Pattern over Pixels: Measuring Pattern Completion Bias in Multimodal Code Generation

Khai-Nguyen Nguyen[✉*]
William & Mary
Williamsburg, USA
rbc5xp@virginia.edu

Oscar Chaparro
William & Mary
Williamsburg, USA
oscarch@wm.edu

Antonio Mastropaolo
William & Mary
Williamsburg, USA
amastropaolo@wm.edu

Abstract

Multimodal large language models (MLLMs) are increasingly used to translate webpage screenshots into front-end code, but repeated UI patterns may sway them toward visually incorrect yet pattern-consistent outputs. In this work, we test how repeated webpage patterns hurt MLLM accuracy on an objective screenshot-to-code fill-in-the-blank task. We introduce the first benchmark for *visual pattern-completion bias*, where one localized element in a repeated UI pattern is perturbed and the model must recover the masked width or font-size value from the screenshot and HTML context. Starting from 30 webpages curated from the DESIGN2CODE dataset, we build 1,440 evaluated screenshots spanning structural card and text-style patterns under standard and noise-overlaid conditions. We evaluate five frontier MLLMs and find that all are strongly biased toward the repeated baseline. Mean bias rate reaches 69.78% on card-width perturbations and 80.22% on text font-size perturbations, while mean accuracy is only 21.17% and 7.89%, respectively. `Codex-5.3` performs best but still drops from 68.61% accuracy on cards to 13.89% on text, while `Flash-3.0` reaches 96.11% bias on text. Noise, subtler perturbations, and boundary positions further increase bias rate. Reasoning analysis further shows that greater reasoning effort correlates with lower bias, yet qualitative evidence reveals that models can identify the anomalous element and still override it with the pattern-consistent answer. Our results identify a concrete failure mode in multimodal code generation and show that its severity is strongly associated with visual saliency.

Project Page Code & Artifact Dataset

CCS Concepts

• **Software and its engineering** → **Software development methods**; **Empirical software validation**; • **Computing methodologies** → **Machine learning**.

Keywords

Large Language Models for Code, Code Generation, Bias

ACM Reference Format:
Khai-Nguyen Nguyen, Oscar Chaparro, and Antonio Mastropaolo. 2026. Pattern over Pixels: Measuring Pattern Completion Bias in Multimodal Code Generation. In *Proceedings of the 41st IEEE/ACM International Conference on Automated Software Engineering (ASE '26), October 12–16, 2026, Munich, Germany*. ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3832783.3834443

*Now at the University of Virginia. This work was done at William & Mary.

ASE '26, Munich, Germany

ACM ISBN 979-8-4007-2882-2/2026/10
https://doi.org/10.1145/3832783.3834443

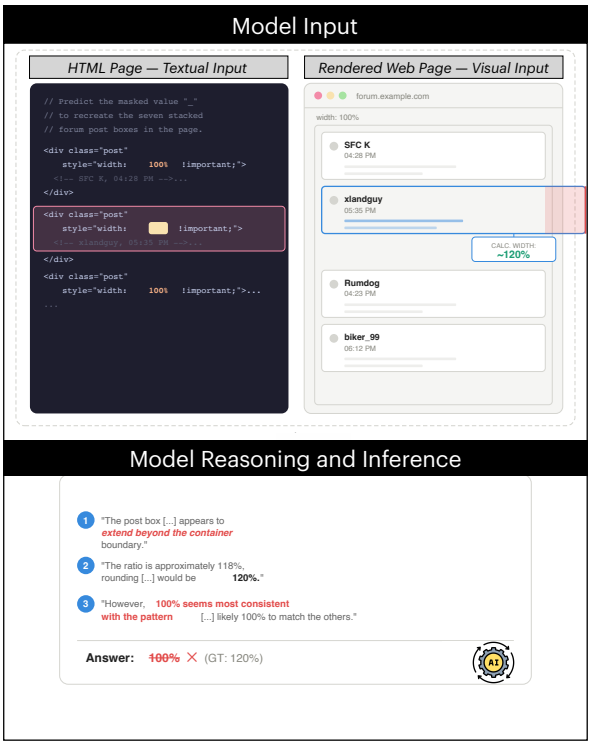


Figure 1: Visual pattern-completion bias in action. Given an HTML screenshot and its masked HTML snippet where one post card has `width:_` and all others `width:100%`, `Sonnet-4.6` observes the card extends beyond its container, computes the correct ratio (≈118% → 120%), then discards its own calculation to "match the others."

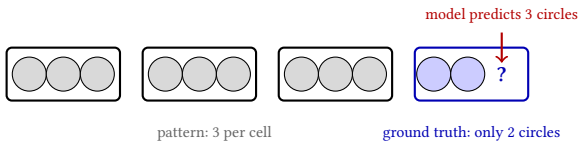
1 Introduction

Large language models (LLMs) have transformed software engineering workflows, from code completion and debugging to automated refactoring and test generation [8, 14, 40, 41, 43]. Recently, multimodal large language models (MLLMs) have extended LLMs capability to the visual domain, enabling the translation of webpage screenshots directly into executable front-end code [17, 18, 47]. As

these systems have grown more capable, making screenshot-to-code a practical workflow for prototyping, design handoff, and front-end implementation, the research community has responded with benchmarks that assess end-to-end generation quality across increasingly complex web interfaces [4, 5, 34, 49].

However, end-to-end evaluation tells us little about whether a model is *actually following the pixels*. A system may produce globally plausible HTML/CSS while silently overriding a small but intentional local deviations present in the screenshot, such as a wider card (*e.g.,* a bounding box implemented as a) or a different font size. In practice, these are exactly the mistakes that matter: the output looks broadly correct, yet the generated code fails to preserve the precise design detail the developer intended to communicate. Figure 1 illustrates this failure: given a forum page where one post card is visibly wider than the rest, a model correctly computes the width at `120%`, then discards its own answer to match the other cards.

In particular, Vo et al. [38] demonstrate that state-of-the-art MLLMs can behave as *pattern completers*: rather than grounding their predictions in the visual input, they default to the statistically typical continuation of a pattern. Consider the simple example below, where a grid of shapes follows a repeating pattern of three circles per cell:



A model presented with this grid, when asked to count the circles in the last cell, is likely to say that it has three circles, because three circles better fit the surrounding pattern, even though the visual input clearly shows only two.

This failure mode is especially consequential for screenshot-to-code generation, where repeating UI structures are pervasive. A collection of uniformly styled cards, for instance, creates precisely the kind of strong visual pattern that can override a local deviation:

In such contexts, a model may generate code that faithfully reproduces the dominant pattern while discarding a small but visually present deviation, thus producing output that is plausible yet unfaithful to the source design.

In this paper, we study this behavior as *visual pattern-completion bias* in screenshot-to-code generation: the tendency of a model to generate code that aligns with a repeated pattern in a webpage's HTML rather than with the localized visual information in the screenshot. To measure this bias, we construct a benchmark of curated webpages from the Design2Code dataset [34]. We focus on two repeated UI-pattern families: (1) structural card patterns and (2) text style patterns. For each pattern, we modify exactly one element while keeping the surrounding page unchanged, mask the corresponding width or font-size value in the HTML snippet, and ask the model to recover the missing value from the screenshot and masked code context. This setup exposes whether the model follows the visual deviation or simply restores the pattern found in the HTML.

Using this benchmark, we evaluate five frontier proprietary MLLMs under two matched visual conditions: standard screenshots and noise-overlaid screenshots. We include two code-specialized models, `Codex-5.3` [30] and `Opus-4.6` [2], and three popular general-purpose models, `ChatGPT-5.3` [29], `Sonnet-4.6` [3], and `Flash-3.0` [10]. Our results reveal a consistent relationship between visual saliency (*i.e.,* how easily the perturbed element can be distinguished) and bias strength: card width perturbations, which are more spatially salient, yield lower bias than text font-size perturbations, which are fine-grained and harder to detect. `Codex-5.3` is the strongest performing model overall but even it collapses on text. Added visual noise, subtler perturbation magnitudes, and boundary positions all further reduce saliency and increase bias, confirming that the harder it is for the model to see the perturbed element, the more it defaults to the repeated pattern.

Importantly, this bias persists even when the model demonstrably attends to the correct visual region, as shown in our model reasoning analyses (Section 4.4). Models can identify and correctly describe a perturbed element, yet still override their own reasoning to produce pattern-consistent output. These findings suggest that current screenshot-to-code models cannot be trusted without explicit verification, and that the UI properties most sensitive to visual fidelity (*e.g.,* subtle spacing and font sizing) are where model reliability is lowest.

In summary, this paper makes the following contributions:

(1) We formulate *visual pattern-completion bias* as a failure mode in screenshot-to-code generation: models may prefer pattern-consistent values in webpage code over localized visual evidence in screenshot.
(2) We introduce a curated benchmark of 30 real-world webpages, yielding 1,440 evaluated screenshots with diverse conditions for fine-grained analysis (Section 3).
(3) We provide evidence showing that the magnitude of pattern-completion bias is strongly associated with visual saliency: more clearly-visible card perturbations yield 69.78% mean bias, while fine-grained text perturbations yield 80.22% (Section 4.1). Adding noise, subtler magnitudes, and boundary positions consistently reduces saliency and increases bias (Section 4.2).
(4) We provide detailed analysis of model reasoning on our benchmark. We find that as models reason more, their pattern-completion bias noticeably decreases (Section 4.3). Furthermore, our qualitative analysis (Section 4.4) reveals that these models can recognize the anomaly in the pattern yet still default to the pattern-consistent bias answer.

2 Related Work

2.1 Screenshot-to-code datasets & benchmarks

There have been extensive studies on datasets and benchmarks for screenshot-to-code. Earlier attempts, such as pix2code [5], demonstrated the feasibility of translating UI images into code. Recently,

Table 1: Pattern2Code compared to prior research. Existing work evaluates/improves page-level screenshot-to-code capability; Pattern2Code instead isolates whether models preserve *localized visual evidence* under conflict with repeated patterns.

Research direction	What is evaluated or optimized	Role in evaluating visual grounding
Screenshot-to-code benchmarks [4, 5, 12, 19, 22, 34, 36, 44, 45, 49, 52]	Whole-page generation quality and realism across increasingly complex web interfaces	Measure global fidelity but do not isolate whether models follow *localized* visual evidence when it conflicts with a repeated page-level pattern
MLLM hallucination & bias [11, 21, 24, 32, 33, 38, 48]	Visual reasoning, VQA, and pattern recognition under misleading or ambiguous conditions	Establish that MLLMs fail as pattern completers but do not study this failure mode in screenshot-to-code generation or repeated UI structures
Pattern2Code (ours): MLLM pattern completion bias	Bias from minimally perturbed repeated UI regions under matched screenshot conditions	Directly tests whether models preserve localized visual deviations or default to pattern-consistent code

WebSight [19] introduces a large synthetic dataset of 2 million pairs of screenshot and HTML, while WebCode2M [12] provides a large real-world dataset for webpage-to-code generation consisting of 2.56M instances. On the benchmark side, Design2Code [34], Web2Code [49], and WebMMU [4] evaluate visually grounded code generation and related web understanding tasks on increasingly realistic-looking webpages. Among these, only Design2Code's webpages come from real-world websites. More recent benchmarks further broaden the evaluation scope. WebUIBench [22] measures multiple WebUI-related capabilities, including perception, HTML programming, and WebUI-to-Code. DesignBench [45] extends evaluation to more comprehensive tasks such as code generation, editing, and repair. FrontendBench [52] evaluates models on front-end development in realistic scenarios, while FullFront [36] extend the task to full front-end engineering workflow. Beyond static page generation, Interaction2Code [44] studies interactive webpage generation. However, **these benchmarks do not directly test if a model strictly follows *localized* visual evidence**. Our work targets precisely this gap.

2.2 Visual front-end code generation

Alongside benchmark development, recent work has advanced screenshot-to-code generation through three complementary directions. *Open-source screenshot-to-code models* are trained or adapted specifically for UI or front-end code generation from screenshots or design prototypes [13, 17, 20, 42, 46, 47]. A second line of work focuses on *frameworks that enhance existing MLLMs* for screenshot-to-code generation. These approaches improve performance through structured pipelines, decomposition, or iterative refinement [18, 39, 51]. These frameworks enhance visually grounded code generation by imposing additional structure on top of existing multimodal models. Finally, *closed-source multimodal coding systems* such as OpenAI Codex [28], Claude Code [1], and Gemini Code Assist [9] has become highly prevalent in real-world software development. They are high-capability multimodal coding assistants that support software engineering workflows and can be applied to visually grounded code generation tasks. In this work, our focus is **diagnostic**: rather than proposing a stronger generation pipeline, **we study a specific failure mode, namely *visual pattern-completion bias***, that existing high-capability systems may exhibit.

2.3 Evaluating bias and hallucination in MLLMs

Our work is also related to recent research on visual bias and hallucination in MLLMs [7, 16, 32, 33, 37, 48, 50]. Among these work, HallusionBench [11], VLind-Bench [21] and PhD [24] demonstrate that multimodal reasoning fails under misleading or ambiguous visual conditions where pretrained knowledge conflicts with visual evidence. Most closely related to our work, VLMsAreBiased [38] show that MLLMs often behave as *pattern completers*: when localized visual evidence conflicts with a familiar pattern, models may default to the typical continuation of said pattern. These benchmarks establish that multimodal models can fail in systematic, prior-driven ways. However, they do not study screenshot-to-code generation, specifically, the interaction between repeated UI structure and code prediction. Our work **transfers the core diagnostic insight to screenshot-to-code setting** and focuses on minimally perturbed webpage screenshots and masked HTML snippets.

2.4 On Advancing the State-of-the-Art

Prior screenshot-to-code benchmarks mainly assess *page-level* capability, *i.e.,* whether a model can generate plausible front-end code from a screenshot. Rather than introducing another general evaluation suite, we isolate a **specific visually grounded failure mode**, namely whether a model follows *localized visual evidence* when it conflicts with a repeated page-level design pattern. This distinction matters because systems with strong page-level performance can still have systematic failures on subtle local deviations. Unlike prior studies of bias in code generation, which have largely examined socially related bias, fairness, or linguistic skew [15, 23, 25], we focus instead on *pattern-completion bias* in screenshot-to-code scenarios. To quantify this behavior, we introduce an evaluation benchmark based on minimally perturbed webpages with masked HTML snippets and matched screenshot conditions. Table 1 summarizes how Pattern2Code relates to prior research directions. To the best of our knowledge, this is the first work to explicitly evaluate pattern-completion bias in screenshot-to-code systems.

3 Study Design

We conduct a controlled *pattern-versus-visual-evidence* evaluation for screenshot-to-code generation. Our key idea is to construct webpage instances in which most of the visual layout follows a repeated design pattern, while **a single localized element intentionally deviates from that pattern**. Under this setup, a well-grounded

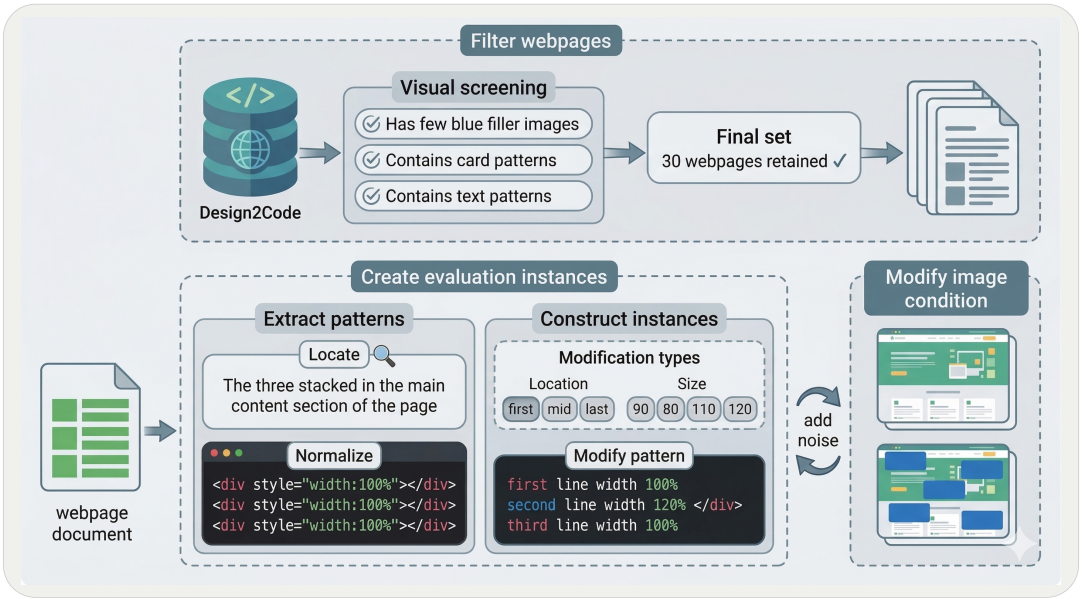


Figure 2: Pattern2Code benchmark construction. From 484 Design2Code webpages, we retain 30 with repeated UI structures, extract and perturb one element per pattern across three positions and four magnitudes, and render each under standard and noise-overlaid conditions (1,440 screenshots per model).

model should recover the localized deviation from the screenshot, whereas a model that over-relies on learned design priors may instead revert to the visually dominant pattern.

This setting is especially relevant to front-end implementation. Webpages often contain repeated UI structures, such as cards, menus, lists, and feature blocks, while still including a small number of intentional exceptions. These exceptions may reflect important design intent, such as emphasis or hierarchy. As a result, a screenshot-to-code model that reconstructs the overall layout correctly but misses such localized deviations can produce outputs that seem plausible while still being implementation-wise incorrect.

To this end, we study *repeated UI patterns* through two perturbation families: (1) structural card patterns and (2) text style patterns. For card patterns, we modify the inline CSS width of one card in a repeated stacked-card panel. For text patterns, we modify the font size of one element in a repeated text sequence. In both cases, we keep the surrounding page unchanged, mask the target value in the corresponding HTML snippet, and ask the model to infer the missing value from the screenshot and masked code context. This design isolates whether the model follows localized visual evidence or defaults to the dominant repeated pattern.

3.1 Benchmark Construction

To evaluate models in realistic screenshot-to-code conditions, we build the benchmark from real-world webpages in the Design2Code dataset [34], focusing on pages that contain *repeated UI patterns*—the natural setting in which pattern-completion bias can be observed. The construction pipeline features three stages: (1) filtering Design2Code webpages to those with suitable repeated structures, (2) extracting card and text-style patterns from each accepted page and perturbing exactly one element per pattern, and (3) rendering each instance under standard and noise-overlaid visual conditions. Figure 2 shows the methodology we implemented to construct our benchmark Pattern2Code.

3.1.1 Filtering the webpages. We begin with the 484 webpages in Design2Code and apply two rounds of manual screening. Because Design2Code replaces many original images in the webpages with blue placeholders, we retain only pages that (1) contain clear repeated UI structures, and (2) are not visually dominated by placeholder regions. The most common cause for not selecting a webpage is the absence of a clear card pattern (413 pages, i guess is 93.9% (93.7)% of rejections), followed by blue-placeholder dominance (144, 32.7%). (As a webpage can have multiple causes of rejection, this does not sum to 100%.) A second pass applying the same criteria removes 14 more, yielding a final set of 30 webpages. The second-pass rejections fell into two categories: debatable patterns (e.g., fewer than three comparable elements or elements too small for the perturbation to be reliably perceived) and structurally hard-to-modify source code.

3.1.2 Constructing the instances. From the 30 accepted webpages, we construct benchmark instances from two perturbation families: *structural card patterns* and *text style patterns*. Structural card patterns are repeated stacked-card panels in which neighboring cards share common dimensions and alignment. Text style patterns are repeated text sequences, such as navigation items or menu entries, in which neighboring elements share a common font size. We extract one card pattern and one text pattern per webpage.

For each selected webpage, we then create perturbed instances by editing elements in three positions: the first element, the middle

element (*e.g.*, the element at the median position in the pattern), and the last element of the repeated pattern. This provides a meaningful positional spread without requiring us to modify every element in every pattern. We normalize the values of each element in the pattern to 100% to standardize evaluation. For each position, we use four perturbation values: `80%`, `90%`, `110%`, and `120%`. For structural card patterns, we modify the inline HTML/CSS width of exactly one card. For text style patterns, we modify the font size of exactly one text element. In both cases, the value of the unmodified elements in the pattern is `100%`, which serves as the *bias-aligned baseline*.

We choose width and font-size perturbations because they are visually salient and less ambiguous than changes in properties such as color, which may be confounded by rendering artifacts.

3.1.3 Modifying image conditions. To study how visual context affects model behavior, we render each webpage screenshot under two matched visual conditions: standard and noise-overlaid. This produces a controlled evaluation setting in which the underlying webpage instance and masked code context remain fixed, while only the visual context changes.

(1) **Standard.** We render the perturbed webpage using Playwright [26] with a viewport size of 1000 × 1400. This condition preserves the full webpage context and serves as the default screenshot-to-code setting.
(2) **Noise-overlaid.** Starting from the standard rendering, we overlay eight opaque rectangles of fixed size (80 × 40 pixels) at random positions on the screenshot using a fixed random seed of 42. This condition introduces irrelevant visual noise while preserving the underlying webpage structure and image dimensions.

Because all conditions are derived from the same base HTML instance, differences in performance can be attributed to changes in visual context rather than changes in code context or target value. We emphasize that the noise-overlaid condition is a *controlled saliency probe* rather than a model of realistic screenshot degradation. We discuss this scope limitation in Section 5.

3.1.4 Benchmark statistics. In total, our evaluation suite comprises of 1,440 instances in total: each of the 30 webpages yields 24 instances (3 positions × 4 magnitudes × 2 pattern families), each rendered under 2 image conditions, for 30 × 24 × 2 = 1,440 evaluated screenshots per model.

3.2 Benchmark Diversity

The accepted webpages span multiple website genres, including blog and personal pages (6), product and business websites (6), forum and community pages (5), education and reference pages (5), directory and catalog sites (4), and institutional and news pages (4). This diversity helps reduce the chance that the observed bias is an artifact of a single webpage genre or design style.

3.3 Task Formulation

Given a (1) rendered webpage screenshot and (2) a masked HTML snippet, the model is asked to recover the missing percentage value. Concretely, we replace the target width or font-size value in the HTML with the token `__` and present the model with instructions shown in the prompt below.

In this prompt, {`pattern`} is a concise natural language description of the pattern to assist the model locate it. For example, in Figure 5a, the pattern is *"The four stacked information cards in the main left column under Detailed Information about Book Reviews (Text)"*. These descriptions follow the principle of "the {number of elements} {pattern family} {pattern location}" and are manually curated. We further request the `"<value>"` to be divisible by 10 to narrow down the answer space for better alignment with the perturbation values, as models can generate raw values like 118% initially (Figure 1).

Model Prompt

You are doing a visual code fill-in-the-blank task. Given the webpage screenshot and HTML snippet below, fill the blank token `__` with ONLY the missing CSS/HTML value. The goal is to predict the masked value to recreate the {`pattern`} in the webpage design. Return the final answer in JSON format: `{"answer":"<value>"}`. The blank must be a percentage value divisible by 10.

3.4 Studied Models and Inference Setup

We evaluate five frontier proprietary MLLMs drawn from three major model families: OpenAI GPT, Anthropic Claude, and Google Gemini. We group them by intended use: two *code-specialized* models (`Codex-5.3` and `Opus-4.6`) designed for software engineering workflows, and three *general-purpose* models (`ChatGPT-5.3`, `Sonnet-4.6`, and `Flash-3.0`). We focus on these model families because they are widely used by professional developers for development work: according to the 13,271 responses from the 2025 Stack Overflow Developer Survey [35], these models are the most popular options (e.g., 81.9% for GPT, 44.9% for Claude, and 34.4% for Gemini), whereas open-source models remain substantially less common, with the most popular being DeepSeek at 21.9%. Furthermore, with the rise of coding assistants like Claude Code [1] and Codex [28], these models are more relevant than ever.

This choice is also practically useful: if even strong and widely used frontier models fail in this setting, then the failure mode is likely to matter in real-world usage. We do not evaluate open-source screenshot-to-code models, as they are not currently used at a comparable rate by professional developers.

We access all models through OpenRouter [31], an AI inference service that connects developers to various LLMs, in order to standardize the inference interface across providers. For each example, we perform a single model call with the screenshot and prompt packaged as a multimodal user message. We set `max_output_tokens` to 4096 to ensure sufficient completion budget during inference; all other parameters are left to the default configuration of each model (e.g., no explicit temperature, top-p, or top-k overrides were set). Each model is evaluated once per rendered screenshot. Inference was performed in late March 2026. Answers are extracted from the final `{"answer": ...}` JSON object in each response. Each sample was run exactly once, and none of the 7,200 evaluated responses failed to yield a parseable answer.

3.5 Evaluation Metrics

We evaluate model performance using three mutually exclusive outcome categories. Let D denote the evaluation set of |D| pairs of *{HTML snippets, webpage screenshot}*. For each instance $i \in D$, let y_i denote the model prediction, g_i the ground-truth perturbed value (*e.g.,* `80%` or `120%`), and b_i the pattern-consistent baseline (always `100%`). We define:

Accuracy. The fraction of predictions that exactly recover the localized deviation from the repeated pattern:

$$\text{Accuracy} = \frac{1}{|D|} \sum_{i \in D} [y_i = g_i] \tag{1}$$

where $[\cdot]$ denotes the Iverson bracket, equal to 1 when the enclosed condition holds and 0 otherwise.

Bias rate. Our primary diagnostic metric, measuring how often the model reverts to the dominant repeated value rather than reproducing the visual evidence:

$$\text{Bias Rate} = \frac{1}{|D|} \sum_{i \in D} [y_i = b_i] \tag{2}$$

Other error. A supplementary category capturing predictions that depart from the repeated baseline ($y_i \neq b_i$) yet still miss the ground truth ($y_i \neq g_i$)—cases where the model detects an inconsistency in the pattern but fails to resolve it correctly:

$$\text{Other Error} = \frac{1}{|D|} \sum_{i \in D} [y_i \neq g_i \wedge y_i \neq b_i] \tag{3}$$

These metrics allow us to distinguish between models that are truly visually grounded (high accuracy), those that are blinded by pattern-completion bias (high bias rate), and those that detect an anomaly but cannot precisely quantify it (high other error).

4 Experimental Results

We organize the results around three research questions. We first establish whether MLLMs follow repeated UI patterns over localized visual evidence when the two disagree. We then examine what makes this bias stronger or weaker by varying visual context and local salience. Finally, we study whether greater model reasoning effort is associated with lower bias or higher accuracy.

We anticipate that, unless otherwise noted, RQ1 and RQ2 report results on the *standard* (noise-free) screenshots only, so that the noise overlay does not confound the factor under analysis: n=360 instances per model and pattern family (30 webpages × 3 positions × 4 perturbation magnitudes), *i.e.,* 720 of the 1,440 screenshots evaluated per model. The noise analysis in Table 3 contrasts these instances with their 360 matched noise-overlaid counterparts per pattern family, and RQ3 draws on all 1,440 responses per model (both pattern families and both image conditions).

4.1 RQ1: Do models follow the HTML pattern or the visual evidence in perturbed scenarios?

All five MLLMs tend to default to the repeated pattern over localized visual evidence (Table 2). The severity of pattern-completion bias is strongly associated with visual saliency: Card width perturbations produce spatially large, layout-level deformations that are relatively easy to see, whereas font-size perturbations produce fine-grained, character-level changes that are much harder to detect. Mean accuracy drops from 21.17% on cards to 7.89% on text, while mean bias rises from 69.78% to 80.22%. The harder it is for the model to see the deviation, the more it defaults to the repeated pattern.

Table 2: All five MLLMs revert to the repeated baseline `100%` when patterns and pixels disagree, but the bias rate differs sharply by model class and pattern family. Card pattern perturbations are easier to recognize, while text patterns perturbations are fine-grained character-level changes. Mean accuracy drops from 21.17% on cards to just 7.89% on text, confirming that lower visual saliency drives stronger bias.

	Card Patterns			Text Patterns		
Model	Acc. (%)	Bias (%)	Oth. (%)	Acc. (%)	Bias (%)	Oth. (%)
Code-specialized models						
Codex-5.3	**68.61**	**26.39**	5.00	**13.89**	70.56	15.56
Opus-4.6	6.39	87.78	5.83	5.56	88.06	6.39
General-purpose models						
ChatGPT-5.3	8.33	69.44	22.22	10.28	**66.94**	22.78
Sonnet-4.6	8.06	83.33	8.61	8.89	79.44	11.67
Flash-3.0	14.44	81.94	3.61	0.83	96.11	3.06
Mean	21.17	69.78	9.06	7.89	80.22	11.89

On structural card patterns, four of five models default to the repeated baseline more than 69% of the time, with only Codex-5.3 answering correctly in the majority of cases (68.61% accuracy, 26.39% bias). Flash-3.0 (14.44% accuracy) is the next best, but still defaults to the baseline 81.94% of the time. Opus-4.6 (87.78% bias), Sonnet-4.6 (83.33% bias), and ChatGPT-5.3 (69.44% bias) all fall below 9% accuracy. ChatGPT-5.3 shows lower bias but does not translate it into accuracy (8.33%); instead, 22.22% of its predictions fall into the "Other" category, indicating that it detected the inconsistency but failed to retrieve the correct value.

On text style patterns, pattern-completion bias rises noticeably for most models. Flash-3.0 reaches 96.11% text bias with under 1% accuracy, collapsing almost entirely. Opus-4.6 reaches 88.06% text bias. Codex-5.3 , the strongest model on cards, drops from 68.61% to 13.89% accuracy and sees its bias rise from 26.39% to 70.56%. Sonnet-4.6 (79.44%) and ChatGPT-5.3 (66.94%) show a different pattern: their text bias is comparable to or slightly below their card bias, but their text accuracy stays below 11%. The gap is absorbed by *other* errors: ChatGPT-5.3 reaches 22.78% *other* errors on text, the highest of any model, meaning it often detects the inconsistency but cannot recover the correct value. Flash-3.0 , by contrast, shows less than 4% *other* errors on both cards and text—when it fails, it almost always defaults to the pattern rather than attempting an alternative. The card-to-text gap at the aggregate level—mean bias rising from 69.78% to 80.22%—is the clearest evidence that visual saliency, not model capability alone, determines the strength of pattern-completion bias.

Code specialization does not reliably reduce pattern-completion bias. Among the two code-specialized models, Codex-5.3 is the most visually grounded model in the study, but Opus-4.6 reaches 87.78% bias on card patterns and 88.06% on text patterns, surpassed only by Flash-3.0 on text. Conversely, the general-purpose ChatGPT-5.3 achieves lower card pattern bias (69.44%) than the

code-specialized Opus-4.6 (87.78%). This suggests that pattern-completion bias depends more on model-specific design choices than on model specialization.

Answer to RQ1

All MLLMs exhibit pattern-completion bias strongly associated with visual saliency: mean accuracy drops from 21.17% on cards to 7.89% on text and mean bias rises from 69.78% to 80.22%. Code specialization does not help: Opus-4.6 is code-specialized yet reaches 87.78% card bias and 88.06% text bias. These results suggest that stronger coding ability does not translate into stronger visual grounding under repeated UI patterns.

4.2 RQ2: What makes pattern-completion bias stronger or weaker?

RQ1 established that bias is strongest when the visual deviation is fine-grained (text vs. cards). We now test this saliency–bias relationship from three additional angles: by reducing saliency through noise, by varying perturbation magnitude, and by changing the position of the deviation within the repeated pattern. All three analyses converge on the same mechanism: *pattern-completion bias is strongest when the localized visual signal is hardest to isolate.*

Table 3: Noise generally amplifies pattern-completion bias. On cards, Flash-3.0 shows the largest increase (+9.44), followed by Sonnet-4.6 (+6.11%); ChatGPT-5.3 is the only model whose card bias decreases under noise. On text, bias is already high under standard conditions and noise has less room to worsen it. Δ denotes the change from Standard to noise-overlaid. The Standard and Noise columns each aggregate n=360 instances per model and pattern family –the same 30 webpages × 3 positions × 4 magnitudes, rendered without and with the noise overlay.

	Card Bias Rate (%)			Text Bias Rate (%)		
Model	Standard	Noise	Δ	Standard	Noise	Δ
Code-specialized models						
Codex-5.3	**26.39**	**30.83**	+4.44	**70.56**	**75.00**	+4.44
Opus-4.6	87.78	90.00	+2.22	88.06	90.28	+2.22
General-purpose models						
ChatGPT-5.3	69.44	67.78	−1.67	66.94	68.33	+1.39
Sonnet-4.6	83.33	89.44	+6.11	79.44	80.56	+1.12
Flash-3.0	81.94	91.39	**+9.44**	96.11	96.11	0.00
Mean	69.78	73.89	+4.11	80.22	82.06	+1.84

Noise amplifies already strong bias. Table 3 isolates the effect of visual noise by holding the HTML context fixed across matched standard and noise-overlaid screenshots. On structural cards, noise pushes four of five models toward higher bias; the exception is ChatGPT-5.3, whose card bias decreases by 1.67% under noise. Flash-3.0 shows the largest increase, jumping from 81.94% to 91.39% bias (+9.44%), followed by Sonnet-4.6 (+6.11%) and Opus-4.6 (+2.22%). On text, bias is already high under standard conditions and noise has less room to worsen it: the mean text delta (+1.84%) is smaller than the mean card delta (+4.11%). In conclusion, noise degrades saliency leading to higher bias, but when the baseline bias is already high, further degradation has diminishing marginal effect.

Subtler perturbations amplifies bias. Table 4 shows that perturbation magnitude is a direct proxy for visual saliency and bias strength. On cards, Codex-5.3 achieves 83.3% accuracy at the highly salient 80% level but drops to 48.9% at the subtle 110% level, where its bias more than quadruples (10.0% → 46.7%). The same gradient appears in the mean: bias rises from 50.7% at 80% to 82.2% at 110%. On text, the effect is even sharper. Mean bias peaks at 88.2% for 90% and 87.4% for 110%, the two values closest to the baseline value of 100%, while easing to 69.8% at the more visible 80%. A directional asymmetry also emerges: models default to the biased answer more often on *widened elements* (110%, 120%) than shrunk ones (80%, 90%), suggesting that increasing the element size is more visually challenging.

Boundary deviations are harder than middle deviations. Table 5 shows that position effects are real but smaller than those of noise or magnitude. Middle-position perturbations are flanked by pattern-consistent elements on both sides, making the deviation more salient by contrast. This yields the lowest mean bias (61.67%) and highest mean accuracy (27.33%). The effect is clearest for the models not already at ceiling: Codex-5.3 improves from 62.50% (first) to 74.17% (middle), and Flash-3.0 jumps from 3.33% to 29.17%. Models already collapsed to the baseline, such as Opus-4.6, remain bias-dominated regardless of position (85.83%–90.00%).

Taken together, noise, magnitude, and position are three different ways of modulating the same underlying variable: the visual saliency of the local deviation. In every case, reducing saliency increases pattern-completion bias. This reinforces the RQ1 finding that the card-to-text gap is driven by saliency: card perturbations are layout-level and spatially obvious; text perturbations are character-level and fine-grained. The mechanism is consistent across all factors we tested.

Answer to RQ2

Noise adds +4.11% mean card bias. Subtler perturbations raise card bias from 50.7% to 82.2%. Middle positions yield the lowest bias (61.67%). Together, these results show that the harder a local deviation is to perceive, the more likely models are to fall back to the repeated baseline instead of following the visual evidence.

4.3 RQ3: Do more reasoning tokens improve performance?

Longer reasoning allows models to scrutinize the details, which may lead to better performance. For this analysis, we draw on the per-example token traces recorded in our inference logs. Only the two OpenAI models provide `usage_reasoning_tokens` values, so we restrict the RQ3 analysis to ChatGPT-5.3 and Codex-5.3.

Table 4: Subtler perturbations amplify bias. On cards, mean bias rises from 50.7% at `80%` to 82.2% at `110%`. On text, the effect is even stronger: `90%` and `110%` perturbations reach ~88% mean bias. Entries report Accuracy / Bias (%).

	Structural Card Patterns				Text-Style Patterns			
Model	80%	90%	110%	120%	80%	90%	110%	120%
Code-specialized models								
Codex-5.3	**83.3 / 10.0**	**77.8 / 18.9**	**48.9 / 46.7**	**64.4 / 30.0**	**28.9 / 41.1**	6.7 / 84.4	3.3 / 85.6	**16.7 / 71.1**
Opus-4.6	20.0 / 71.1	5.6 / 86.7	0.0 / 96.7	0.0 / 96.7	14.4 / 81.1	1.1 / 94.4	0.0 / 96.7	6.7 / 80.0
General-purpose models								
ChatGPT-5.3	18.9 / 48.9	11.1 / 73.3	0.0 / 80.0	3.3 / 75.6	7.8 / 63.3	**11.1 / 75.6**	**6.7 / 68.9**	15.6 / **60.0**
Sonnet-4.6	20.0 / 62.2	12.2 / 82.2	0.0 / 96.7	0.0 / 92.2	26.7 / 66.7	3.3 / 91.1	1.1 / 88.9	4.4 / 71.1
Flash-3.0	31.1 / 61.1	11.1 / 85.6	8.9 / 91.1	6.7 / 90.0	3.3 / 96.7	0.0 / 95.6	0.0 / 96.7	0.0 / 95.6
Mean	34.7 / 50.7	23.6 / 69.3	11.6 / 82.2	14.9 / 76.9	16.2 / 69.8	4.4 / 88.2	2.2 / 87.4	8.7 / 75.6

Table 5: Position within the repeated pattern affects bias on structural card patterns. Middle-position perturbations, flanked by pattern elements on both sides, are most salient and yield the lowest mean bias (61.67%). Boundary positions show higher bias.

	First		Middle		Last	
Model	Acc.	Bias	Acc.	Bias	Acc.	Bias
Code-specialized models						
Codex-5.3	62.50	33.33	**74.17**	**20.00**	69.17	25.83
Opus-4.6	5.83	87.50	9.17	85.83	4.17	90.00
General-purpose models						
ChatGPT-5.3	7.50	62.50	10.83	64.17	6.67	81.67
Sonnet-4.6	3.33	90.00	13.33	74.17	7.50	85.83
Flash-3.0	3.33	95.83	29.17	64.17	10.83	85.83
Mean	16.50	73.83	27.33	61.67	19.67	73.83

We note that reasoning-token count is observed rather than experimentally controlled: models choose how much to reason per example. As such, the relationship we report is a correlation between reasoning effort and bias.

Figures 3 and 4 tell a clear story: *examples answered with longer reasoning exhibit sharply lower bias, and accuracy improves up to a threshold where the gains diminish.*

The contrast is sharpest for ChatGPT-5.3 . When the model produces an answer with zero reasoning tokens, which happens in over half of all examples (806 of 1,440; 56.0%), it is almost always biased (93.2%). Once reasoning kicks in at an intermediate level (centered at ~64 tokens), accuracy rises to 21.3% and bias is cut by more than half to 40.8%. Beyond that threshold, however, performance plateaus. The highest-effort region (centered at ~149 tokens) yields almost identical numbers (21.3% accuracy, 39.7% bias).

Codex-5.3 shows a similar pattern, but with a more gradual increase. Accuracy rises from 20.8% in the lowest-effort bin (centered at ~97tokens) to 45.6–47.5% across two intermediate bins (centered at ~200 and ~321 tokens), while bias retreats from 78.1% to roughly 40%. Yet the highest-effort bin (centered at ~611 tokens) adds little: accuracy drops to 44.0% and bias moves to 39.3%. The takeaway is consistent: examples with more deliberation land on

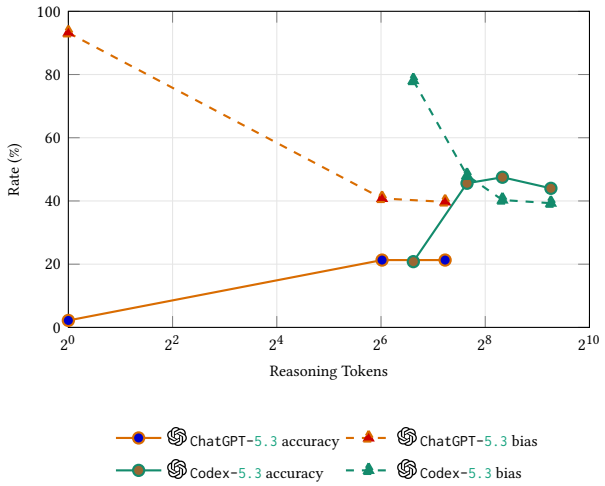


Figure 3: Reasoning effort *vs* accuracy and bias rates for the two OpenAI models. Each point is a model-specific effort bin positioned by the $\log_2$ of its mean reasoning-token count. More reasoning consistently correlates with lower bias.

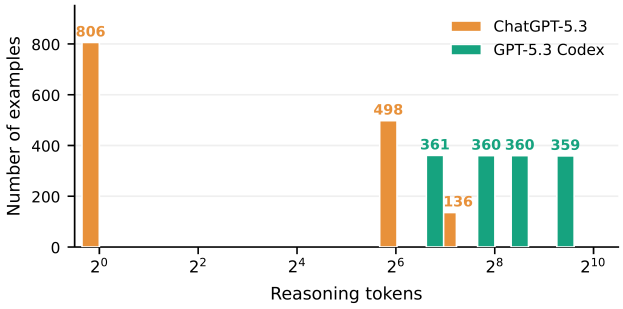


Figure 4: Number of examples per reasoning effort: ChatGPT-5.3 reasoning token distribution unbalanced: 806 of 1,440 examples use zero reasoning tokens. Codex-5.3 always reasons, with bins of roughly equal size.

the default bias answer far less often, but the extra effort does not guarantee better accuracy.

Answer to RQ3

Longer reasoning correlates with lower bias, but with diminishing returns for accuracy. Intermediate effort drops bias from 93.2% to ~40% for ChatGPT-5.3 . General models like ChatGPT-5.3 tend to jump directly to the final answer, leading to stronger bias rate.

4.4 Qualitative analysis: examine model reasoning on modified patterns

To investigate why these models are systematically biased, we examine the free-text reasoning traces produced by the three models that expose them: Sonnet-4.6 , Opus-4.6 , and Flash-3.0 . We omit the two OpenAI models as their reasoning tokens are encrypted. These traces come from the *same* inference runs and the same prompt as the quantitative evaluation: the models emit their reasoning followed by the required JSON answer, and we score the JSON answer while analyzing the accompanying text.

We identify a recurring three-step pattern that we call the *pattern conformity* behavior: (1) the model observes the visual deviation, (2) it begins correct reasoning, and (3) it overrides its own observation to conform to the repeated pattern. We illustrate with representative examples from standard card screenshots.

Models perceive the deviation but override it. In many cases, models explicitly describe the visual anomaly and even estimate the correct value (Figures 1 & 5), then abandon their reasoning in favor of the pattern. For instance, on a 120%-width card perturbation (Figure 1), Sonnet-4.6 computes the width ratio as ~118%, correctly identifies that rounding yields 120%, and then concludes: *"However, looking more carefully, 100% seems most consistent with the pattern"*. Opus-4.6 also describes a card as *"about 80% of the full width"* on a ground-truth 80% instance, then writes: *"It's most likely also 100% to match the consistent layout of all four cards."* (Figure 5a). In both cases, the model had the correct answer and discarded it. Similarly, Flash-3.0 after noting that a card *"spans a significantly smaller portion of the row"*, concludes *"100% is the most consistent choice despite the visual rendering of the box inside"* (Figure 5b). These responses show that the bias operates as a reasoning heuristic: models believe that repeated elements *should* share the same value.

4.5 Quantifying the Failure Modes

While the qualitative examples in Section 4.4 demonstrates how visual overrides happen in pattern-completion bias, they do not tell us how *prevalent* each mechanism is. We therefore systematically classified every biased response that contains a free-text rationale, with 237 traces in total (192 from Sonnet-4.6 , 22 from Opus-4.6 , 23 from Flash-3.0), drawn from the same inference runs across both image conditions. We divide them into three mutually exclusive failure modes:

- **(A) Code-anchored:** the rationale reasons only from the HTML source (*e.g.,* "the other cards use 100%, so this should match") and never references the rendered screenshot; the baseline answer is a default, not a visual judgment.
- **(B) Perceived-consistent:** the rationale cites the screenshot but reports the perturbed element as *visually matching* its neighbors; the deviation is present in the pixels but is not registered.
- **(C) Observed-then-overrode:** the rationale explicitly notes that the element looks different ("wider," "narrower," "extends beyond"), yet the final answer is still the baseline. This behavior demonstrates strong pattern-completion bias in MLLMs which we illustrated in Figure 5.

LLM-As-a-judge setup. Labels are assigned by an LLM judge (`gpt-5.4-mini` via OpenRouter, temperature 0) using the two-step decision rule below. A key design choice is that the A/B/C discriminator is the trace's *observation* of the pattern: phrases like "should match the others" appear in both B and C rationales, so category C requires an explicit claim that the element *looks* different, even if the trace later talks itself back to "consistent."

Judge Prompt

Classify ONE reasoning trace into exactly one mode. Setup: the model filled a masked CSS width/font-size for one element among siblings; in the screenshot that element was genuinely perturbed to a DIFFERENT size, yet the model answered the baseline (100%, same as siblings). Decide WHY, in strict order:

STEP 1 – Does the trace describe the TARGET's RENDERED appearance (its size relative to siblings)? Talk purely about the HTML/CSS source ("the others are `width:100%`, so match") does NOT count.

NO visual description → **`A (CODE_ANCHORED)`**

STEP 2 – (only if it looked) Does it assert that the target looks DIFFERENT – narrower/wider/smaller/larger/bolder/overflows/"extends beyond", or estimates a non-100 size (~80%, ~120%)?

Only ever says SAME / consistent / uniform / matches → **`B (PERCEIVED_CONSISTENT)`**

Asserts a difference at any point, EVEN IF it then reconciles to "same" or picks 100% to match the pattern → **`C (OBSERVED_THEN_OVERRODE)`**

CRITICAL: "should match the others / be consistent" is NOT evidence of C – B says that too. C requires an explicit claim that the target LOOKS DIFFERENT.

Answer with ONE letter: A, B, or C.

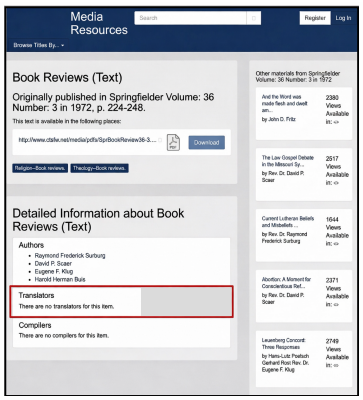
—

Model reasoning: {content}

Cross-validation. We validated the judge against 30 traces sampled uniformly from the same population and hand-labeled *before* inspecting the judge's output: agreement was 29/30 (96.7%), and the single disagreement traced to a transcription error in the hand labels rather than a judge mistake.

Two conclusions follow from Table 6. First, the bias is not an artifact of ignoring the image: in ~74% of verbalized biased responses the model demonstrably engages the screenshot (B + C). Second, the dominant case is *perceptual*: the pattern prior most often suppresses the perception of a deviation that is present in the pixels (B, 59.1%), while the explicit override of Figure 5 is a real but smaller, model-dependent mode (C, 14.8% overall but 45.5% for Opus-4.6).

(a) Self-talk-down: Opus-4.6 on a 80%-width card



Ground truth: 80%

*"The Translators card appears to be **about 80% of the full width** compared to the other cards which span 100%. [...] However, examining more carefully [...] the Translators card seems to extend to about the same width as the others. [...] It's most likely also **100% to match the consistent layout** of all four cards."* Answer: 100% ✗

(b) "Despite the visual rendering": Flash-3.0

Ground truth: 80%

*"The 'Message' field's input is **visibly narrower** than the input fields above it. [...] it **spans a significantly smaller portion** of the row compared to the full-width fields above. [...] **100% is the most consistent choice despite the visual rendering** of the box inside."*

Answer: 100% ✗

Figure 5: Two examples of pattern-completion bias in model reasoning. In (a), Opus-4.6 correctly estimates the card width at 80% but overrides its own observation to match the repeated pattern. In (b), Flash-3.0 explicitly acknowledges the visual deviation ("visibly narrower", "significantly smaller") yet chooses the pattern-consistent value "despite the visual rendering." Both models perceive the deviation but default to 100%.

Table 6: Failure-mode distribution over the 237 verbalized biased responses. Perceived-consistency (B) dominates; the explicit override (C) is real but smaller and model-dependent.

Model	n	A: code (%)	B: perceived (%)	C: override (%)
Opus-4.6	22	13.6	40.9	**45.5**
Sonnet-4.6	192	26.0	**61.5**	12.5
Flash-3.0	23	39.1	**56.5**	4.3
Overall	237	26.2	**59.1**	14.8

Takeaway from Failure Mode Analysis

Analysis of the reasoning traces reveals that models (1) observe the visual deviation and begin correct inference, then (2) override their own observation to conform to the repeated pattern. Labeling of all 237 biased responses shows that ~74% engage the screenshot, with perceived-consistency (59.1%) as the dominant mechanism and explicit override at 14.8%.

5 Threats to Validity

Internal validity. Our benchmark relies on controlled synthetic construction: we inject a single localized deviation into otherwise pattern-consistent webpages and create noisy screenshot as synthetic transformations of the original renderings. This gives us precise control over visual context and makes it easier to attribute errors to pattern-completion bias. A similar design choice appears in prior multimodal benchmarks such as HallusionBench, VALSE, and IllusionVQA, which also use controlled perturbations to study specific visual failure modes. Likewise, screenshot-to-code benchmarks such as Design2Code and WebSight rely on curated or generated webpages to support consistency and reproducibility. At the same time, our synthetic setup does not fully capture the visual complexity of real-world web page screenshots, where unpredictable layout variations are admissible and may overstate or understate model susceptibility in practice. Extending the benchmark to more diverse web pages remains a key direction for future work.

Construct validity. Our primary metric is *bias rate*, defined as the proportion of predictions equal to 100%. This choice directly targets the failure mode under study, pattern-completion bias, but it emphasizes one specific error direction and does not fully capture other forms of visually grounded reasoning failure. Furthermore, our task formulation constrains answers to percentage values divisible by 10, which simplifies evaluation but may not reflect the full range of CSS values a model would produce in unconstrained generation.

External validity. We evaluate two perturbation families—inline CSS width on card panels and inline font-size on text sequences—across 30 Design2Code webpages. Although this setting cleanly isolates pattern-completion bias, it does not cover other localized deviations such as spacing, alignment, ordering, color, or component presence/absence. All seed webpages originate from a single source (Design2Code); benchmarks constructed from other web

corpora, design styles, or webpage structures may surface different bias magnitudes or patterns. Moreover, our findings characterize *one-shot, unaided* MLLM behavior: in practice, developers and agentic systems may re-render generated code, inspect computed styles, or apply visual-diff checks, and such tool-augmented workflows may exhibit different bias profiles than the raw models we study.

Conclusion validity. Our study reports results for five proprietary frontier MLLMs from three model families. Although these represent the most widely adopted options in professional development [35], they do not cover open-weight models or specialized screenshot-to-code systems. Different architectures, training regimes, or alignment strategies may yield different bias profiles. Accordingly, our findings should be read as evidence that pattern-completion bias exists and follows a saliency-driven mechanism in current frontier systems, not as an exhaustive characterization of all multimodal code generation models.

6 Discussion and Conclusion

We introduced a controlled benchmark for measuring *visual pattern-completion bias* in screenshot-to-code generation. The benchmark asks a simple but important question: *when a localized visual deviation conflicts with a repeated UI pattern, does the model follow the pixels or the pattern?"* Across 720 base instances rendered under two matched image conditions (1,440 screenshots per model), the answer is unambiguous: all five MLLMs exhibit pattern-completion bias, and its severity tracks the strength of the visual signal (*i.e.,* saliency). Card-width perturbations are detectable by the strongest model (Codex-5.3 at 68.61% accuracy), but text font-size perturbations cause even the best model to collapse (13.89% accuracy, 70.56% bias). Mean text bias reaches 80.22%, with Flash-3.0 at 96.11%. Noise, subtler magnitudes, and boundary positions each further reduce saliency and increase bias, confirming a single mechanism across every factor we tested.

These results impact and have implications along four axes:

A saliency threshold shapes when models can be trusted. Pattern-completion bias operates along a gradient, not as a binary switch. When a localized deviation is spatially large, frontier models can sometimes override their pattern prior. As the deviation shrinks below a perceptual threshold, moving from layout-level card perturbations to character-level font-size changes, models collapse to the dominant pattern at striking rates. **For practitioners, this implies that screenshot-to-code output can be broadly trusted for coarse layout decisions but should *not* be trusted for fine-grained styling without explicit verification**. The very properties that matter in polished front-end work (*e.g.,* subtle spacing, font sizing) sit below this threshold.

End-to-end metrics can hide local grounding failures. Standard screenshot-to-code benchmarks focus on overall visual similarity or structural overlap. A model can perform well on these metrics while still missing the small local deviations that our benchmark is designed to expose. In our setting, this failure is not random: models often revert to the repeated baseline `100%`. Because the resulting outputs still look globally plausible, these errors may not be captured by page-level metrics. This suggests the need for complementary evaluation settings that use controlled edits and localized perturbations, so grounding failures can be observed directly.

Toward mitigation: prompting, verification, and training. Our findings point to several complementary directions for reducing pattern-completion bias in practice. First, the RQ3 results show that even moderate reasoning effort halves bias for ChatGPT-5.3 (93.2% → 40.8%), **suggesting that prompts which explicitly instruct the model to examine each element individually, rather than requesting a single holistic completion, could reduce default-to-pattern behavior**. Our qualitative analysis (Section 4.4) shows that models already perform this reasoning internally but then override it; prompts that penalize or flag self-contradiction in the reasoning trace could help close this gap. Second, a lightweight post-generation verification pass, re-rendering the generated code and comparing the target element's computed CSS value against the screenshot, could catch the most egregious errors without requiring changes to the model itself. Third, longer-term training interventions such as reinforcement learning from visual feedback, where the reward signal is derived from pixel-level comparison of the generated code's rendering against the input screenshot, could directly penalize outputs that are pattern consistent but visually incorrect. Curriculum learning [6] as a training technique that exposes models to progressively subtler deviations during fine-tuning may also help lower the saliency threshold at which grounding breaks down.

Implications for professional adoption. Screenshot-to-code tools are no longer experimental—they are embedded in the workflows of a large fraction of professional developers. Our findings imply that developers using these tools for generating code from pictures depicting real web components, should treat the generated output as a *draft* that preserves global structure but may silently normalize fine-grained styling decisions. This, holds relevance, especially in contexts where pixel-level fidelity matters, such as design systems with strict spacing tokens, accessibility-sensitive layouts, or brand-compliant interfaces. Teams relying on screenshot-to-code pipelines would benefit from integrating automated visual regression checks that flag discrepancies between the input screenshot and the rendered output at the element level.

Looking ahead, the pattern conformity phenomenon, where models derive the correct answer and then discard it in favor of pattern conformity, points to a deeper tension in how current *MLLMs* balance prior knowledge against input evidence. Our benchmark provides the first controlled, reproducible setting in which this tension can be measured and tracked as models improve. As screenshot-to-code technology matures and software engineering workflows grow increasingly automated, we hope this framework encourages testing procedures that push multimodal models toward faithful visual understanding, capable of operating on entire codebases, rather than toward more confident pattern completion.

Acknowledgments

This work was supported in part by NSF grant IIS-2533367 and NSF grant CCF-245105. The views expressed herein are the authors' own and do not necessarily reflect those of the sponsors.

Data Availability

We made all study artifacts, including the dataset, source code, and documentation, publicly accessible on Zenodo [27].

References

[1] Anthropic. 2026. *Claude Code overview.* https://code.claude.com/docs/en/overview Claude Code Docs. Accessed: 2026-03-09.

[2] Anthropic. 2026. Introducing Claude Opus 4.6. https://www.anthropic.com/news/claude-opus-4-6. https://www.anthropic.com/news/claude-opus-4-6 Accessed: 2026-03-26.

[3] Anthropic. 2026. Introducing Claude Sonnet 4.6. https://www.anthropic.com/news/claude-sonnet-4-6. https://www.anthropic.com/news/claude-sonnet-4-6 Accessed: 2026-03-26.

[4] Rabiul Awal, Mahsa Massoud, Aarash Feizi, Zichao Li, Suyuchen Wang, Christopher Pal, Aishwarya Agrawal, David Vazquez, Siva Reddy, Juan A Rodriguez, et al. 2025. Webmmu: A benchmark for multimodal multilingual website understanding and code generation. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*. 25129–25156. https://doi.org/10.18653/v1/2025.emnlp-main.1276

[5] Tony Beltramelli. 2018. pix2code: Generating code from a graphical user interface screenshot. In *Proceedings of the ACM SIGCHI symposium on engineering interactive computing systems*. 1–6. https://doi.org/10.1145/3220134.3220135

[6] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*. 41–48. https://doi.org/10.1145/1553374.1553380

[7] Nitzan Bitton-Guetta, Yonatan Bitton, Jack Hessel, Ludwig Schmidt, Yuval Elovici, Gabriel Stanovsky, and Roy Schwartz. 2023. Breaking common sense: Whoops! a vision-and-language benchmark of synthetic and compositional images. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2616–2627. https://doi.org/10.1109/iccv51070.2023.00247

[8] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374* (2021). https://doi.org/10.48550/arXiv.2107.03374

[9] Google. 2026. *Gemini Code Assist overview.* https://developers.google.com/gemini-code-assist/docs/overview Google Developers Documentation. Accessed: 2026-03-09.

[10] Google DeepMind. 2025. Gemini 3 Flash: Frontier Intelligence Built for Speed. https://blog.google/products/gemini/gemini-3-flash/. https://blog.google/products/gemini/gemini-3-flash/ Accessed: 2026-03-26.

[11] Tianrui Guan, Fuxiao Liu, Xiyang Wu, Ruiqi Xian, Zongxia Li, Xiaoyu Liu, Xijun Wang, Lichang Chen, Furong Huang, Yaser Yacoob, et al. 2024. Hallusionbench: an advanced diagnostic suite for entangled language hallucination and visual illusion in large vision-language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 14375–14385. https://doi.org/10.1109/cvpr52733.2024.01363

[12] Yi Gui, Zhen Li, Yao Wan, Yemin Shi, Hongyu Zhang, Yi Su, Bohua Chen, Siyuan Wu, Xing Zhou, Wenbin Jiang, Hai Jin, and Xiangliang Zhang. 2024. WebCode2M: A Real-World Dataset for Code Generation from Webpage Designs. *arXiv preprint arXiv:2404.06369* (2024). https://doi.org/10.48550/arXiv.2404.06369

[13] Yi Gui, Yao Wan, Zhen Li, Zhongyi Zhang, Dongping Chen, Hongyu Zhang, Yi Su, Bohua Chen, Xing Zhou, Wenbin Jiang, and Xiangliang Zhang. 2025. UICopilot: Automating UI Synthesis via Hierarchical Code Generation from Webpage Designs. *arXiv preprint arXiv:2505.09904* (2025). https://doi.org/10.48550/arXiv.2505.09904

[14] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. 2024. Large language models for software engineering: A systematic literature review. *ACM Transactions on Software Engineering and Methodology* 33, 8 (2024), 1–79. https://doi.org/10.1145/3695988

[15] Dong Huang, Jie M. Zhang, Qingwen Bu, Xiaofei Xie, Junjie Chen, and Heming Cui. 2025. Bias Testing and Mitigation in LLM-based Code Generation. https://doi.org/10.48550/arXiv.2309.14345 arXiv:2309.14345 [cs.SE]

[16] Wen Huang, Hongbin Liu, Minxin Guo, and Neil Gong. 2024. Visual Hallucinations of Multi-modal Large Language Models. In *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, Lun-Wei Ku, Andre Martins, and Vivek Srikumar (Eds.). Association for Computational Linguistics, 9614–9631. https://doi.org/10.18653/V1/2024.FINDINGS-ACL.573

[17] Lingjie Jiang, Shaohan Huang, Xun Wu, Yixia Li, Dongdong Zhang, and Furu Wei. 2025. Viscodex: Unified multimodal code generation via merging vision and coding models. *arXiv preprint arXiv:2508.09945* (2025). https://doi.org/10.48550/arXiv.2508.09945

[18] Yilei Jiang, Yaozhi Zheng, Yuxuan Wan, Jiaming Han, Qunzhong Wang, Michael R Lyu, and Xiangyu Yue. 2025. Screencoder: Advancing visual-to-code generation for front-end automation via modular multimodal agents. *arXiv preprint arXiv:2507.22827* (2025). https://doi.org/10.48550/arXiv.2507.22827

[19] Hugo Laurençon, Léo Tronchon, and Victor Sanh. 2024. Unlocking the Conversion of Web Screenshots into HTML Code with the WebSight Dataset. *arXiv preprint arXiv:2403.09029* (2024). https://doi.org/10.48550/arXiv.2403.09029

[20] Kenton Lee, Mandar Joshi, Iulia Raluca Turc, Hexiang Hu, Fangyu Liu, Julian Martin Eisenschlos, Urvashi Khandelwal, Peter Shaw, Ming-Wei Chang, and Kristina Toutanova. 2023. Pix2struct: Screenshot parsing as pretraining for visual language understanding. In *International Conference on Machine Learning*. PMLR, 18893–18912.

[21] Kang-il Lee, Minbeom Kim, Seunghyun Yoon, Minsung Kim, Dongryeol Lee, Hyukhun Koh, and Kyomin Jung. 2025. Vlind-bench: Measuring language priors in large vision-language models. In *Findings of the Association for Computational Linguistics: NAACL 2025*. 4129–4144. https://doi.org/10.18653/v1/2025.findings-naacl.231

[22] 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* (2025). https://doi.org/10.48550/arXiv.2506.07818

[23] Lin Ling, Fazle Rabbi, Song Wang, and Jinqiu Yang. 2025. Bias Unveiled: Investigating Social Bias in LLM-Generated Code. https://doi.org/10.48550/arXiv.2411.10351 arXiv:2411.10351 [cs.SE]

[24] Jiazhen Liu, Yuhan Fu, Ruobing Xie, Runquan Xie, Xingwu Sun, Fengzong Lian, Zhanhui Kang, and Xirong Li. 2024. Phd: A chatgpt-prompted visual hallucination evaluation dataset. *arXiv preprint arXiv:2403.11116* (2024). https://doi.org/10.48550/arXiv.2403.11116

[25] Yan Liu, Xiaokang Chen, Yan Gao, Zhe Su, Fengji Zhang, Daoguang Zan, Jian-Guang Lou, Pin-Yu Chen, and Tsung-Yi Ho. 2023. Uncovering and quantifying social biases in code generation. *Advances in Neural Information Processing Systems* 36 (2023), 2368–2380. https://doi.org/10.52202/075280-0110

[26] Microsoft. 2020. *Playwright: Fast and Reliable End-to-End Testing for Modern Web Apps.* Retrieved August 4, 2026 from https://playwright.dev/

[27] Khai-Nguyen Nguyen, Oscar Chaparro, and Antonio Mastropaolo. 2026. Pattern2Code Replication Package. https://doi.org/10.5281/zenodo.19341952

[28] OpenAI. 2025. *https://openai.com/index/introducing-codex/.* https://openai.com/index/introducing-codex/

[29] OpenAI. 2026. GPT-5.3 Instant: Smoother, More Useful Everyday Conversations. https://openai.com/index/gpt-5-3-instant/. https://openai.com/index/gpt-5-3-instant/ Accessed: 2026-03-26.

[30] OpenAI. 2026. Introducing GPT-5.3-Codex. https://openai.com/index/introducing-gpt-5-3-codex/. https://openai.com/index/introducing-gpt-5-3-codex/ Accessed: 2026-03-26.

[31] OpenRouter. 2026. *OpenRouter.* https://openrouter.ai/ Accessed: 2026-03-20.

[32] Letitia Parcalabescu, Michele Cafagna, Lilitta Muradjan, Anette Frank, Iacer Calixto, and Albert Gatt. 2022. VALSE: A Task-Independent Benchmark for Vision and Language Models Centered on Linguistic Phenomena. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (Eds.). Association for Computational Linguistics, Dublin, Ireland, 8253–8280. https://doi.org/10.18653/v1/2022.acl-long.567

[33] Haz Sameen Shahgir, Khondker Salman Sayeed, Abhik Bhattacharjee, Wasi Uddin Ahmad, Yue Dong, and Rifat Shahriyar. 2024. IllusionVQA: A challenging optical illusion dataset for vision language models. *arXiv preprint arXiv:2403.15952* (2024). https://doi.org/10.48550/arXiv.2403.15952

[34] 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)*. 3956–3974. https://doi.org/10.18653/v1/2025.naacl-long.199

[35] Stack Overflow. 2025. *Technology — 2025 Stack Overflow Developer Survey.* https://survey.stackoverflow.co/2025/technology#most-popular-technologies-ai-models-ai-models-prof Accessed: 2026-03-20.

[36] Haoyu Sun, Huichen Will Wang, Jiawei Gu, Linjie Li, and Yu Cheng. 2025. FullFront: Benchmarking MLLMs Across the Full Front-End Engineering Workflow. *arXiv preprint arXiv:2505.17399* (2025). https://doi.org/10.48550/arXiv.2505.17399

[37] Shengbang Tong, Zhuang Liu, Yuexiang Zhai, Yi Ma, Yann LeCun, and Saining Xie. 2024. Eyes wide shut? exploring the visual shortcomings of multimodal llms. In *CVPR*. https://doi.org/10.1109/cvpr52733.2024.00914

[38] An Vo, Khai-Nguyen Nguyen, Mohammad Reza Taesiri, Vy Tuong Dang, Anh Totti Nguyen, and Daeyoung Kim. 2025. Vision Language Models are Biased. *CoRR* abs/2505.23941 (2025). https://doi.org/10.48550/ARXIV.2505.23941 arXiv:2505.23941

[39] Yuxuan Wan, Chaozheng Wang, Yi Dong, Wenxuan Wang, Shuqing Li, Yintong Huo, and Michael R. Lyu. 2024. Automatically Generating UI Code from Screenshot: A Divide-and-Conquer-Based Approach. *arXiv preprint arXiv:2406.16386* (2024). https://doi.org/10.48550/arXiv.2406.16386

[40] Junjie Wang, Yuchao Huang, Chunyang Chen, Zhe Liu, Song Wang, and Qing Wang. 2024. Software testing with large language models: Survey, landscape, and vision. *IEEE Transactions on Software Engineering* 50, 4 (2024), 911–936. https://doi.org/10.1109/tse.2024.3368208

[41] Jules White, Sam Hays, Quchen Fu, Jesse Spencer-Smith, and Douglas C Schmidt. 2024. Chatgpt prompt patterns for improving code quality, refactoring, requirements elicitation, and software design. In *Generative ai for effective software development*. Springer, 71–108. https://doi.org/10.1007/978-3-031-55642-5_4

[42] Jason Wu, Eldon Schoop, Alan Leung, Titus Barik, Jeffrey P. Bigham, and Jeffrey Nichols. 2024. UICoder: Finetuning Large Language Models to Generate User Interface Code through Automated Feedback. *arXiv preprint arXiv:2406.07739* (2024). https://doi.org/10.48550/arXiv.2406.07739

[43] Chunqiu Steven Xia, Yuxiang Wei, and Lingming Zhang. 2023. Automated program repair in the era of large pre-trained language models. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 1482–1494. https://doi.org/10.1109/icse48619.2023.00129

[44] Jingyu Xiao, Yuxuan Wan, Yintong Huo, Zixin Wang, Xinyi Xu, Wenxuan Wang, Zhiyao Xu, Yuhang Wang, and Michael R. Lyu. 2024. Interaction2Code: Benchmarking MLLM-based Interactive Webpage Code Generation from Interactive Prototyping. *arXiv preprint arXiv:2411.03292* (2024). https://doi.org/10.48550/arXiv.2411.03292

[45] Jingyu Xiao, Ming Wang, Man Ho Lam, Yuxuan Wan, Junliang Liu, Yintong Huo, and Michael R. Lyu. 2025. DesignBench: A Comprehensive Benchmark for MLLM-based Front-end Code Generation. *arXiv preprint arXiv:2506.06251* (2025). https://doi.org/10.48550/arXiv.2506.06251

[46] Shuhong Xiao, Yunnong Chen, Jiazhi Li, Liuqing Chen, Lingyun Sun, and Tingting Zhou. 2024. Prototype2Code: End-to-end Front-end Code Generation from UI Design Prototypes. *arXiv preprint arXiv:2405.04975* (2024). https://doi.org/10.48550/arXiv.2405.04975

[47] Zhen Yang, Wenyi Hong, Mingde Xu, Xinyue Fan, Weihan Wang, Jiele Cheng, Xiaotao Gu, and Jie Tang. 2025. UI2Codeˆ N: A Visual Language Model for Test-Time Scalable Interactive UI-to-Code Generation. *arXiv preprint arXiv:2511.08195* (2025). https://doi.org/10.48550/arXiv.2511.08195

[48] Moon Ye-Bin, Nam Hyeon-Woo, Wonseok Choi, and Tae-Hyun Oh. 2024. Beaf: Observing before-after changes to evaluate hallucination in vision-language models. In *European Conference on Computer Vision*. Springer, 232–248. https://doi.org/10.1007/978-3-031-73247-8_14

[49] Sukmin Yun, Rusiru Thushara, Mohammad Bhat, Yongxin Wang, Mingkai Deng, Jinhong Wang, Tianhua Tao, Junbo Li, Haonan Li, Preslav Nakov, et al. 2024. Web2code: A large-scale webpage-to-code dataset and evaluation framework for multimodal llms. *Advances in neural information processing systems* 37 (2024), 112134–112157. https://doi.org/10.52202/079017-3560

[50] Kankan Zhou, Eason Lai, Wei Bin Au Yeong, Kyriakos Mouratidis, and Jing Jiang. 2023. ROME: Evaluating Pre-trained Vision-Language Models on Reasoning beyond Visual Common Sense. In *Findings of the Association for Computational Linguistics: EMNLP*. https://doi.org/10.18653/v1/2023.findings-emnlp.683

[51] Ting Zhou, Yanjie Zhao, Xinyi Hou, Xiaoyu Sun, Kai Chen, and Haoyu Wang. 2024. Bridging Design and Development with Automated Declarative UI Code Generation. *arXiv preprint arXiv:2409.11667* (2024). https://doi.org/10.48550/arXiv.2409.11667

[52] Hongda Zhu, Yiwen Zhang, Bing Zhao, Jingzhe Ding, Siyao Liu, Tong Liu, Dandan Wang, Yanan Liu, and Zhaojian Li. 2025. FrontendBench: A Benchmark for Evaluating LLMs on Front-End Development via Automatic Evaluation. *arXiv preprint arXiv:2506.13832* (2025). https://doi.org/10.48550/arXiv.2506.13832

Received 2026-03-27; accepted 2026-06-18