.30, were also closely adapted from the prior work to ensure methodological consistency.

Each annotator sees two design cases with one ground truth and labels a win, lose, or tie. We collect 3 labels per design pair on 100 examples sampled through stratified sampling (weighted by UI type and design complexity) from our replication results. The final label is determined by majority voting.

Figure A.27 illustrates pairwise human preferences comparing GPT-4.1 (baseline) against other models. Consistent with our metric results, GPT-4.1 demonstrates a strong advantage over other models. This inclination is especially apparent against Claude-3.5-Sonnet and GPT-4o, which showed a subpar performance in our metric result. The results indicate that our metric well aligns with the human preference setup.

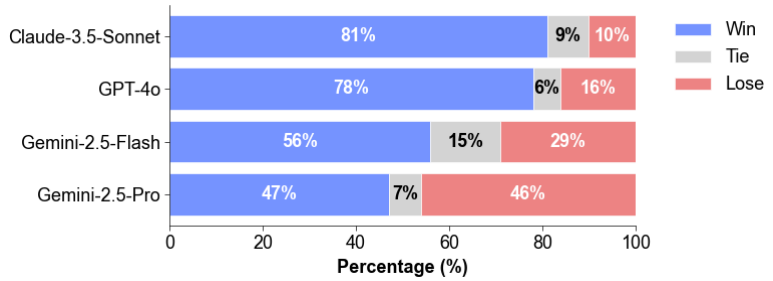


Figure A.27: **Human Pairwise Preference Result**: The baseline, GPT-4.1, was majorly preferred over the other models, consistent with our metric result.

A.6.3 Performance Evolution Across Turns

We analyze how agent performance evolves across turns by visualizing turn-wise changes in four metrics. We observe that all models exhibit measurable improvements from the initial state to the final turn, but with distinct optimization characteristics. Notably, Claude-3.5-Sonnet shows the most stable progression, with BLIP improving from -0.008 to 0.518 and saliency increasing by +0.076, alongside low variance (SSIM std. 0.042). GPT-4o exhibits similarly balanced improvements, including a BLIP gain of +0.505 with low saliency variability. GPT-4.1 achieves the largest overall gains, particularly in saliency (+0.204) and component-wise similarity (+0.287), though with higher variance (saliency std. 0.120). Gemini-2.5-Flash and Gemini-2.5-Pro also show monotonic improvements, with Gemini-2.5-Pro achieving the largest SSIM gain (+0.077) and a saliency gain of +0.222, while both variants exhibit higher turn-wise variability (saliency std. 0.133). Across models, no SSIM or BLIP declines are observed in the updated results.

These patterns reflect distinct strategic tendencies. GPT-4.1 rapidly reaches high initial scores (e.g., SSIM = 0.697) and finishes early in most cases, as shown in the turn range distribution in Figure A.29, suggesting a fast-convergence strategy. However, this often leads to performance degradation in longer tasks. Conversely, Claude-3.5-Sonnet and Gemini-2.5-Pro maintain gradual, low-variance improvements over 50 turns, indicating a more incremental optimization strategy. SSIM shows the highest standard deviation across models, indicating potential trade-offs between structural fidelity and iterative refinement. These findings underscore that a high final score does not necessarily indicate process stability. Models that converge to similar end states may follow markedly different turn-wise trajectories, with varying degrees of volatility and correction behavior. Benchmarking multi-turn agents should therefore evaluate not only outcomes but also the optimization dynamics that produce them, to more faithfully capture the characteristics of real-world design workflows.

A.6.4 Tool Invocation Statistics

Tool Precision (range: $[0, 1]$) measures the proportion of model-used tools that are correct. It is computed as the intersection between model-used tools and human-annotated tools, divided by the total number of unique tools used by the model. Tool Recall (range: $[0, 1]$) measures the proportion of human-annotated tools that the model successfully used. It is computed as the intersection divided by the total number of human-annotated tools. Tool Diversity represents the average number of unique tools used per case, calculated as the mean across all cases. Excluding the three Connection tools, all evaluation cases require at least one Inspection tool, yielding a range of $[1, 47]$.

Table A.7: Turn-by-turn performance summary for the replication task(max 50 turns). Initial and final metric scores are shown at the first (s_0) and last (s_T) turns, respectively. Δ_m indicates the score change, and Std. denotes standard deviation over turns.

Model	**Metric**	**Initial**	**Final**	**Δ**	**Std.**
GPT-4o	SSIM	0.697	0.739	0.042	0.049
	Saliency	0.408	0.478	0.070	0.089
	Comp. Wise	0.429	0.671	0.242	0.097
	BLIP	-0.010	0.495	0.505	0.210
GPT-4.1	SSIM	0.696	0.767	0.070	0.054
	Saliency	0.408	0.612	0.204	0.120
	Comp. Wise	0.428	0.716	0.287	0.105
	BLIP	-0.009	0.655	0.664	0.265
Claude-3.5-Sonnet	SSIM	0.698	0.725	0.027	0.042
	Saliency	0.407	0.483	0.076	0.093
	Comp. Wise	0.429	0.666	0.237	0.084
	BLIP	-0.008	0.518	0.527	0.198
Gemini-2.5-Flash	SSIM	0.699	0.736	0.036	0.042
	Saliency	0.408	0.619	0.210	0.133
	Comp. Wise	0.427	0.702	0.275	0.104
	BLIP	-0.010	0.571	0.581	0.237
Gemini-2.5-Pro	SSIM	0.697	0.774	0.077	0.047
	Saliency	0.408	0.630	0.222	0.133
	Comp. Wise	0.429	0.694	0.265	0.088
	BLIP	-0.010	0.620	0.630	0.231

Failure Handling

To ensure stable execution, each tool invocation is retried up to three times, and samples that fail after all retries are skipped. This strategy mitigates transient network or server-side errors. We additionally observed a model-specific failure in Gemini-2.5-Flash during experiments: a `MALFORMED_FUNCTION_CALL` error in the Gemini API caused the model to terminate at turn 0 in 7 of 100 cases. These failed samples were excluded from evaluation. Other models did not exhibit similar API-level failures.

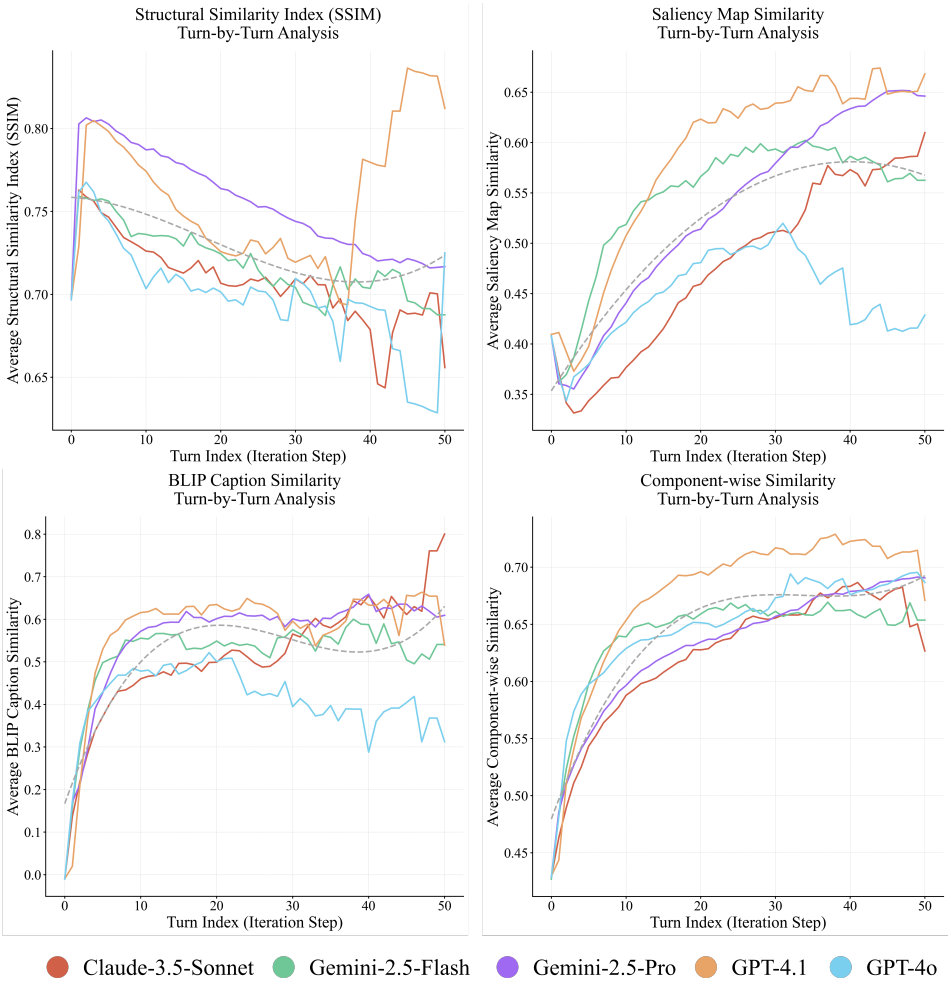


Figure A.28: **Turn-by-Turn Performance Curves for Key Similarity Metrics.**

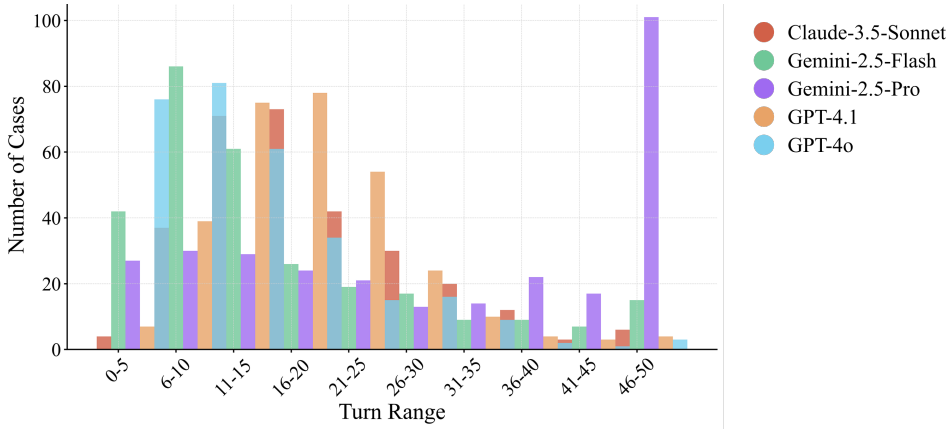


Figure A.29: **Max Turn Distribution by model in replication task.**

Table A.8: Tool invocation statistics per model for Task 1.

Model	Tool Precision	Tool Recall	Tool Diversity
GPT-4o	0.3548	0.9900	3.6100
GPT-4.1	0.4530	1.0000	3.4000
Claude-3.5-Sonnet	0.4080	0.9700	3.0200
Gemini-2.5-Flash	0.5017	1.0000	2.3100
Gemini-2.5-Pro	0.4670	0.9900	3.2700

Table A.9: Tool invocation statistics per model for Task 2.

Model	Tool Precision	Tool Recall	Tool Diversity
GPT-4o	0.5699	1.0000	5.1800
GPT-4.1	0.7215	1.0000	8.3100
Claude-3-5-Sonnet	0.7785	1.0000	3.8300
Gemini-2.5-Flash	0.6283	0.9300	3.9900
Gemini-2.5-Pro	0.5998	0.9000	5.3700

Table A.10: Tool invocation statistics per model for Task 3.

Model	Tool Precision	Tool Recall	Tool Diversity
GPT-4o	0.3340	1.0000	4.1400
GPT-4.1	0.4559	1.0000	4.6200
Claude-3-5-Sonnet	0.5168	1.0000	4.0400
Gemini-2.5-Flash	0.6530	1.0000	3.9200
Gemini-2.5-Pro	0.6309	1.0000	3.7800

Figure A.30: **Human Preference Study Instruction:** The instruction for the human preference study.

```
TASK OVERVIEW:
In this survey, you will be given a reference mobile UI design's screenshot (Ground Truth), as well
as two candidate mobile UI designs (Example A and Example B) that try to replicate the reference
mobile UI design. Your task is to judge which of the two candidates is closer to the reference.

This study aims to evaluate how well AI agents can replicate reference UI designs in a canvas-based
environment. Each UI design task involves a replication scenario where an AI agent is given a ground-
truth UI canvas (created by a human designer) and asked to reproduce it as closely as possible by
creating a new design on an empty canvas.

You will be shown screenshots of:
- The Reference Canvas (Ground Truth)
- Two Generated Canvases (produced by VLM agents attempting to replicate the reference)

INSTRUCTIONS:
1. For each question, you will see 3 images side by side:
   - Left: Baseline generation (Example A, baseline model)
   - Center: Reference image (Ground Truth)
   - Right: Comparison generation (Example B, different model)
   (A sample image is provided below for reference.)

2. Question: "Which generation is more similar to the ground truth?"

3. Choose one of the following options:
   - Left is better (Left image is more similar to reference)
   - Right is better (Right image is more similar to reference)
   - Tie (Both images are equally similar to reference)

4. COMPARISON GUIDE:

Initial Step: Content Check
- Text Content: Examine if the text on the candidate mobile UI design matches the reference. Pay
special attention to missing or extra content, especially key elements like titles.
- Image Content: Assess the placement of the blue placeholder blocks (for images).
- Primary Judgment Criterion: If one example has significant missing or additional content compared
to the other, it should be considered less similar to the reference.

Second Step: Layout Check
- Element Arrangement: If the content (text and images) of both examples is similarly good or bad,
proceed to evaluate the arrangement of these elements. Check if their organization, order, and
hierarchy match the reference.
- Secondary Judgment Criterion: If differences in layout are observed, the example with the layout
most similar to the reference should be rated higher.

Final Step: Style Check
- Style Attributes: Only if Example 1 and Example 2 are comparable in content and layout, examine the
 style elements like font style, color, and size.
- Tertiary Judgment Criterion: In cases where content and layout are equally matched, preference
should be given to the example with style attributes closer to the reference.

Overall Judgment
Based on the criteria in the order of priority (Content > Layout > Style), make an overall judgment
on which example (Example 1 or Example 2) is more similar to the reference mobile UI design.

Please complete all 30 questions (6 questions per image set x 5 image sets) in this form.

Click "Next" to start the survey.
```

Bibliography

[1] Anthropic. Introducing claude 3.5 sonnet. https://www.anthropic.com/news/claude-3-5-sonnet, June 2024. Accessed: 2025-05-15.

[2] Tony Beltramelli. pix2code: Generating code from a graphical user interface screenshot. In *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, EICS '18, New York, NY, USA, 2018. Association for Computing Machinery.

[3] Bill Buxton. *Sketching user experiences: getting the design right and the right design*. Morgan kaufmann, 2010.

[4] Chin-Yi Cheng, Forrest Huang, Gang Li, and Yang Li. Play: parametrically conditioned layout generation using latent diffusion. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org, 2023.

[5] DaEun Choi, Sumin Hong, Jeongeon Park, John Joon Young Chung, and Juho Kim. Creative-connect: Supporting reference recombination for graphic design ideation with generative ai. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, CHI '24, New York, NY, USA, 2024. Association for Computing Machinery.

[6] Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.

[7] Biplab Deka, Zifeng Huang, Chad Franzen, Joshua Hibschman, Daniel Afergan, Yang Li, Jeffrey Nichols, and Ranjitha Kumar. Rico: A mobile app dataset for building data-driven design applications. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*, UIST '17, page 845–854, New York, NY, USA, 2017. Association for Computing Machinery.

[8] Jennifer Ferreira, James Noble, and Robert Biddle. Agile development iterations and ui design. In *Agile 2007 (AGILE 2007)*, pages 50–58, 2007.

[9] Figma. Figma: The collaborative interface design tool, 2024.

[10] Figma. Figma community — explore free design files, plugins & more, 2025. Accessed: 2025-05-13.

[11] Zhi Gao, Bofei Zhang, Pengxiang Li, Xiaojian Ma, Tao Yuan, Yue Fan, Yuwei Wu, Yunde Jia, Song-Chun Zhu, and Qing Li. Multi-modal agent tuning: Building a VLM-driven agent for efficient tool usage. In *The Thirteenth International Conference on Learning Representations*, 2025.

[12] Aryan Garg, Yue Jiang, and Antti Oulasvirta. Controllable GUI exploration. *CoRR*, abs/2502.03330, 2025.

[13] Yi Gui, Zhen Li, Yao Wan, Yemin Shi, Hongyu Zhang, Bohua Chen, Yi Su, Dongping Chen, Siyuan Wu, Xing Zhou, Wenbin Jiang, Hai Jin, and Xiangliang Zhang. Webcode2m: A real-world dataset

for code generation from webpage designs. In *Proceedings of the ACM on Web Conference 2025*, WWW '25, page 1834–1845, New York, NY, USA, 2025. Association for Computing Machinery.

[14] Yi Gui, Yao Wan, Zhen Li, Zhongyi Zhang, Dongping Chen, Hongyu Zhang, Yi Su, Bohua Chen, Xing Zhou, Wenbin Jiang, and Xiangliang Zhang. Uicopilot: Automating ui synthesis via hierarchical code generation from webpage designs. In *Proceedings of the ACM on Web Conference 2025*, WWW '25, page 1846–1855, New York, NY, USA, 2025. Association for Computing Machinery.

[15] Yue Jiang, Luis A. Leiva, Hamed Rezazadegan Tavakoli, Paul R. B. Houssel, Julia Kylmälä, and Antti Oulasvirta. Ueyes: Understanding visual saliency across user interface types. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, CHI '23, New York, NY, USA, 2023. Association for Computing Machinery.

[16] Tae Soo Kim, DaEun Choi, Yoonseo Choi, and Juho Kim. Stylette: Styling the web with natural language. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, CHI '22, New York, NY, USA, 2022. Association for Computing Machinery.

[17] Hugo Laurençon, Léo Tronchon, and Victor Sanh. Unlocking the conversion of web screenshots into html code with the websight dataset, 2024.

[18] Gang Li, Gilles Baechler, Manuel Tragut, and Yang Li. Learning to denoise raw mobile ui layouts for improving datasets at scale. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, CHI '22, New York, NY, USA, 2022. Association for Computing Machinery.

[19] Jianan Li, Jimei Yang, Aaron Hertzmann, Jianming Zhang, and Tingfa Xu. Layoutgan: Generating graphic layouts with wireframe discriminators. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[20] Jie Li, Hancheng Cao, Laura Lin, Youyang Hou, Ruihao Zhu, and Abdallah El Ali. User experience design professionals' perceptions of generative artificial intelligence. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, CHI '24, New York, NY, USA, 2024. Association for Computing Machinery.

[21] Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. BLIP-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 19730–19742. PMLR, 23–29 Jul 2023.

[22] Ryan Li, Yanzhe Zhang, and Diyi Yang. Sketch2Code: Evaluating vision-language models for interactive web design prototyping. In Luis Chiruzzo, Alan Ritter, and Lu Wang, editors, *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 3921–3955, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics.

[23] Yuwen Lu, Yuewen Yang, Qinyi Zhao, Chengzhi Zhang, and Toby Jia-Jun Li. Ai assistance for ux: A literature review through human-centered ai, 2024.

[24] Yuwen Lu, Chengzhi Zhang, Iris Zhang, and Toby Jia-Jun Li. Bridging the gap between ux practitioners' work practices and ai-enabled design support tools. In *Extended Abstracts of the 2022 CHI*

Conference on Human Factors in Computing Systems, CHI EA '22, New York, NY, USA, 2022. Association for Computing Machinery.

[25] Mobbin Ltd. Mobbin: Ui & ux design inspiration for mobile & web apps, 2025. Accessed: 2025-05-13.

[26] Kevin Moran, Carlos Bernal-Cárdenas, Michael Curcio, Richard Bonett, and Denys Poshyvanyk. Machine learning-based prototyping of graphical user interfaces for mobile apps. *IEEE Transactions on Software Engineering*, 46(2):196–221, 2020.

[27] Runliang Niu, Jindong Li, Shiqi Wang, Yali Fu, Xiyu Hu, Xueyuan Leng, He Kong, Yi Chang, and Qi Wang. Screenagent: a vision language model-driven computer control agent. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, IJCAI '24, 2024.

[28] OpenAI. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024. Accessed: 2025-05-15.

[29] OpenAI. Gpt-4.1 mini. https://openai.com/index/gpt-4-1/, 2025. Accessed: 2025-05-13.

[30] OpenAI. Gpt-4.5: Openai's large language model. https://openai.com/index/gpt-4-5, 2025. Accessed: 2025-05-15.

[31] OpenAI. Introducing gpt-4.1 in the api. https://openai.com/index/gpt-4-1/, April 2025. Accessed: 2025-05-15.

[32] Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan, editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, Hong Kong, China, November 2019. Association for Computational Linguistics.

[33] Mitchel Resnick, Brad Myers, Kumiyo Nakakoji, Ben Shneiderman, Randy Pausch, Ted Selker, and Mike Eisenberg. Design principles for tools to support creative thinking. 2005.

[34] Chenglei Si, Yanzhe Zhang, Ryan Li, Zhengyuan Yang, Ruibo Liu, and Diyi Yang. Design2Code: Benchmarking multimodal code generation for automated front-end engineering. In Luis Chiruzzo, Alan Ritter, and Lu Wang, editors, *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 3956–3974, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics.

[35] Sketch. Stack Layout — sketch.com. https://www.sketch.com/docs/designing/stack-layout/, 2025. [Accessed 28-07-2025].

[36] Kihoon Son, DaEun Choi, Tae Soo Kim, and Juho Kim. Demystifying tacit knowledge in graphic design: Characteristics, instances, approaches, and guidelines. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, CHI '24, New York, NY, USA, 2024. Association for Computing Machinery.

[37] Erik Stolterman and James Pierce. Design tools in practice: studying the designer-tool relationship in interaction design. In *Proceedings of the Designing Interactive Systems Conference*, DIS '12, page 25–28, New York, NY, USA, 2012. Association for Computing Machinery.

[38] Zecheng Tang, Chenfei Wu, Juntao Li, and Nan Duan. Layoutnuwa: Revealing the hidden layout expertise of large language models. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024.

[39] Yuxuan Wan, Chaozheng Wang, Yi Dong, Wenxuan Wang, Shuqing Li, Yintong Huo, and Michael Lyu. Divide-and-conquer: Generating ui code from screenshots. *Proc. ACM Softw. Eng.*, 2(FSE), June 2025.

[40] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004.

[41] Colin Ware. *Visual thinking for design*. Elsevier, 2010.

[42] Jingyu Xiao, Yuxuan Wan, Yintong Huo, Zixin Wang, Xinyi Xu, Wenxuan Wang, Zhiyao Xu, Yuhang Wang, and Michael R. Lyu. Interaction2code: Benchmarking mllm-based interactive webpage code generation from interactive prototyping, 2025.

[43] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

[44] Mingyue Yuan, Jieshan Chen, Yongquan Hu, Sidong Feng, Mulong Xie, Gelareh Mohammadi, Zhenchang Xing, and Aaron John Quigley. Towards human-ai synergy in ui design: Supporting iterative generation with llms. *ACM Trans. Comput.-Hum. Interact.*, October 2025. Just Accepted.

[45] Sukmin Yun, Haokun Lin, Rusiru Thushara, Mohammad Qazim Bhat, Yongxin Wang, Zutao Jiang, Mingkai Deng, Jinhong Wang, Tianhua Tao, Junbo Li, Haonan Li, Preslav Nakov, Timothy Baldwin, Zhengzhong Liu, Eric P. Xing, Xiaodan Liang, and Zhiqiang Shen. Web2code: A large-scale webpage-to-code dataset and evaluation framework for multimodal llms. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 112134–112157. Curran Associates, Inc., 2024.

[46] Tianming Zhao, Chunyang Chen, Yuanning Liu, and Xiaodong Zhu. Guigan: Learning to generate gui designs using generative adversarial networks. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 748–760, 2021.

[47] Ting Zhou, Yanjie Zhao, Xinyi Hou, Xiaoyu Sun, Kai Chen, and Haoyu Wang. Declarui: Bridging design and development with automated declarative ui code generation. *Proc. ACM Softw. Eng.*, 2(FSE), June 2025.

Acknowledgment

석사 과정 동안 카이스트 인터렉션 연구실을 통해 연구자로서, 그리고 한 사람으로서 성숙해지는 시간을 보낼 수 있었습니다. 사려 깊은 구성원분들과 함께 밤새 긴 시간을 보내고 어려움과 성취를 나눌 수 있어서 행복했습니다. 지난 시간 동안 저에게 도움을 주신 모든 분들께 감사의 말씀을 드립니다.

- 저를 응원해 주시고 아낌없이 지원해 주신 부모님과 가족들에게 감사드립니다.
- 부족한 저를 기다려주시고 항상 최선의 길로 지도해주신 김주호 교수님께 감사드립니다.
- 저와 함께 연구를 진행하며 도움을 주신 교내외 공동 저자분들께 감사드립니다.
- 제가 이곳에서 생활하는 동안 저를 배려해 주시고 배움을 주신 KIXLAB 구성원분들께 감사드립니다.
- 함께 생활하며 힘이 되어준 CSTL 및 여러 HCI 연구실 구성원분들께 감사드립니다.
- 본 학위논문을 검토해 주시고 소중한 의견을 주신 오혜연 교수님과 김현우 교수님께 감사드립니다.

Curriculum Vitae

Name : Daeheon Jeong

Date of Birth : February 7, 1997

E-mail : daeheon.jeong@kaist.ac.kr

Educations

2024. 9. – Present M.S. in Artificial Intelligence, Kim Jaechul Graduate School of AI, KAIST

2020. 3. – 2024. 2. B.A. in Economics, College of Economics, Sogang University

Career

2023. 12. – Present Student Researcher, KIXLAB (KAIST Interaction Lab), School of Computing, KAIST

Publications

1. **D. Jeong**, S. Byun, K. Son, D. H. Kim, and J. Kim, "A Benchmark for Vision-Language Models on Tool-Based User Interface Design," *Proceedings of the AAAI Conference on Artificial Intelligence*, 2026.
2. D. H. Kim, **D. Jeong**, S. Yadgarova, H. Shin, J. Son, H. Subramonyam, and J. Kim, "PlanTogether: Facilitating AI Application Planning Using Information Graphs and Large Language Models," *Proceedings of the CHI Conference on Human Factors in Computing Systems*, 2025.
3. **D. Jeong** and H. Chu, "Visual Embedding of Screen Sequences for User-Flow Search in Example-driven Communication," *Proceedings of the Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*, 2025.
4. **D. Jeong**, F. Reuting, and H. Jeong, "Auditory Map Generation Using Boundary Detection in a Web Environment," *Proceedings of the HCI Society of Korea Conference*, 2022.
5. **D. Jeong**, Y. Kim, J. Park, and M. Cho, "Spatial Screen Reader Using Touch Interface for Visually Impaired," *Proceedings of the Korea Computer Congress*, 2022.