FigmaBench: Evaluating Design-to-Code Generation in Real-World Handoff Scenarios

Anonymous ACL submission

Abstract

Automating design-to-code translation remains a longstanding goal in software engineering. While Multimodal Large Language Models (MLLMs) have shown promise in design-to-code tasks, existing benchmarks rely solely on screenshots, discarding the structured metadata indispensable in real-world design handoff workflows. Furthermore, current evaluation paradigms are deficient, as code-level metrics fail to account for implementation variability and visual metrics overlook fine-grained defects, while standardized assessments for responsive behaviors remain absent. To address these limitations, we introduce FigmaBench, an industrial-grade benchmark for multimodal design-to-code generation. We collect raw design files from the Figma Community and apply a multi-stage pipeline to curate 1,234 high-quality samples, each comprising high-resolution screenshots and complete JSON metadata. We further propose a comprehensive evaluation framework with four complementary metrics: visual consistency (VCS), structural layout alignment (SLA), textual and stylistic fidelity (TSF), and responsive quality score (RQS). Through extensive evaluation of state-of-the-art MLLMs, we uncover a critical *fidelity-responsiveness paradox*: models achieving high visual fidelity tend to generate rigid, non-responsive code. We trace this phenomenon to a *metadata trap*, where models shortcut layout reasoning by directly transcribing absolute coordinates rather than generating fluid structures. Our data and code are available at https://anonymous.4open.science/r/FigmaBench-6C84/.

1 Introduction

The capability to automate the translation of visual designs into functional code is a pivotal aspect of modern software engineering. In industrial workflows, this process, known as design handoff, typically involves designers crafting high-fidelity user interfaces (UIs) using professional tools (e.g., Figma) which are subsequently implemented by front-end developers. While this labor-intensive phase has long been a target for automation, the rapid advancement of multimodal large language models (MLLMs) has recently sparked a surge of research into the design-to-code generation paradigm (Gui et al., 2025c; Yang et al., 2025; Zhao et al., 2025; Wu et al., 2025). To evaluate these capabilities, several benchmarks have been introduced (Xiao et al., 2025b; Gui et al., 2025a; Laurençon et al., 2024; Si et al., 2025), providing paired screenshots, reference HTML code, and corresponding evaluation metrics.

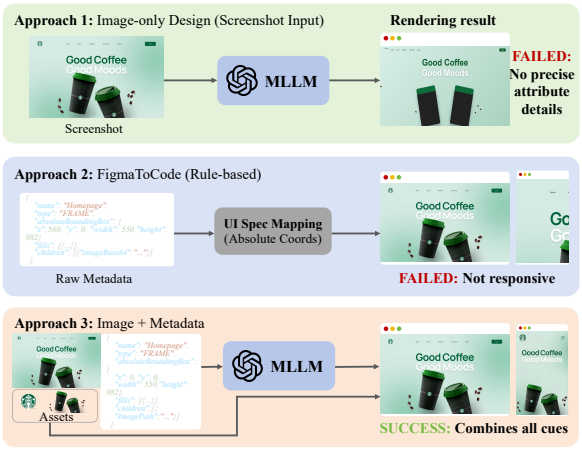

Figure 1: Comparison of input paradigms for design-to-code generation.

However, as illustrated in Figure 1, existing benchmarks predominantly adopt an image-to-code paradigm (Approach 1), utilizing rasterized screenshots as the sole input (Si et al., 2025). This fundamentally deviates from industrial workflows where professional design tools yield not only visual previews but also rich structured metadata: hierarchical DOM trees, precise coordinates, typography attributes, and asset references. Such metadata is indispensable for generating pixel-perfect, maintainable, and responsive code; by overlooking it, existing benchmarks fail to capture the full

complexity of real-world design handoff. Conversely, engineering-oriented tools (Approach 2) such as FigmaToCode (Wan et al., 2024b) do leverage Figma's structured JSON, yet they rely on direct attribute-to-HTML mapping without semantic reasoning. This enumeration-based approach produces rigid code that lacks responsiveness and exhibits poor maintainability.

Beyond input modality limitations, current evaluation paradigms also present notable challenges. Traditional methods employ code-level metrics to measure the similarity between generated and reference code (Gui et al., 2025a). Metrics such as html-BLEU (Soselia et al., 2023) and CodeBLEU (Ren et al., 2020) may be useful in certain contexts, but may not fully capture the equivalence of layout, visual style, functionality, or semantic identity. Identical interfaces can be realized through substantially different code structures. Although recent research has shifted towards visual similarity evaluation (Radford et al., 2021; Si et al., 2025) by leveraging perceptual metrics to compare rendered screenshots with reference designs, these high-level metrics often suffer from detecting subtle yet critical defects, such as text baseline misalignment, incorrect font weights. Additionally, standardized metrics for assessing responsive behaviors remain absent, despite responsiveness being a core requirement for mobile applications where interfaces must fluidly adapt to diverse screen dimensions.

To address these challenges, we introduce FigmaBench, the first benchmark providing both Figma structured JSON data and screenshots for multimodal design-to-code generation. Unlike existing benchmarks (detailed in Table 1), we collect raw design files from the Figma Community and apply a multi-stage pipeline to curate 1,234 high-quality samples, each comprising screenshots paired with complete JSON metadata. This dual-input paradigm faithfully replicates the professional design handoff process. To enable fine-grained assessment, we propose a multi-dimensional evaluation framework with four complementary metrics: visual consistency (VCS) for global semantic alignment, structural layout alignment (SLA) for geometric precision, textual and stylistic fidelity (TSF) for content accuracy, and responsive quality score (RQS) for adaptive layout behavior. Evaluating state-of-the-art (SOTA) MLLMs uncovers a *fidelity-responsiveness paradox*: models excelling in visual restoration often generate rigid, non-responsive code. This stems from a *metadata trap*, where models shortcut layout reasoning by transcribing absolute coordinates rather than generating fluid structures.

Table 1: Comparison of FigmaBench with other Design-to-Code datasets.

Dataset	Samples	Real World	Metadata	Assets
pix2code (Beltramelli, 2018)	1,742	✗	✗	✗
Design2Code (Si et al., 2025)	484	✓	✗	✗
MRWeb (Wan et al., 2024a)	500	✗	✗	✓
LaTCoder (Gui et al., 2025b)	128	✓	✗	✗
WebUIBench (Lin et al., 2025)	719	✓	✗	✗
WebCode2M (Gui et al., 2025a)	768	✓	✗	✗
FigmaBench (Ours)	1,234	✓	✓	✓

In summary, our contributions are as follows:

- We construct FigmaBench, the first benchmark providing both Figma structured JSON data and screenshots, comprising 1,234 diverse real-world UI design samples.
- we propose a multi-dimensional evaluation framework with four complementary metrics, enabling fine-grained assessment that addresses the limitations of current methods.
- Extensive evaluations of SOTA MLLMs reveal a critical *fidelity-responsiveness paradox*, where the root cause is a *metadata trap*. This offers actionable insights for future model development.

2 Background

2.1 Related Work

The task of automating UI code generation from visual designs has evolved significantly, starting from early deep learning approaches to recent advancements leveraging MLLMs (Chen et al., 2018; Gui et al., 2025c; Zhao et al., 2025; Li et al., 2025). Following Pix2Code (Beltramelli, 2018) and datasets like WebSight (Laurençon et al., 2024), the dominant paradigm remains Image-to-Code. Recent innovations, including Design2Code (Si et al., 2025), DCGen (Wan et al., 2024b), and LaTCoder (Gui et al., 2025b), have optimized generation via self-revision or modular strategies, with IW-bench (Guo et al., 2025) standardizing evaluation.However, relying solely on visual inputs presents inherent ambiguity, making it challenging to recover precise design attributes. While works like DeclarUI (Zhou et al., 2025), Interaction2Code (Xiao et al., 2025a), and MRWeb (Wan et al., 2024a) have extended synthesis to dynamic behaviors,direct input-level support for design primitives remains less explored.

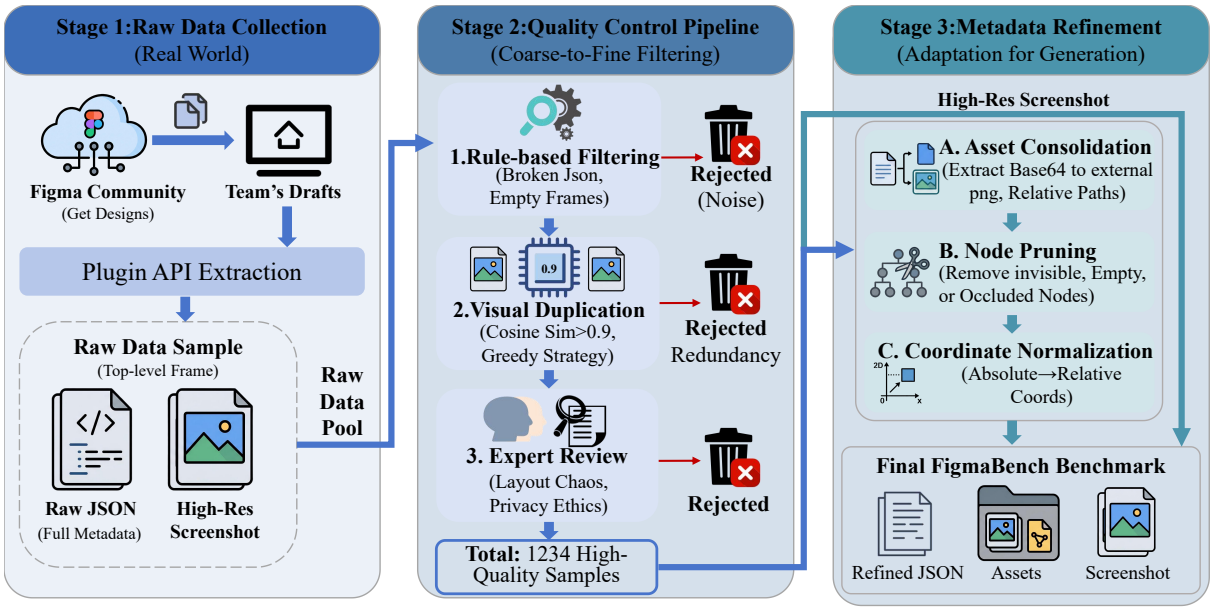

Figure 2: The data construction pipeline of FigmaBench, including raw data collection, quality control pipeline, and metadata refinement.

Since industrial design handoff relies on metadata beyond screenshots, we propose FigmaBench, integrating visual semantics with metadata to better simulate industrial workflows.

2.2 Problem Definition

Unlike previous approaches relying solely on visual screenshots (Si et al., 2025; Wu et al., 2023), our formulation incorporates both unstructured visual information and structured metadata as inputs. Formally, let a design sample be $\mathcal{D} = (I, \mathcal{J})$, where $I \in \mathbb{R}^{H \times W \times 3}$ is the high-resolution screenshot capturing global visual semantics, and $\mathcal{J}$ denotes the Figma JSON metadata containing hierarchical DOM structures and explicit attributes (e.g., coordinates, text content, asset URLs). The objective is to generate a code snippet C that maximizes the conditional probability:

$$C^* = \arg\max_{C} P(C \mid I, \mathcal{J}). \quad (1)$$

The generation is subject to two constraints: (1) rendering fidelity: at the original viewport, the rendered output $\mathcal{R}(C)$ must visually and geometrically align with the input screenshot I; and (2) responsive robustness: across varying viewports $\mathcal{V} = \{v_{\text{mobile}}, v_{\text{tablet}}, v_{\text{desktop}}\}$, the rendered results $\{\mathcal{R}(C, v)\}_{v \in \mathcal{V}}$ must maintain structural integrity and usability, rather than rigidly replicating pixel positions.

3 The FigmaBench

In this section, we introduce FigmaBench, an industrial-grade benchmark, to bridge the gap in evaluating multimodal design-to-code generation. Since identical visual effects can be achieved through diverse code implementations, and raw design files lack semantic reference code, we intentionally discard noisy code references. Instead, we employ screenshots as visual ground truth while leveraging structured Figma JSON for precise structural and stylistic verification. This design enables FigmaBench to comprehensively assess MLLMs' capability to translate visual intent into rendered reality, ensuring both pixel-level fidelity and attribute-level accuracy. The construction pipeline, illustrated in Figure 2, consists of three stages: raw data collection, multi-granular quality control, and generation-oriented metadata refinement. Further details are provided in Appendix A.

3.1 Raw Data Collection

Figma Community is a widely-used open platform hosting real-world UI/UX projects shared by professional designers and developers, exhibiting high ecological validity. We therefore select it as our primary data source. Leveraging the Figma Plugin API, we crawled 1,050 public design files spanning diverse topics, each containing varying numbers of design pages, totaling approximately 9,000 pages. We treat each top-level frame as an individual sample and collect two core components: (1) **full meta-**

data (JSON), containing hierarchical DOM structures, styling attributes, layout constraints, and embedded assets (e.g., Base64-encoded images and vector paths); and (2) **high-resolution screenshot**, serving as both model input and visual ground truth for evaluation. In total, 16,755 raw samples are initially collected, providing a solid foundation for subsequent quality filtering.

3.2 Quality Control

Since the data is crawled from the Figma Community, its quality varies considerably. For instance, some designs consist of merely a single image without structural elements, while designs within the same page often exhibit high redundancy (e.g., multiple frames representing UI/UX variations of the same screen). To curate high-quality and diverse samples, we design a coarse-to-fine, multi-stage filtering mechanism:

Rule-based Filtering. We parse JSON structures to remove low-quality samples, including: (1) corrupted files or broken assets; (2) empty structures lacking DOM nodes (e.g., single-image screenshots); and (3) outliers with extreme node counts ($> 1{,}000$) to fit model context windows.

Visual-Semantic De-duplication. To eliminate redundancy, we employ DINOv3 (Siméoni et al., 2025) to extract screenshot embeddings. We compute pairwise cosine similarities and apply a greedy strategy to remove duplicates exceeding a similarity threshold of 0.9, ensuring visual diversity.

Expert Review. Four senior front-end developers manually inspect the remaining samples. Exclusion criteria include disordered layouts, implicit structural redundancy, and content violating privacy or ethical standards.

Through this multi-stage pipeline progressing from heuristic rules to visual diversity enforcement to manual expert review, we curate a final set of 1,234 high-quality, diverse design samples.

3.3 Metadata Refinement

Raw Figma JSON fully preserves design specifications but contains redundant properties added to accommodate the Figma platform. Directly feeding such metadata into MLLMs introduces substantial noise that distracts the model from key attributes. To address this, we apply systematic refinement:

Asset Consolidation. Base64-encoded images and vector nodes consume substantial storage and cause context length explosion when directly input to MLLMs. We localize these assets by extracting them as external PNG/SVG files and replacing inline data with relative path references, significantly reducing token consumption while preserving rendering fidelity.

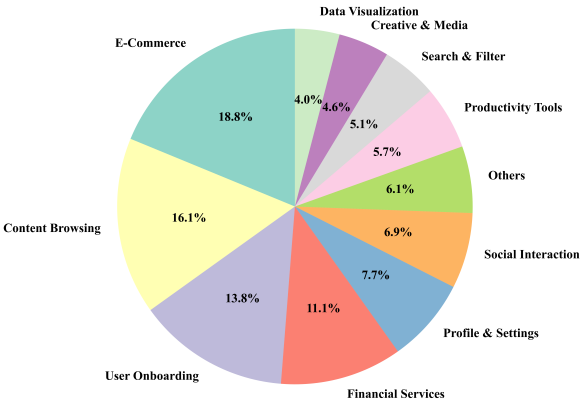

Figure 3: Distribution of domains across FigmaBench.

Node Pruning. We recursively prune the DOM tree to remove visually irrelevant nodes, including invisible layers, empty containers without styles, and fully occluded elements.

Coordinate Normalization. Raw Figma coordinates are absolute positions relative to the canvas rather than the design's root node. We convert all `absoluteBoundingBox` values to a root-relative coordinate system, positioning the root at the origin $(0, 0)$. This transformation facilitates the model's understanding of relative spatial relationships among UI elements.

3.4 Data Statistics and Analysis

As highlighted in Table 1, FigmaBench stands out among existing design-to-code benchmarks by being the only one sourced from Figma design files, incorporating both structured JSON metadata and associated assets (e.g., images and components). These elements are critical for generating production-grade, responsive UI code in industrial workflows, addressing limitations in prior image-only or synthetic datasets. To characterize the diversity and challenge level of FigmaBench, we analyze key complexity metrics via cumulative distribution functions (Figure 4). The designs span a wide range of structural complexity, with hierarchy depths reaching up to approximately 12 levels, node counts reaching up to 420 components

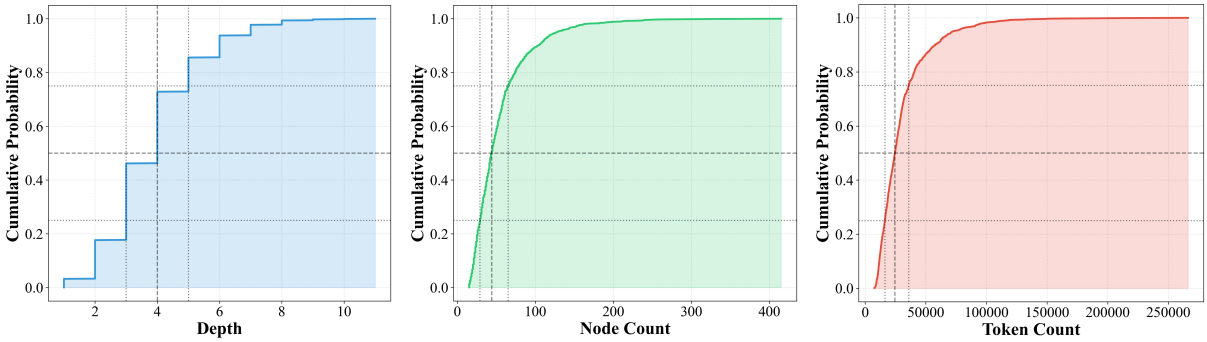

Figure 4: Cumulative distribution functions of key complexity metrics: design hierarchy depth, component node count, and Figma JSON token length.

in the most complex cases. These distributions capture realistic industry characteristics: from simpler layouts to highly intricate ones with deep nesting, dense component structures, and verbose metadata, thereby presenting substantial challenges for long-context understanding and robust multimodal reasoning. Furthermore, samples span 11 functional domains (Figure 3), with dominant categories including E-Commerce, Content Browsing, and User Onboarding. This composition mirrors prevalent real-world UI design scenarios, ensuring broad semantic diversity across transactional, informational, and interactive interfaces.

4 Experimental Setup

This section details our experimental framework for evaluating MLLMs on FigmaBench. We first introduce a multi-dimensional evaluation protocol designed to assess generation quality. Subsequently, we describe the evaluated models, followed by implementation details to ensure reproducibility. Please refer to Appendix B for more details.

4.1 Evaluation Metrics

To comprehensively evaluate generation quality, we construct a multi-dimensional framework covering high-level visual consistency, fine-grained structural alignment, textual fidelity, and responsive behavior. Crucially, a renderability check is enforced prior to calculation: if the code fails to parse or renders a blank page, all metrics are zeroed to strictly penalize non-functional outputs.

High-level Visual Consistency (VCS) To assess semantic consistency between the generated render (I_{gen}) and the reference design (I_{ref}), we employ the self-supervised vision model DINOv3 (Siméoni et al., 2025) as the feature extractor. Its robust feature space effectively captures global layout topology and color distribution. The score is defined as the cosine similarity:

$$\text{VCS} = \frac{v_{\text{gen}} \cdot v_{\text{ref}}}{\|v_{\text{gen}}\| \|v_{\text{ref}}\|}, \tag{2}$$

where $v_{\text{gen}}, v_{\text{ref}} \in \mathbb{R}^d$ denote the feature vectors extracted from I_{gen} and I_{ref}, respectively.

Structural Layout Alignment (SLA) Visual similarity alone fails to detect subtle component misalignments (e.g., text occlusion), which are fatal defects in complex interfaces. To quantify geometric precision, we model layout evaluation as a Minimum Weight Bipartite Matching problem. We parse reference elements $\mathcal{E}_{\text{ref}}$ from Figma JSON and extract generated DOM elements $\mathcal{E}_{\text{gen}}$ via a headless browser. We employ the Intersection over Union (IoU) metric to measure the geometric overlap between bounding boxes. A cost matrix is constructed with $C_{i,j} = 1 - \text{IoU}(b_i, b'_j)$. Using the Hungarian Algorithm (Kuhn, 1955), we obtain the optimal matching $\mathcal{M}$ and calculate:

$$\text{SLA} = \frac{\sum_{(i,j)\in\mathcal{M}} \text{IoU}(b_i, b'_j)}{\max(|\mathcal{E}_{\text{ref}}|, |\mathcal{E}_{\text{gen}}|)}. \tag{3}$$

By using $\max(|\mathcal{E}_{\text{ref}}|, |\mathcal{E}_{\text{gen}}|)$ as the denominator, SLA inherently penalizes both missing elements (under-generation) and hallucinated elements (over-generation), providing a balanced measure of structural fidelity.

Textual and Stylistic Fidelity (TSF) Since standard OCR metrics disregard DOM structure, we propose TSF to decouple content matching from style verification as follows:

(i) Content-Driven Alignment: To avoid geometric matching errors caused by minor rendering shifts, we prioritize semantic alignment using

Ratcliff-Obershelp string similarity. The optimal semantic matching set $\mathcal{M}_{\text{text}}$ is solved via the Hungarian Algorithm.

(ii) Style Verification: For each matched pair, we evaluate four typographic dimensions. To accommodate cross-browser rendering variations, font size matching allows a ± 1px tolerance, and color matching employs a perceptual similarity threshold (RGB cosine similarity ≥ 0.95). Font weight and family require exact matches due to their discrete/-categorical nature.

(iii) Hallucination-Aware Aggregation: MLLMs occasionally generate textual content absent in the reference design. To strictly penalize such hallucinations, we define the hallucination rate as

$$\gamma = \frac{\max(0, |\mathcal{T}_{\text{gen}}| - |\mathcal{M}_{\text{text}}|)}{|\mathcal{T}_{\text{gen}}|}, \tag{4}$$

where $|\mathcal{T}_{\text{gen}}|$ denotes the number of text nodes in the generated code, and $|\mathcal{M}_{\text{text}}|$ denotes the number of text elements present in the reference design metadata. The final score incorporates this penalty:

$$\text{TSF} = \left(\frac{\text{Cont} + \text{Style}}{2}\right) \times (1 - \gamma), \tag{5}$$

where Cont and Style represent the average content similarity and style similarity scores, respectively.

Responsive Quality Score (RQS) Responsiveness is a fundamental requirement for high-quality front-end code. However, existing metrics fail to capture dynamic responsive behaviors across viewports. To bridge this gap, we introduce the reference-free responsive quality score (RQS).

(i) Overflow Score (OVF): Preventing horizontal overflow within constrained viewports is critical. Therefore, we calculate the local defect rate R_{local} as the proportion of visible elements exhibiting layout errors (e.g., physical overflow or internal truncation), and derive $\text{OVF} = \max(0, 1 - R_{\text{local}})$.

(ii) Efficiency and Flexibility: We penalize inefficient space utilization (e.g., retaining a narrow mobile layout on a wide desktop screen) via the Efficiency score $\text{EFF} = \min(1.0, W_{\text{content}}/W_{\text{standard}})$, where $W_{\text{standard}} = 960$px follows common responsive design conventions. Simultaneously, to quantify structural flexibility, we define flexibility score FLX as the proportion of structural elements (e.g., `div`, `section`) exhibiting width changes exceeding 50px across viewports.

(iii) Aggregation: Given that layout overflow is a severe usability defect that renders other design qualities irrelevant, we treat OVF as a gating factor. The final metric is:

$$\text{RQS} = \text{OVF} \times \frac{\text{EFF} + \text{FLX}}{2}. \tag{6}$$

4.2 Base MLLMs

To comprehensively understand the performance of MLLMs on FigmaBench and identify the capability gap between open-source and proprietary models, we conducted extensive experiments on a diverse set of state-of-the-art models. We evaluated six proprietary models: GPT-4o (OpenAI et al., 2024), GPT-5 (OpenAI, 2025), Claude Sonnet 4 (Anthropic, 2025), Grok 4 Fast (xAI, 2025), Gemini 2.5 Flash (Google DeepMind, 2025a), and Gemini 3 Pro (Google DeepMind, 2025b). For open-source models, we assessed six variants across three leading model families to cover different parameter scales: Gemma3 (12B, 27B) (Gemma Team et al., 2025), InternVL3.5 (8B, 30B) (Wang et al., 2025), and Qwen3-VL (8B, 32B) (Bai et al., 2025).

4.3 Implementation

To ensure a rigorous comparison, we employed a unified prompting strategy across all experiments, querying all MLLMs with identical system instructions to eliminate prompt engineering variables. The full prompt templates used for different input modalities are detailed in Appendix B. Regarding computational infrastructure, proprietary models were accessed via their official APIs, while open-weights models were deployed locally on dual NVIDIA H100 GPUs (80GB VRAM each) to accommodate the extensive context windows required for processing complex metadata. To maximize reproducibility, the temperature parameter was set to 0 for all experiments. For the visual consistency score (VCS), we use DINOv3-ViT-B/16 as the feature extractor.

5 Evaluation

We evaluate MLLMs on FigmaBench through four perspectives: main performance, ablation study on input modalities, impact of design complexity, and empirical lessons derived from our findings.

5.1 Main Results

In this section, we report the quantitative performance of the evaluated MLLMs on FigmaBench. Table 2 summarizes the comprehensive results.

Table 2: Performance of MLLMs on FigmaBench using a unified prompting strategy.

Model	VCS	SLA	Cont	Style	TSF	OVF	EFF	FLX	RQS
Open-source									
Gemma3-12B	50.09	27.13	65.85	61.29	63.26	73.78	48.56	13.74	22.98
InternVL3.5-8B	55.58	17.62	65.01	44.13	52.16	**84.68**	**72.41**	**48.81**	**50.98**
Qwen3-VL-8B	51.16	29.85	66.31	59.53	60.40	83.13	46.42	34.81	33.85
Gemma3-27B	53.26	43.63	66.68	64.66	65.44	77.52	42.32	11.73	19.82
InternVL3.5-30B-A3B	53.44	21.76	68.77	48.45	55.92	82.51	65.18	37.39	42.54
Qwen3-VL-32B	55.19	36.66	69.56	60.33	61.50	80.01	44.21	33.09	31.29
Closed-source									
GPT-4o	66.14	35.01	76.08	72.84	73.85	71.56	49.94	6.12	18.02
GPT-5	75.89	72.04	93.98	83.38	87.11	75.16	49.79	1.62	17.10
Grok 4 Fast	72.53	48.55	91.30	78.91	81.65	73.48	50.68	2.60	17.55
Claude Sonnet 4	76.19	71.53	92.39	84.52	87.13	70.52	47.41	0.73	14.74
Gemini 2.5 Flash	74.77	67.04	91.79	84.09	84.79	71.78	51.84	1.58	16.89
Gemini 3 Pro	**77.43**	**75.79**	**95.01**	**84.85**	**87.52**	74.69	50.70	1.64	17.17

Global Visuals and Local Structure. Frontier models demonstrate strong capabilities in visual reconstruction, with Gemini 3 Pro achieving the highest visual consistency (VCS) of 77.43%. However, a consistent performance gap is observed between VCS and structural layout alignment (SLA) across all models. For instance, GPT-4o scores 66.14% in VCS but only 35.01% in SLA. This disparity indicates that while current MLLMs excel at capturing global color distributions and rough topology, they struggle with precise geometric alignment and fine-grained element positioning.

Text Content and Stylistic Accuracy. In terms of textual and stylistic fidelity (TSF), most proprietary models achieve scores above 80%, suggesting robust OCR capabilities. However, a deeper decomposition reveals an imbalance: content similarity (Cont) consistently outperforms style similarity (Style). Taking GPT-5 as an example, its content accuracy is at 93.98%, whereas style accuracy lags at 83.38%. This suggests that while models can accurately extract text, they frequently hallucinate or misinterpret specific typographic attributes such as font weight and exact color codes.

The Paradox of Responsiveness. A critical inverse correlation emerges between fidelity and robustness. Surprisingly, weaker models (e.g., InternVL3.5-8B) achieve the highest RQS (50.98%). Qualitative inspection reveals this is accidental robustness: unable to parse complex absolute coordinates, these models fallback to primitive HTML flow (generic `div` stacking), which is inherently fluid but visually unfaithful. In contrast, SOTA models "overfit" to the precise coordinates provided in metadata, resulting in rigid, pixel-perfect, yet brittle implementations.

5.2 Ablation Study: Input Modalities

To decouple the effects of visual inputs and structured metadata, we compared three settings on both GPT-4o and Gemma3-12b: image-only, metadata-only, and image + metadata (full). The results are presented in Table 3.

Necessity of Multimodal Inputs. The Metadata-Only setting yields poor performance across all metrics, as the JSON lacks visual context for rendering logic. Comparing the other two settings, the addition of metadata to the image-only baseline significantly boosts fidelity. For GPT-4o, visual consistency (VCS) improves by +18.46% and textual fidelity (TSF) by +32.04%. A similar trend is observed in Gemma3-12B. This confirms that visual encoders alone are insufficient for capturing precise design details, validating the advantage of the multimodal input paradigm in FigmaBench.

The "Metadata Trap". The ablation reveals the root cause of low responsiveness in the full setting. When metadata is removed (image-only), GPT-4o's RQS increases substantially from 18.02% to 51.61%, and Gemma3-12B's RQS increases from 22.98% to 50.51%. This drastic jump indicates that models possess the intrinsic capability to gen-

Table 3: Ablation study on the effect of image and metadata for GPT-4o and Gemma3-12B models.

Method	VCS	SLA	TSF	RQS
GPT-4o				
Image-Only	47.68	16.87	41.81	**51.61**
Metadata-Only	60.97	34.27	70.71	14.02
Image + Metadata	**66.14**	**35.01**	**73.85**	18.02
Gemma3-12B				
Image-Only	44.91	18.22	36.28	**50.51**
Metadata-Only	46.71	26.72	61.63	16.34
Image + Metadata	**50.09**	**27.13**	**63.26**	22.98

erate responsive layouts via visual reasoning. The degradation in the full setting confirms a "Metadata Trap": the presence of precise coordinate data induces models to shortcut layout reasoning, leading them to transcribe absolute positions rather than synthesizing fluid structures.

5.3 Impact of Design Complexity

We further analyzed model performance across samples with varying complexity using GPT-4o, defined by node count and hierarchy depth, as illustrated in Figure 5.

Fidelity Degradation. We observe a clear negative trend in fidelity metrics as design complexity increases along both dimensions. Specifically, VCS, SLA, and TSF all decline as node count and tree depth grow, indicating that maintaining visual and content accuracy becomes increasingly challenging in complex hierarchical designs. This validates FigmaBench as a challenging benchmark that effectively differentiates model capabilities in complex industrial scenarios.

Complexity-Induced Fallback. Contrary to the monotonic degradation in fidelity metrics, RQS exhibits an anomalous upward trend in high-complexity scenarios. This corroborates the bounds of the ***metadata trap***: when design complexity saturates the model's ability to map metadata to visuals, the *absolute positioning* strategy breaks down. Consequently, models are forced into a fallback mode, generating generic containers instead of precise coordinates. This creates a deceptive rise in RQS, further highlighting that responsiveness without fidelity is a failure mode, not a feature.

5.4 Empirical Findings

Based on the evaluation, we distill three critical lessons for future research:

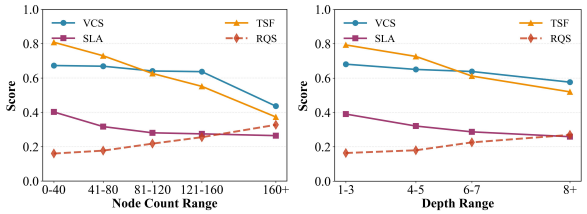

Figure 5: Model performance analysis across varying design complexities using GPT-4o.

Lesson 1: The Global-Local Perception Gap. Models excel at capturing overall visual appearance (high VCS) but struggle with geometric alignment (low SLA). Similarly, text content is accurately reproduced while typographic attributes are frequently misinterpreted. This suggests MLLMs prioritize salient, explicit information over subtle, implicit details.

Lesson 2: The Metadata Paradox. Metadata is indispensable for content fidelity but induces "cognitive offloading" where models shortcut layout reasoning by transcribing absolute coordinates. This paradox is further evidenced by the complexity analysis: as designs grow more complex, absolute positioning collapses, inadvertently yielding more fluid layouts. Resolving this requires decoupling content extraction from layout synthesis.

Lesson 3: Latent Responsive Capabilities. The high RQS in the image-only setting reveals that models already possess responsive knowledge, but this capability is suppressed by explicit coordinates. Future work can focus on inference-time strategies like coordinate masking or intermediate layout planning to activate these latent capabilities.

6 Conclusion

We introduce FigmaBench, the industrial-grade benchmark for multimodal design-to-code generation, comprising 1,234 high-quality samples. Our multi-dimensional evaluation framework enables fine-grained assessment beyond existing metrics. Through extensive experiments, we uncover the fidelity-responsiveness paradox: models achieving high visual fidelity tend to produce rigid, non-responsive code. We trace this to a metadata trap, where precise coordinates induce shortcut learning that bypasses layout reasoning. Our findings highlight the need for decoupling content extraction from layout synthesis and activating latent responsive capabilities in future MLLM development.

Limitation

While FigmaBench establishes a rigorous foundation for evaluating multimodal design-to-code generation, we identify specific areas for future expansion. First, we deliberately focus on generating standard HTML/CSS to assess the fundamental capability of visual-structural alignment. By excluding framework-specific logic (e.g., react, vue) and dynamic interactions in this version, we minimize framework-dependent noise, ensuring a cleaner evaluation of the model's core layout reasoning abilities. Finally, while our curated set of 1,234 samples ensures high-density metadata and strict quality control, it represents a high-quality evaluation set rather than a large-scale pre-training corpus.

Ethical Statement

Our research employs publicly available models and datasets with proper citations. This approach minimizes the risk of generating toxic content, leveraging the widely used and non-toxic nature of our datasets and prompts.

References

Anthropic. 2025. System card: Claude opus 4 & claude sonnet 4. https://www-cdn.anthropic.com/6be99a52cb68eb70eb9572b4cafad13df32ed995.pdf. Accessed: 2025-12-21.

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, Wenbin Ge, Zhifang Guo, Qidong Huang, Jie Huang, Fei Huang, Binyuan Hui, Shutong Jiang, Zhaohai Li, Mingsheng Li, and 45 others. 2025. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*.

Tony Beltramelli. 2018. pix2code: Generating code from a graphical user interface screenshot. In *Proceedings of the ACM SIGCHI symposium on engineering interactive computing systems*, pages 1–6.

Chunyang Chen, Ting Su, Guozhu Meng, Zhenchang Xing, and Yang Liu. 2018. From ui design image to gui skeleton: a neural machine translator to bootstrap mobile gui implementation. In *Proceedings of the 40th International Conference on Software Engineering*, pages 665–676.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, Louis Rouillard, Thomas Mesnard, Geoffrey Cideron, Jean bastien Grill, Sabela Ramos, Edouard Yvinec, Michelle Casbon, Etienne Pot, Ivo Penchev, and 197 others. 2025. Gemma 3 technical report. *Preprint*, arXiv:2503.19786.

Google DeepMind. 2025a. Gemini 2.5 pro model card. https://storage.googleapis.com/deepmind-media/Model-Cards/Gemini-2-5-Pro-Model-Card.pdf. Accessed: 2025-12-21.

Google DeepMind. 2025b. Gemini 3 pro model card. https://deepmind.google/models/model-cards/gemini-3-pro/. Accessed: 2025-12-21.

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. 2025a. Webcode2m: A real-world dataset for code generation from webpage designs. In *Proceedings of the ACM on Web Conference 2025*, WWW '25, pages 1834–1845, New York, NY, USA. Association for Computing Machinery.

Yi Gui, Zhen Li, Zhongyi Zhang, Guohao Wang, Tianpeng Lv, Gaoyang Jiang, Yi Liu, Dongping Chen, Yao Wan, Hongyu Zhang, Wenbin Jiang, Xuanhua Shi, and Hai Jin. 2025b. Latcoder: Converting webpage design to code with layout-as-thought. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, KDD '25, page 721–732, New York, NY, USA. Association for Computing Machinery.

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

Hongcheng Guo, Wei Zhang, Junhao Chen, Yaonan Gu, Jian Yang, Junjia Du, Shaosheng Cao, Binyuan Hui, Tianyu Liu, Jianxin Ma, Chang Zhou, and Zhoujun Li. 2025. IW-bench: Evaluating large multimodal models for converting image-to-web. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 6449–6466, Vienna, Austria. Association for Computational Linguistics.

Harold W Kuhn. 1955. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97.

Hugo Laurençon, Léo Tronchon, and Victor Sanh. 2024. Unlocking the conversion of web screenshots into html code with the websight dataset. *Preprint*, arXiv:2403.09029.

Yuhang Li, Chenchen Zhang, Ruilin Lv, Ao Liu, Ken Deng, Yuanxing Zhang, Jiaheng Liu, Wiggin Zhou, and Bo Zhou. 2025. Relook: Vision-grounded rl with a multimodal llm critic for agentic web coding. *arXiv preprint arXiv:2510.11498*.

Zhiyu Lin, Zhengda Zhou, Zhiyuan Zhao, Tianrui Wan, Yilun Ma, Junyu Gao, and Xuelong Li. 2025. Webuibench: A comprehensive benchmark for evaluating multimodal large language models in webui-to-code. *arXiv preprint arXiv:2506.07818*.

OpenAI. 2025. Gpt-5 system card. https://openai.com/index/gpt-5-system-card/. Accessed: 2025-12-21.

OpenAI, Aaron Hurst, Adam Lerer, Adam P. Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, Aleksander Mądry, Alex Baker-Whitcomb, Alex Beutel, Alex Borzunov, Alex Carney, Alex Chow, Alex Kirillov, Alex Nichol, and 400 others. 2024. Gpt-4o system card. *Preprint*, arXiv:2410.21276.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. 2021. Learning transferable visual models from natural language supervision. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 8748–8763. PMLR.

Shuo Ren, Daya Guo, Shuai Lu, Long Zhou, Shujie Liu, Duyu Tang, Neel Sundaresan, Ming Zhou, Ambrosio Blanco, and Shuai Ma. 2020. Codebleu: a method for automatic evaluation of code synthesis. *arXiv preprint arXiv:2009.10297*.

Chenglei Si, Yanzhe Zhang, Ryan Li, Zhengyuan Yang, Ruibo Liu, and Diyi Yang. 2025. Design2code: Benchmarking multimodal code generation for automated front-end engineering. In *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.

Oriane Siméoni, Huy V. Vo, Maximilian Seitzer, Federico Baldassarre, Maxime Oquab, Cijo Jose, Vasil Khalidov, Marc Szafraniec, Seungeun Yi, Michaël Ramamonjisoa, Francisco Massa, Daniel Haziza, Luca Wehrstedt, Jianyuan Wang, Timothée Darcet, Théo Moutakanni, Leonel Sentana, Claire Roberts, Andrea Vedaldi, and 7 others. 2025. Dinov3. *Preprint*, arXiv:2508.10104.

Davit Soselia, Khalid Saifullah, and Tianyi Zhou. 2023. Learning ui-to-code reverse generator using visual critic without rendering. *arXiv preprint arXiv:2305.14637*.

Yuxuan Wan, Yi Dong, Jingyu Xiao, Yintong Huo, Wenxuan Wang, and Michael R Lyu. 2024a. Mrweb: An exploration of generating multi-page resource-aware web code from ui designs. *arXiv preprint arXiv:2412.15310*.

Yuxuan Wan, Chaozheng Wang, Yi Dong, Wenxuan Wang, Shuqing Li, Yintong Huo, and Michael R Lyu. 2024b. Automatically generating ui code from screenshot: A divide-and-conquer-based approach. *arXiv preprint arXiv:2406.16386*.

Weiyun Wang, Zhangwei Gao, Lixin Gu, Hengjun Pu, Long Cui, Xingguang Wei, Zhaoyang Liu, Linglin Jing, Shenglong Ye, Jie Shao, Zhaokai Wang, Zhe Chen, Hongjie Zhang, Ganlin Yang, Haomin Wang, Qi Wei, Jinhui Yin, Wenhao Li, Erfei Cui, and 56 others. 2025. Internvl3.5: Advancing open-source multimodal models in versatility, reasoning, and efficiency. *Preprint*, arXiv:2508.18265.

Fan Wu, Cuiyun Gao, Shuqing Li, Xin-Cheng Wen, and Qing Liao. 2025. Mllm-based ui2code automation guided by ui layout information. *Proceedings of the ACM on Software Engineering*, 2(ISSTA):1123–1145.

Jason Wu, Siyan Wang, Siman Shen, Yi-Hao Peng, Jeffrey Nichols, and Jeffrey P Bigham. 2023. Webui: A dataset for enhancing visual ui understanding with web semantics. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, pages 1–14.

xAI. 2025. Grok 4 model card. https://data.x.ai/2025-08-20-grok-4-model-card.pdf. Accessed: 2025-12-21.

Jingyu Xiao, Yuxuan Wan, Yintong Huo, Zixin Wang, Xinyi Xu, Wenxuan Wang, Zhiyao Xu, Yuhang Wang, and Michael R. Lyu. 2025a. Interaction2code: Benchmarking mllm-based interactive webpage code generation from interactive prototyping. *arXiv preprint arXiv:2411.03292*.

Jingyu Xiao, Ming Wang, Man Ho Lam, Yuxuan Wan, Junliang Liu, Yintong Huo, and Michael R Lyu. 2025b. Designbench: A comprehensive benchmark for mllm-based front-end code generation. *arXiv preprint arXiv:2506.06251*.

Yiying Yang, Wei Cheng, Sijin Chen, Xianfang Zeng, Fukun Yin, Jiaxu Zhang, Liao Wang, Gang Yu, Xingjun Ma, and Yu-Gang Jiang. 2025. Omnisvg: A unified scalable vector graphics generation model. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Xuanle Zhao, Deyang Jiang, Zhixiong Zeng, Lei Chen, Haibo Qiu, Jing Huang, Yufeng Zhong, Liming Zheng, Yilin Cao, and Lin Ma. 2025. Vincicoder: Unifying multimodal code generation via coarse-to-fine visual reinforcement learning. *arXiv preprint arXiv:2511.00391*.

Ting Zhou, Yanjie Zhao, Xinyi Hou, Xiaoyu Sun, Kai Chen, and Haoyu Wang. 2025. Declarui: Bridging design and development with automated declarative ui code generation. *Proceedings of the ACM on Software Engineering*, 2(FSE):219–241.

A Data Construction Pipeline Details

This appendix provides a detailed description of our data construction pipeline, including the raw data collection implementation and metadata refinement process.

A.1 Raw Data Collection Implementation

To systematically extract design data from Figma Community, we utilize the Figma Plugin API for batch traversal and export.

A.1.1 Document Traversal

Figma organizes designs in a hierarchical structure: `Document` → `Pages` → `Frames`. Our plugin traverses this hierarchy using the following API calls, as shown in Figure 6.

```
#Step 1: Get all pages in the document
const pages = figma.root.children.map(page
=> ({
  id: page.id,
  name: page.name
}));

#Step 2: For each page, get top-level frames
const frames = page.children.filter(child =>
  child.type === 'FRAME'
);
```

Figure 6: Figma API traversal for hierarchical design structures.

We treat each top-level `FRAME` node as an individual design sample. This granularity aligns with how designers typically organize UI screens in Figma.

A.1.2 Data Export

For each frame, we extract two components:

(1) Full Metadata (JSON): We recursively export the complete node tree, preserving all styling attributes, layout constraints, and hierarchical relationships, as shown in Figure 7.

(2) High-Resolution Screenshot: We use the exportAsync API to render each frame as a PNG image, as shown in Figure 8.

The scale factor of 2.0 ensures high-fidelity screenshots suitable for both visual evaluation and model input.

A.1.3 Batch Processing

To handle large-scale collection (1,050 design files, ∼9,000 pages), we implemented several optimizations:

```
// Export node with all properties
const exportNode = (node) => ({
  id: node.id,
  name: node.name,
  type: node.type,
  absoluteBoundingBox:
node.absoluteBoundingBox,
  fills: node.fills,
  strokes: node.strokes,
  effects: node.effects,
  children: node.children?.map(exportNode),
  // ... additional properties
});
```

Figure 7: Recursive export of the complete Figma node tree with full metadata.

```
// Export frame as PNG
const bytes = await frame.exportAsync({
  format: 'PNG',
  constraint: { type: 'SCALE', value: 2.0 }
});
```

Figure 8: Recursive export of the complete Figma node tree with full metadata.

- **Progress Tracking:** Three-level progress monitoring (Document → Page → Frame) with checkpoint support for resumable collection.
- **Memory Management:** Stream processing with explicit garbage collection after each frame to prevent memory overflow.
- **Pre-filtering:** Early rejection of frames exceeding complexity thresholds (node count > 1,000) before full export.

A.2 Rule-based Filtering: Detailed Criteria

Our rule-based filtering removes samples that fail to meet minimum quality standards. Table 4 summarizes the specific criteria and their rationale.

A.3 Metadata Refinement: Implementation Details

The metadata refinement process transforms raw Figma JSON into a clean, generation-ready format. We describe each step with concrete examples.

A.3.1 Asset Consolidation

Raw Figma JSON embeds images as Base64 strings directly in the metadata, causing severe

Table 4: Rule-based filtering criteria applied during quality control.

Rule	Criterion	Rationale
Corrupted Files	Missing/broken assets	Cannot be rendered correctly
Empty Structure	Zero DOM nodes	Single-image with no elements
Extreme Complexity	Node count $>$ 1,000	Exceeds model context limits
Missing BBox	`absoluteBoundingBox` is null	Abnormal node, cannot compute layout
Invisible Nodes	`visible=false` or `opacity=0`	No visual contribution

context length explosion. For example, a single 200×200 PNG image encoded in Base64 consumes approximately 50,000 characters—equivalent to roughly 12,500 tokens.

Process: We extract all Base64-encoded images and SVG content, save them as external files, and replace the embedded data with relative path references as shown in Figure 9.

```
#Before refinement
{
 "type": "RECTANGLE",
 "imageBase64": "...",
 "imageHash": "abc123..."
}

#After refinement
{
 "type": "RECTANGLE",
 "imagePath": "extracted_images/img_a1b2c3.png"
}
```

Figure 9: The process of extracting embedded Base64 data.

A.3.2 Node Pruning

We recursively traverse the DOM tree and remove nodes that have no visual contribution to the final rendering, as illustrated in Figure 10. The pruning algorithm considers multiple factors:

(1) Visibility Check: Remove nodes with `visible=false` or `opacity=0`.

(2) Empty Container Removal: Remove GROUP/FRAME nodes that contain no children and have no styling properties (fills, strokes, effects).

(3) Out-of-Bounds Detection: Remove child nodes whose bounding boxes are completely outside their parent container's bounds.

(4) Occlusion Detection: Remove nodes that are completely covered by opaque sibling nodes rendered above them. This requires checking:

- The upper node has an opaque solid fill (`opacity=1`, `blendMode=NORMAL`)
- The upper node's bounding box completely covers the lower node
- Neither node has rotation applied

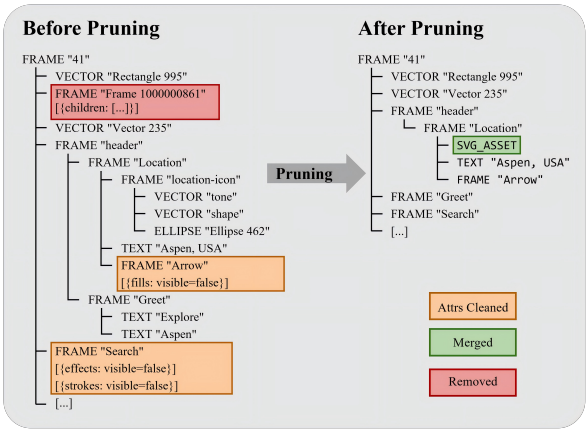

Figure 10: Node pruning case study. Red nodes are removed due to invisibility or occlusion.

A.3.3 Coordinate Normalization

Raw Figma coordinates use absolute canvas positions, which can result in large arbitrary values (e.g., $x = 3847, y = 2156$). We normalize all coordinates relative to the root node, as illustrated in Figure 11.

```
#Before: absolute canvas coordinates
{
 "absoluteBoundingBox": {"x": 3847.0, "y": 2156.0,
  "width": 375.0, "height": 812.0
 }
}

#After: normalized to root origin (0,0)
{
 "absoluteBoundingBox": {"x": 0.0, "y": 0.0,
  "width": 375.0, "height": 812.0
 }
}
```

Figure 11: Coordinate normalization process.

Additionally, we round all floating-point values

to 3 decimal places to reduce token consumption without sacrificing precision.

A.3.4 Attribute Filtering

We remove editor-specific attributes that are irrelevant to code generation, as shown in Table 5.

Table 5: Editor-specific attributes removed during refinement.

Removed Attribute	Reason
`locked`	Editor state, not visual
`exportSettings`	Export configuration
`background`	Deprecated, replaced by fills
`backgroundColor`	Deprecated

A.3.5 SVG Asset Merging

Vector graphics nodes (VECTOR, LINE, ELLIPSE, etc.) are often decomposed into multiple primitives in Figma. We identify groups of vector nodes and merge them into single SVG_ASSET nodes:

Merge Criteria:

- Single vector node types: VECTOR, STAR, LINE, ELLIPSE, REGULAR_POLYGON, BOOLEAN_OPERATION
- GROUP/FRAME containing only vector children
- Nodes with names containing "merge" keyword

A.4 Quality Control: Case Studies

A.4.1 Case 1: Single-Image Designs

Some Figma designs consist of a single rasterized image without any structured elements. These provide no value for evaluating design-to-code generation capabilities, as shown in Figure 12.

```
// Rejected: empty structure
{
 "type": "FRAME",
 "children": [{
  "type": "RECTANGLE",
  "fills": [{"type": "IMAGE", ...}]
 }]
}
```

Figure 12: Example of a rejected single-image design structure.

A.4.2 Case 2: Visual Deduplication

Using DINOv3 embeddings, we identify and remove near-duplicate designs. Figure 13 shows an example where multiple color variations of the same UI layout are reduced to a single representative sample.

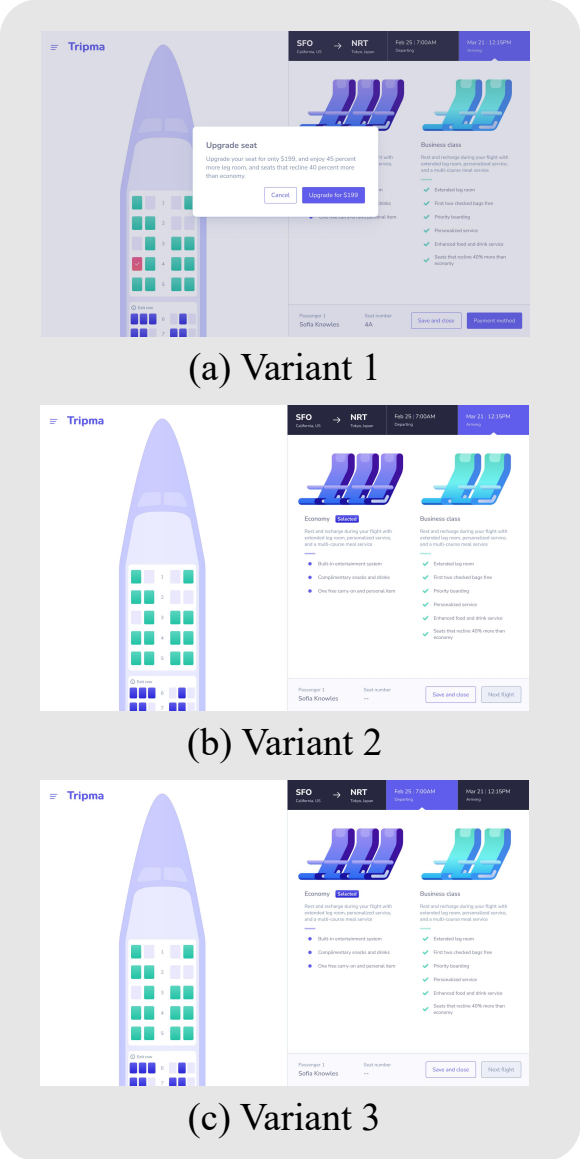

Figure 13: Visual deduplication example. Multiple color variations (similarity > 0.9) are reduced to one representative sample.

A.5 Processing Statistics

Table 6 summarizes the sample counts at each pipeline stage.

A.6 Refinement Impact Analysis

To quantify the effectiveness of metadata refinement, we measured the token reduction achieved by each operation, as shown in Table 7.

Table 6: Sample counts at each stage of the quality control pipeline.

Stage	Samples	Removed
Raw Collection	16,755	—
After Rule-based Filtering	8,432	8,323
After Visual Deduplication	2,156	6,276
After Expert Review	1,234	922

Table 7: Average token reduction achieved by each refinement step.

Refinement Step	Avg. Token Reduction
Asset Consolidation	67.3%
Node Pruning	12.8%
Attribute Filtering	8.4%
Coordinate Normalization	3.2%
Total Reduction	**74.6%**

The most significant reduction comes from asset consolidation, which replaces lengthy Base64 strings with short file paths. This transformation is critical for enabling MLLMs to process complex designs within their context limits.

B Experimental Details

This section provides supplementary experimental details, including the evaluation metrics, prompt templates and test result demonstrations.

B.1 Evaluation Metric Details

RQS is a reference-free metric assessing layout robustness independently. However, high RQS with low fidelity (VCS/SLA) may indicate *accidental fluidity* from primitive fallback layouts rather than genuine responsive design. Thus, RQS should be interpreted jointly with fidelity metrics.

Note on StyleMatch: Font size matching allows $\pm$1px tolerance; color matching uses RGB cosine similarity ≥ 0.95; font weight and family require exact matches.

RQS Hyperparameters and Design Rationale. RQS computation uses the following thresholds derived from established responsive design conventions: viewports $\mathcal{V} = \{375, 768, 1920\}$ represent standard mobile, tablet, and desktop breakpoints; $\epsilon_{overflow} = 1$px provides sub-pixel tolerance for overflow detection to accommodate browser rounding; $W_{standard} = 960$px follows common max-width conventions for centered content containers (Bootstrap/Foundation default); $\tau_{adapt} = 50$px sets the minimum width change to qualify as significant adaptation, filtering noise from minor reflows; and $W_{min} = 50$px excludes icons and micro-elements from FLX consideration.

Overflow Detection Criteria. An element is flagged as overflowing if: (1) its bounding box extends beyond the viewport boundary ($x + w > W_{viewport} + \epsilon$), or (2) its internal scroll width exceeds client width while using `overflow: hidden`, causing content truncation. Semantic scrollers (e.g., `<table>`, `<pre>`, flex containers with `nowrap`) are excluded as intentional design patterns.

Limitation on Intentional Whitespace. EFF penalizes layouts that underutilize desktop width. However, some designs intentionally employ generous whitespace for aesthetic purposes (e.g., minimalist landing pages). In such cases, EFF may underestimate design quality. We acknowledge this as a limitation inherent to reference-free metrics and recommend interpreting EFF jointly with VCS for designs where intentional sparsity is suspected.

B.2 Human Evaluation of RQS

To validate the alignment between RQS and expert judgment on responsive quality, we conducted a human evaluation study.

Sample Selection. We selected 150 generated pages from the test set, covering 3 representative models: InternVL3.5-8B, Qwen3-VL-32B, and GPT-5. For each model, we sampled 50 designs balanced across low/medium/high complexity tiers based on node count.

Annotators. We recruited 3 front-end developers (denoted as E1, E2, E3) with 3+ years of professional experience in responsive web development. Each annotator independently evaluated all 150 samples.

Evaluation Protocol. Annotators viewed the rendered pages across three viewport widths (375px, 768px, 1920px) using a standardized browser environment. For each sample, they rated the overall responsive quality on a 5-point Likert scale (1=very poor, 5=excellent), considering overflow behavior, space utilization, and layout adaptability.

Results. Table 8 presents the human evaluation results. Note that the RQS values reported here are computed on the 50-sample subset used for human evaluation, which may slightly differ from the full-set results in Table 2 due to sampling variance. Expert ratings align with RQS trends: InternVL3.5-8B receives the highest scores while GPT-5 scores lowest despite superior fidelity. The strong correlation ($\rho = 0.69$, $p < 0.001$) and high inter-annotator agreement ($\alpha = 0.72$) validate RQS as an effective

proxy for expert judgment.

Table 8: Human evaluation results on sampled subset.

Model	Examples	Human Avg.	RQS
InternVL3.5-8B	50	2.68	52.13
Qwen3-VL-32B	50	2.05	30.47
GPT-5	50	1.41	16.82

Statistics: Spearman $\rho = 0.69$ ($p < 0.001$)
Agreement: Krippendorff's $\alpha = 0.72$

B.3 Prompt Templates

To ensure a fair comparison, all evaluated models receive identical prompts for the same input modality configuration. This subsection presents the exact prompt templates used in our experiments.

B.3.1 Image + Metadata Setting

The following prompt is used when both the *image-metadata* is provided :

Prompt: Image + Metadata

As a professional front-end developer, please generate a complete HTML document using HTML and CSS based on the provided Figma JSON data and the corresponding UI page screenshot.

Input Data:
- **Figma JSON:** Contains detailed information such as layout, dimensions, colors, typography, and asset bindings. This is the primary source of truth.
- **Screenshot:** The corresponding UI screenshot serves only as supplementary reference to clarify unclear details.
- **Assets:** The original Figma JSON entries with "type: IMAGE" and "type: SVG_ASSET" have been replaced with actual local paths (e.g., "imagePath": "extracted_images img_690fecddec7845ccaa2f42f4972ecfb0.png"). Therefore, for images and SVG resources, please directly use the corresponding local paths provided in the JSON.

Important Implementation Requirements:
1. **Coordinate System:** All coordinate information in the Figma JSON is relative to the root node. The `absoluteBoundingBox` values have been pre-processed to be relative coordinates.
2. **Image Resource Loading:** For nodes with `imagePath` and `svgPath`, precisely reference the provided path.
3. **CSS Style:** The "styles" field in the Figma JSON contains various style information (e.g., font size, color, font weight, etc.).
4. **Output:** Return only the complete HTML document. Do not wrap the output in Markdown code fences. Do not include any explanation or extra text.

B.3.2 Metadata-Only Setting

The following prompt is used when only *metadata* is provided:

Prompt: Metadata-Only

As a professional front-end developer, please generate a complete HTML document using HTML and CSS based on the provided Figma JSON data.

Input Data:
- **Figma JSON:** Contains detailed information such as layout, dimensions, colors, typography, and asset bindings. This is the primary source of truth.
- **Assets:** The original Figma JSON entries with "type: IMAGE" and "type: SVG_ASSET" have been replaced with actual local paths (e.g., "imagePath": "extracted_images img_690fecddec7845ccaa2f42f4972ecfb0.png"). Therefore, for images and SVG resources, please directly use the corresponding local paths provided in the JSON.

Important Implementation Requirements:
1. **Coordinate System:** All coordinate information in the Figma JSON is relative to the root node. The `absoluteBoundingBox` values have been pre-processed to be relative coordinates.
2. **Image Resource Loading:** For nodes with `imagePath` and `svgPath`, precisely reference the provided path.
3. **CSS Style:** The "styles" field in the Figma JSON contains various style information (e.g., font size, color, font weight, etc.).
4. **Output:** Return only the complete HTML document. Do not wrap the output in Markdown code fences. Do not include any explanation or extra text.

B.3.3 Image-Only Setting

The following prompt is used when only the *image* is provided:

Prompt: Image-Only

As a professional front-end developer, please generate a complete HTML document using HTML and CSS based on the provided screenshot.

Important Implementation Requirements:
1. Return a single HTML file that uses HTML and CSS to reproduce the given screenshot. Include all CSS code in the HTML file itself.
2. Pay attention to size, text, position, and color of all elements, as well as the overall layout.
3. Return only the complete HTML document. Do not wrap the output in Markdown code fences. Do not include any explanation or extra text.

B.4 Test Result Demonstrations

We present examples from GPT-4o and Gemma3-12B models under different input modality settings (Figure 14, Figure 15, Figure 16). Both Metadata-Only and Image-Only settings show poor reconstruction quality, demonstrating the advantage of multimodal input paradigm.

C Potential Risks

While automated design-to-code generation can improve developer productivity, we identify the following potential risks:

Code Quality Concerns. Generated code may contain accessibility issues (e.g., missing ARIA labels, poor contrast ratios) or security vulnerabilities (e.g., XSS-prone patterns) if not properly reviewed. Users of code generation systems should always conduct thorough testing and code review before deployment.

Workforce Impact. Widespread adoption of automated UI code generation tools could potentially displace entry-level front-end development positions. We encourage responsible deployment with appropriate workforce transition support.

Benchmark Misuse. FigmaBench is intended for research evaluation purposes. Using benchmark scores as the sole criterion for production tool selection without considering real-world requirements may lead to suboptimal decisions.

D Ethical Considerations

Data Collection and Privacy. All design samples in FigmaBench are collected from Figma Community, a public platform where designers voluntarily share their work under Creative Commons or similar open licenses. By publishing on Figma Community, designers implicitly consent to public use of their designs. We do not collect any personally identifiable information (PII) from designers or users. The collected designs are used solely for research purposes.

PII and Offensive Content Check. We conducted a systematic review of collected samples to identify and remove any designs containing: (1) personally identifiable information such as real names, email addresses, phone numbers, or profile photos; (2) potentially offensive, discriminatory, or inappropriate content. This review was performed both algorithmically (using keyword filtering) and manually (expert inspection of flagged samples). Samples failing these checks were excluded from the final benchmark.

E Reproducibility

We are committed to ensuring the reproducibility of our research:

Data Availability. The complete FigmaBench dataset, including all design samples, metadata, and reference screenshots, will be publicly released upon publication.

Code Release. All evaluation scripts, metric implementations, and data processing pipelines are available in our open-source repository. This includes the automated evaluation framework and visualization tools.

Experimental Configuration. Model configurations and hyperparameters are detailed in Section 4.3. Prompt templates are provided in Appendix B.3.

Result Reporting. All metrics are single-run results with deterministic decoding (temperature=0). We report mean values across all test samples.

F Human Annotations

We recruit three professional front-end developers with at least three years of responsive web development experience to manually evaluate the generated pages in our human study. The participants are compensated at a rate of approximately $100 per hour, which is consistent with common standards for remote data annotation. This payment rate is considered fair given the participants' demographic and their expertise in front-end development. All annotators provided informed consent and were informed that their anonymized ratings would be used for research purposes. This study involves evaluation of system outputs by expert annotators and does not constitute human subjects research under standard IRB definitions; the evaluation task posed no risks to participants beyond normal professional activities.

G AI Assistants in Research and Writing

Yes, we did utilize AI assistants in certain aspects of our research and writing process. Specifically, we employed generative AI tools, such as ChatGPT, to assist with writing portions of the Python code and in drafting parts of the appendix, as well as for polishing and refining sections of the paper. The AI tools were particularly helpful for enhancing clarity, improving grammatical structure, and ensuring a more concise presentation of our ideas.

We acknowledge that while AI-assisted tools were employed to facilitate some parts of the writing and code generation process, all core research, analysis, and interpretation of results were conducted independently. The use of AI tools was limited to supporting tasks that did not impact the integrity or originality of the research. Addition-

ally, we ensured that the final content was carefully reviewed and verified to maintain academic rigor and accuracy.

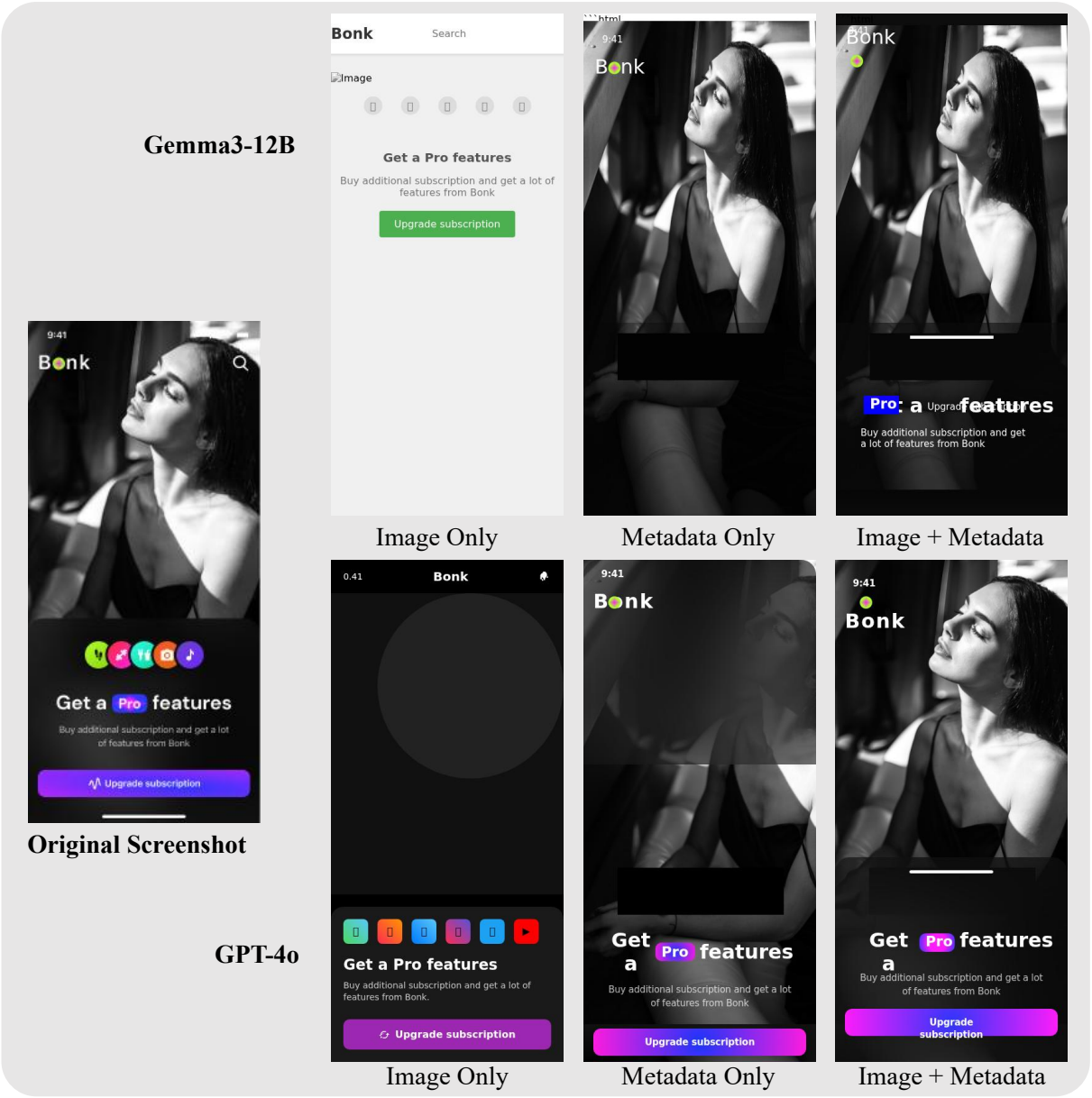

Figure 14: Case Study 1.

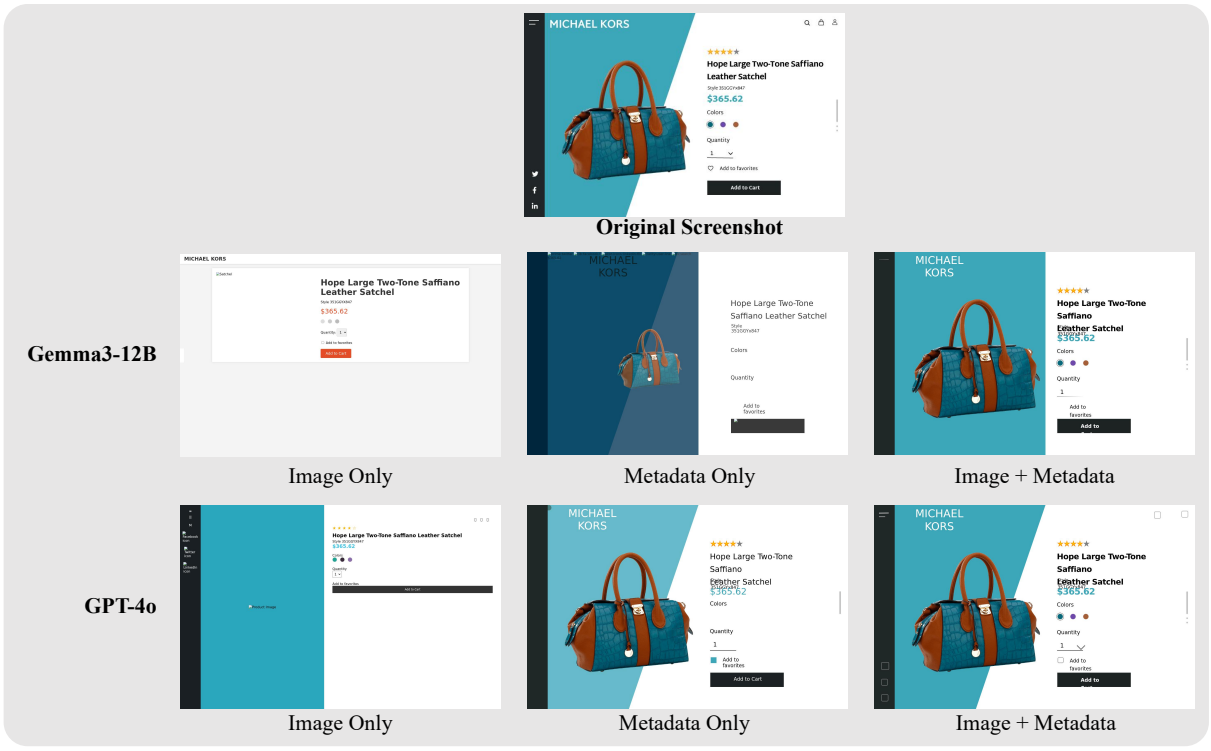

Figure 15: Case Study 2.

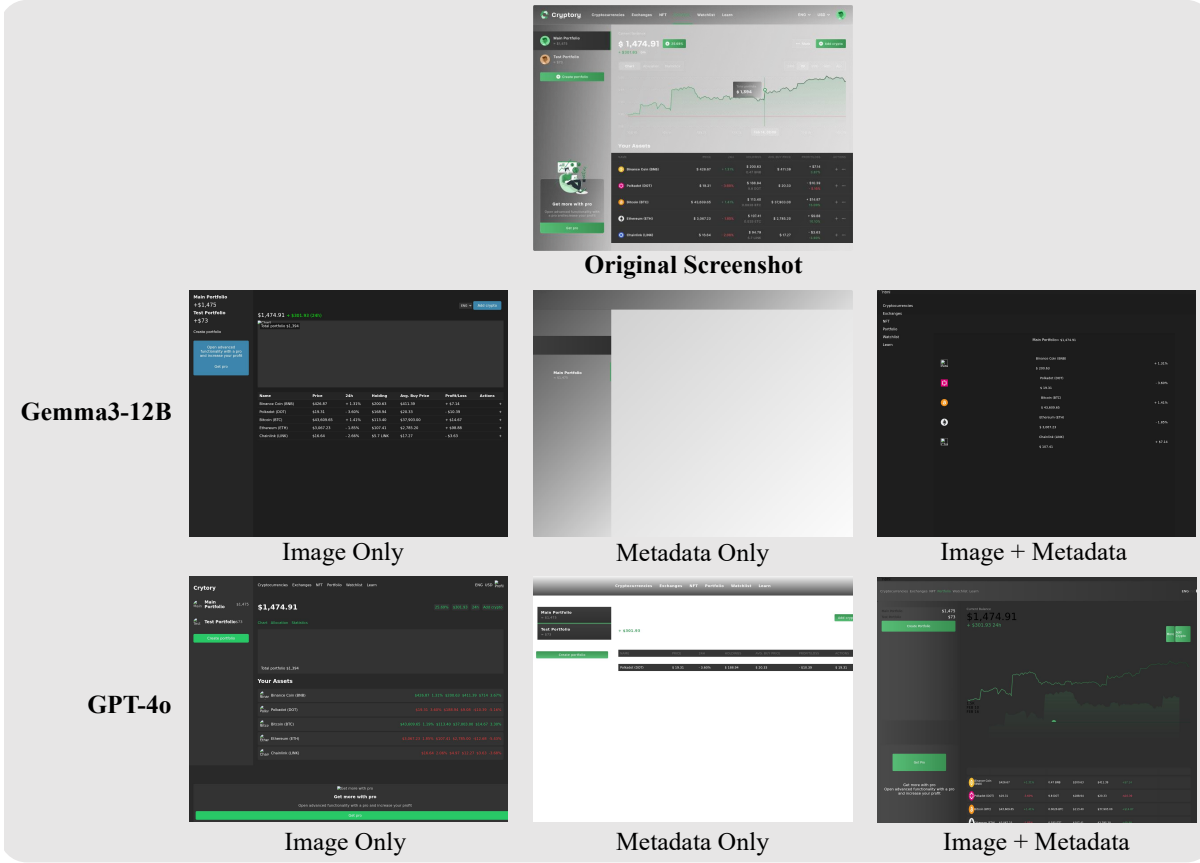

Figure 16: Case Study 3.