ABPQ: AUTOMATED CONSTRUCTION OF CHALLENGING UI-TO-CODE BENCHMARKS VIA ROUND-TRIP CONSISTENCY

Anonymous authors
Paper under double-blind review

ABSTRACT

Existing UI-to-Code benchmarks face a tradeoff between evaluation comprehensiveness and construction scalability. Benchmarks containing paired UI-HTML data support fine grained objective evaluation but are costly to construct and are becoming increasingly saturated, whereas benchmarks containing only UI screenshots are easier to scale but typically rely on LLM-as-Judge. To address this issue, we propose PQ Score, which identifies samples that are challenging for models without requiring reference code by measuring information loss during the "screenshot $\rightarrow$ code $\rightarrow$ rendered screenshot $\rightarrow$ code" round trip process. Based on PQ Score, we further propose the ABPQ benchmark construction pipeline with minimal reliance on human effort and use it to construct ABPQ-FULL, containing 500 samples, and its challenging subset ABPQ-HARD, containing 200 samples, from 4,771,476 real UI screenshots. The final samples retain their corresponding HTML, supporting multiple automatic metrics, LLM-as-Judge evaluation, and human evaluation. Experiments on ten representative proprietary and open source models show that ABPQ is more challenging than representative existing benchmarks on nearly all metrics and that current advanced models remain far from saturation. In addition, PQ Score exhibits strong positive correlations with other automatic metrics, validating its effectiveness for difficult sample filtering. ABPQ provides a new approach to constructing scalable, continually updated UI-to-Code benchmarks with minimal reliance on human effort.

1 INTRODUCTION

UI-to-Code aims to automatically generate executable frontend code from UI screenshots and is an important task that connects visual design with software implementation (Laurençon et al., 2024). It can not only improve the efficiency of interface development and prototype iteration, but also comprehensively evaluate the visual understanding, spatial reasoning, and code generation capabilities of multimodal models (Wan et al., 2025; Jiang et al., 2025). However, existing UI-to-Code benchmarks still have clear limitations. Benchmarks containing paired UI-HTML data enable the computation of fine grained objective metrics using reference code, but are costly to collect and filter and remain limited in scale and difficulty (Si et al., 2025). As shown in Table 1, advanced models already achieve near-saturated performance even on specially constructed HARD subsets. Another emerging category of benchmarks contains only UI screenshots, making data acquisition easier and enabling higher difficulty (He et al., 2026). However, due to the lack of reference code, these benchmarks usually rely solely on LLM-as-Judge for evaluation and may be affected by judge preferences. More importantly, the construction and quality control of both types of benchmarks rely heavily on human effort, and the resulting benchmarks generally remain static once constructed. As model capabilities improve, rebuilding more challenging datasets will again incur substantial costs.

To address these issues, we propose PQ Score and an ABPQ benchmark construction pipeline centered on it. PQ Score evaluates a model's ability to understand and reconstruct UIs from the perspective of round trip consistency. The model first generates code from a screenshot and renders it, and then generates code again from the rendered result. By measuring the loss of UI layout structure information during this process, PQ Score determines how difficult a sample is for the model. Building on this metric, the ABPQ pipeline can automatically identify difficult samples for current

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Table 1: Performance of representative models on Design2Code-HARD. Despite its hard-sample construction, scores remain high, indicating near-saturation among current advanced models. All metrics range from 0 to 100; higher is better.

Model	Text	Block	Position	Color
GPT-5.6-Sol High	97.7	83.7	94.2	97.0
Claude-Opus-4.7	97.0	83.1	92.9	87.8
Claude-Opus-5	96.3	84.4	92.8	90.5
Gemini-3.7-Flash	97.0	77.9	90.0	92.5
Doubao-Seed-2.0-Pro	96.4	74.3	90.6	90.0
Qwen3.5-397B-A17B	98.0	78.1	87.8	90.6
Kimi-K3	97.8	87.5	93.4	92.0

models from large scale screenshot collections, requires only minimal human involvement, and can be continually updated as model capabilities advance. The pipeline itself does not require paired HTML and can therefore leverage the large number of UI screenshots without code annotations available in real world environments.

We apply this pipeline to 4,771,476 real UI screenshots and construct ABPQ-FULL with 500 samples and its challenging subset ABPQ-HARD with 200 samples. Although the construction pipeline uses only screenshots, we still provide the corresponding HTML for the final samples, thereby supporting objective metrics based on code, LLM-as-Judge evaluation, and human evaluation. Experiments on ten representative proprietary and open source models show that current advanced models remain far from saturation on the ABPQ benchmark and that ABPQ is more challenging than existing benchmarks on nearly all metrics.

Our main contributions are as follows:

1. We propose PQ Score, a UI-to-Code evaluation metric that requires no reference code and is based on round trip consistency. It can effectively identify samples that are challenging for models.
2. We propose an automated benchmark construction pipeline centered on PQ Score, which substantially reduces reliance on human effort and enables difficult samples to be efficiently mined from large scale screenshot collections and continually updated.
3. We construct ABPQ-FULL and ABPQ-HARD to support comprehensive evaluation across multiple dimensions. Experiments show that they are significantly more challenging than existing benchmarks and that current advanced models still have substantial room for improvement.

2 Related Work

LLM-Based Code Generation. As code data have been incorporated into training, general purpose large language models have gradually demonstrated capabilities in code completion and generation (Achiam et al., 2023), and have further acquired capabilities in complex reasoning, codebase understanding, and multi file modification (Liu et al., 2024; Zhang et al., 2024). On this basis, programming systems have begun to introduce task planning (Jiang et al., 2024), tool calling (Zhang et al., 2023), and execution feedback (Wang et al., 2024), forming coding agents that can iteratively complete software development tasks. Multimodal large language models have further extended code generation capabilities to UI-to-Code. Their applications have developed from early direct generation of code from screenshots (Gui et al., 2025a) to the use of layout decomposition and hierarchical strategies for handling complex interfaces (Wan et al., 2025; Gui et al., 2025b), and further to multiround iterative optimization through rendering feedback (Yang et al., 2025). Although specific methods continue to evolve, the core of UI-to-Code capabilities remains the direct understanding of UI screenshots and the generation of corresponding executable code.

UI-to-Code Datasets and Benchmarks. The development of UI-to-Code generation has benefited from datasets and benchmarks that map visual layouts to executable code. In terms of datasets, real world datasets (Gui et al., 2025a) collect data from real webpages and perform extensive pruning and filtering. Synthetic datasets (Laurençon et al., 2024; Yun et al., 2024) use templates and other methods to synthesize data at scale. The former ensures data authenticity, while the latter ensures a substantial data volume. In terms of benchmarks, early benchmarks (Si et al., 2025) were constructed from real UI-HTML pairs and evaluated models through code comparison and rendering quality. Because standalone image resources are easier to obtain than real UI-HTML pairs, and LLM-as-Judge has become increasingly viable, recent benchmarks (Yang et al., 2025; He et al., 2026) have gradually shifted toward evaluation based on single images and LLM-as-Judge.

3 ABPQ

3.1 PQ Score

We propose PQ Score, a metric for evaluating the UI-to-Code capabilities of models. The metric assesses a model's structural understanding of a given UI and its ability to reconstruct the corresponding code by measuring round trip consistency. Given an input UI screenshot A, the model first generates HTML code P. It then renders P into a page screenshot B and generates HTML code Q from B. Although the same UI can be implemented with different code, if the model can consistently understand and reconstruct the interface, P and Q should ideally preserve similar layout hierarchies and DOM structures after the "screenshot $\rightarrow$ code $\rightarrow$ screenshot $\rightarrow$ code" round trip, with only limited loss of structural information. Based on this observation, we measure the model's round trip consistency by comparing the DOM tag trees constructed from P and Q after removing details unrelated to the visible structure of the page. Specifically, we use the ordered tree edit distance to measure the structural similarity between the two DOM trees:

$$S_{\mathrm{DOM}}(P, Q) = \max\left(0,\, 1 - \frac{\mathrm{TED}(T_P, T_Q)}{\max\left(|T_P|, |T_Q|\right)}\right). \tag{1}$$

where T_P and T_Q are the DOM tag trees constructed from P and Q, respectively, TED denotes the edit distance between the two trees, and $|T_P|$ and $|T_Q|$ denote their respective numbers of nodes. PQ Score is then defined as:

$$\text{PQ Score}(P, Q) = 100 \times S_{\mathrm{DOM}}(P, Q). \tag{2}$$

PQ Score ranges from 0 to 100. A higher score indicates greater consistency between the DOM structures of the two generated code outputs, meaning that the model preserves more information about the UI layout structure during the round trip process and demonstrates stronger UI understanding and code reconstruction capabilities. Conversely, a lower score indicates a greater loss of structural information, suggesting that the UI poses a greater generation challenge for the model. Further details on the calculation of PQ Score are provided in Appendix A.

3.2 ABPQ Pipeline

As shown in Figure 1, the ABPQ pipeline consists of three stages: Web-Scale Screenshot Collection, Hierarchical PQ Score Filtering, and Quality Control & Selection. The main text focuses on the overall pipeline and its key design choices, with further details provided in Appendix B.

3.2.1 Web-Scale Screenshot Collection.

We first collected approximately 7 million UI-HTML samples. It is worth noting that the paired HTML was collected not to support sample filtering, but to provide reference code for the final benchmark, enabling a more comprehensive evaluation using objective metrics based on reference code rather than relying solely on LLM-as-Judge. In fact, the ABPQ construction pipeline itself requires only UI screenshots and can therefore be directly applied to large volumes of image data collected automatically, whose acquisition cost is significantly lower than that of paired UI-HTML data. For the collected samples, we first performed deduplication based on image hashes and removed invalid screenshots that were almost entirely black or white using rule based filtering. Following WebCode2M (Gui et al., 2025a), we then filtered the screenshots using the scores produced

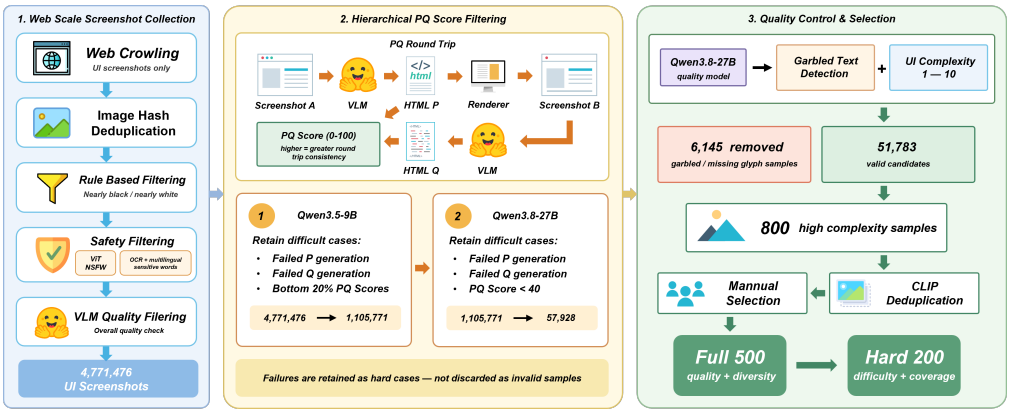
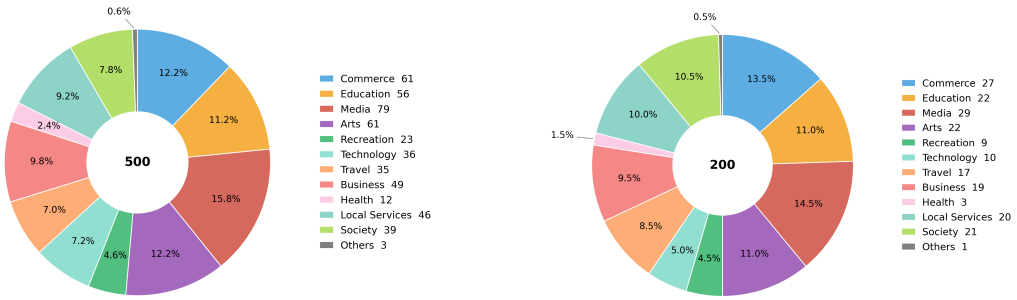


Figure 1: Overview of ABPQ. ABPQ collects web-scale UI screenshots, mines challenging samples through hierarchical PQ Score filtering, and performs quality control with minimal human involvement to construct ABPQ-FULL and ABPQ-HARD.



(a) ABPQ-FULL **(b)** ABPQ-HARD

Figure 2: Distributions of ABPQ-FULL and ABPQ-HARD across 12 domains. Both sets provide broad and balanced coverage without a dominant domain, while ABPQ-HARD closely preserves the distribution of ABPQ-FULL despite its smaller size.

by a ViT NSFW classifier together with the frequencies of multilingual sensitive words in the text extracted by a lightweight OCR model. Finally, we used Qwen3.5-4B (Qwen Team, 2026a) to conduct a comprehensive assessment of the pages. After these steps, 4,771,476 samples were retained.

3.2.2 Hierarchical PQ Score Filtering.

In principle, PQ Score could be computed once using the strongest available model. However, doing so for millions of samples would incur prohibitive computational costs. We therefore adopt a hierarchical filtering strategy that progresses from smaller to larger models, gradually focusing on more challenging samples while controlling the computational cost. In the first stage, Qwen3.5-9B processes all 4,771,476 screenshots. We retain three categories of samples: samples for which P cannot be generated, samples for which P can be generated but Q cannot be generated from the corresponding rendered screenshot B, and samples with PQ Scores below 59.1, corresponding to the bottom 20%. Importantly, generation failures are treated as difficult samples that reflect limitations in the model's capabilities, rather than as invalid data to be discarded. After the first stage, 1,105,771 samples are retained. In the second stage, we use the stronger Qwen3.8-27B (Qwen Team, 2026b) to rerun the ABPQ process on these samples. We continue to retain samples for which either P or Q cannot be generated, as well as samples with a PQ Score below 40, resulting in 57,928 highly challenging samples.

3.2.3 QUALITY CONTROL & SELECTION.

After obtaining the challenging candidate set, we further use Qwen3.8-27B (Qwen Team, 2026b) for quality control. The model identifies screenshots containing large amounts of garbled text and assigns each valid UI screenshot a complexity level from 1 to 10. This step removes 6,145 samples containing garbled text, leaving 51,783 valid samples. From these samples, we select 800 samples with high complexity and perform deduplication based on CLIP similarity (Radford et al., 2021). The resulting set is then manually reviewed by reviewers, all of whom hold PhD degrees in computer science. We retain 500 samples deemed suitable for evaluation, forming ABPQ-FULL. From ABPQ-FULL, we further select ABPQ-HARD, a set of 200 samples chosen while preserving coverage as much as possible.

3.3 BENCHMARK STATISTICS

We analyze benchmark diversity along three dimensions: domain coverage, page type, and user task. ABPQ-FULL and ABPQ-HARD cover 12 domains, 17 page types, and 19 user tasks, reflecting substantial diversity in content and functionality. Figure 2 presents the distributions across domains, where both sets exhibit broad and reasonably balanced coverage, with no single domain dominating. The distributions across page types and user tasks exhibit a similar degree of diversity and are presented in Appendix C. Despite containing fewer samples, ABPQ-HARD closely matches the distributions of ABPQ-FULL. This consistency allows ABPQ-HARD to serve as a reliable and efficient evaluation set with minimal distribution bias relative to ABPQ-FULL.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Models. We evaluate five proprietary models: GPT-5.6-Sol with the reasoning effort set to High, Claude-Opus-4.7, Claude-Opus-5, Gemini-3.7-Flash, and Doubao-Seed-2.0-Pro. We also evaluate five open source models of varying sizes: Ministral-3-14B-2512 (Liu et al., 2026), Qwen3.8-27B (Qwen Team, 2026b), InternVL3.5-38B (Wang et al., 2025), Qwen3.5-397B-A17B (Qwen Team, 2026a), and Kimi-K3 (2.8T) (Team et al., 2026).

Metrics. For automatic evaluation, we use the Block, Text, Position, and Color metrics from Design2Code (Si et al., 2025), the Visual Score (VS) from Vision2Web (He et al., 2026), and our proposed PQ Score. To reduce bias from any single judge when evaluating VS, we use three judge models: GPT-5.6-Sol (High), Claude-Opus-5, and Gemini-3.7-Flash. All automatic evaluation metrics range from 0 to 100. For human evaluation, each sample is independently rated by three reviewers with PhD degrees in computer science on a scale from 0 to 5 in terms of content completeness, layout, and style. The average score of the three reviewers is used as the final result. Further details are provided in Appendix D.

4.2 EXPERIMENTAL RESULTS

4.2.1 AUTOMATIC EVALUATION

Table 2 reports the evaluation results of each model on ABPQ-HARD, while the complete results on ABPQ-FULL are provided in Appendix E. Both sets of results support the same conclusions. Even current leading open source and proprietary models remain far from saturation on several metrics. The best VS-GPT, Position, Color, and PQ Score achieved by any model are only 49.5, 57.1, 54.9, and 55.9, respectively, indicating substantial room for improvement in overall visual quality, element positioning, color reproduction, and page layout. Notably, different models excel on different metrics. For example, GPT-5.6-Sol achieves the best performance on VS-GPT and VS-Opus, whereas Claude-Opus-5 performs best on Block and PQ Score. This suggests that no single metric can fully characterize UI-to-Code capabilities. Multidimensional evaluation is therefore necessary to reveal the strengths and limitations of different models, rather than relying solely on LLM-as-Judge.

Table 2: Automatic evaluation results on ABPQ-HARD. We compare five closed-source and five open-source models using three VS judges, four Design2Code metrics, and PQ Score. Higher is better, and bold indicates the best result within each model group.

Model	VS-GPT	VS-Opus	VS-Gemini	Text	Block	Position	Color	PQ Score
Closed-source								
GPT-5.6-Sol High	**49.5**	**62.6**	**79.2**	97.1	69.4	53.5	50.3	44.6
Claude-Opus-5	39.5	61.3	77.8	97.4	**83.2**	54.3	51.8	**55.9**
Claude-Opus-4.7	32.2	48.9	62.1	**97.7**	80.1	54.7	50.5	54.6
Gemini-3.7-Flash	31.8	53.0	76.0	96.5	76.9	54.3	**52.9**	53.2
Doubao-Seed-2.0-Pro	32.0	47.4	66.0	97.1	81.3	**55.3**	51.0	49.0
Open-source								
Ministral-3-14B-2512	25.6	33.2	41.7	91.7	69.9	52.7	48.8	43.2
Qwen3.8-27B	29.4	44.0	59.4	90.0	70.7	51.7	45.6	42.4
InternVL3.5-38B	20.7	22.7	27.7	95.2	80.5	**57.1**	**54.9**	47.0
Qwen3.5-397B-A17B	32.8	49.1	69.0	**98.1**	**80.7**	55.2	49.9	**52.7**
Kimi-K3	**40.9**	**60.0**	**81.2**	92.5	75.7	51.2	46.6	49.4

4.2.2 HUMAN EVALUATION

Table 3 reports the human evaluation results on ABPQ-HARD. Although the three reviewers differ in their scoring scales, their judgments of the overall model ranking are largely consistent. GPT-5.6-Sol (High) and Claude-Opus-5 rank first and second, respectively, according to all three reviewers; Qwen3.5-397B-A17B, Gemini-3.7-Flash, and Doubao-Seed-2.0-Pro form the upper middle tier; and InternVL3.5-38B consistently ranks last. Notably, the best performing model, GPT-5.6-Sol, achieves an average score of only 3.24, slightly above a score of 3, which is defined as "can partially serve as a replacement." InternVL3.5-38B scores only 1.62, falling between scores of 1 and 2, which are defined as "completely unable to replace the original page" and "can replace the original page to a limited extent," respectively. This further demonstrates that even the most advanced current models struggle to generate results on ABPQ that can adequately replace the original pages.

Table 3: Human evaluation results on ABPQ-HARD. Each model is independently rated by three reviewers on a 0–5 scale. Despite different scoring scales, the reviewers produce consistent model rankings. Even the best model achieves only 3.24 on average, showing that current models remain far from fully replacing the original pages. Higher is better.

Model	Reviewer 1	Reviewer 2	Reviewer 3	Average
GPT-5.6-Sol High	3.59	2.92	3.20	3.24
Claude-Opus-5	3.10	2.45	2.47	2.67
Qwen3.5-397B-A17B	2.77	2.28	2.02	2.36
Gemini-3.7-Flash	2.65	2.22	2.17	2.35
Doubao-Seed-2.0-Pro	2.59	2.22	2.05	2.29
Claude-Opus-4.7	2.68	2.14	1.68	2.17
Qwen3.8-27B	2.53	2.08	1.78	2.13
Kimi-K3	2.31	1.77	1.90	1.99
Ministral-3-14B-2512	2.39	2.00	1.37	1.92
InternVL3.5-38B	2.11	1.69	1.06	1.62

5 ANALYSIS

5.1 CORRELATION BETWEEN PQ SCORE AND OTHER METRICS

We use the Pearson correlation coefficient to measure the correlations between PQ Score and the other metrics, as it is well suited for determining whether two sets of scores exhibit a linear relationship. To obtain more stable statistical estimates, we perform the calculation on the larger ABPQ-FULL, with the corresponding results for ABPQ-HARD provided in Appendix F. As shown in Table 4, PQ Score is strongly positively correlated with all metrics ($r = 0.66$–0.87), exhibiting the highest correlations with Text and Block at 0.87 and 0.82, respectively, while maintaining consistent

Table 4: Pearson correlations between PQ Score and other automatic evaluation metrics on ABPQ-FULL. All correlations are strongly positive, with the highest correlations observed for Text and Block among all evaluated metrics.

Metric	Pearson's r	Correlation Strength
VS-GPT	0.67	Strong
VS-Opus	0.78	Strong
VS-Gemini	0.75	Strong
Text	0.87	Very Strong
Block	0.82	Very Strong
Position	0.66	Strong
Color	0.68	Strong

Table 5: Spearman correlations between automatic metrics and human evaluation on ABPQ-HARD. VS metrics and Text align most strongly, while PQ Score shows moderate agreement.

Metric	Spearman's ρ	Correlation Strength
VS-Opus	0.80	Very Strong
VS-GPT	0.70	Strong
VS-Gemini	0.65	Strong
Text	0.66	Strong
PQ Score	0.42	Moderate
Block	0.19	Weak
Color	0.15	Weak
Position	0.07	Weak

Table 6: Cross-benchmark comparison of three representative models. ABPQ-HARD and ABPQ-FULL consistently yield the lowest scores on nearly all metrics across all three models, showing greater difficulty than Design2Code, Design2Code-HARD, and Vision2Web. Lower scores indicate greater difficulty; bold marks the lowest value in each column. "–" indicates that the metric is unavailable because Vision2Web does not provide the reference code required for its computation.

(a) GPT-5.6-Sol

Benchmark	VS-GPT	VS-Opus	VS-Gemini	Text	Block	Position	Color	PQ Score
Design2Code	58.9	73.8	89.6	98.7	94.3	97.4	97.2	71.8
Design2Code-HARD	52.8	72.8	88.8	97.7	83.7	94.2	97.0	74.4
Vision2Web	53.5	72.8	89.3	–	–	–	–	69.6
ABPQ-HARD	**49.5**	**62.6**	**79.2**	**97.1**	**69.4**	**53.5**	**50.3**	**44.6**
ABPQ-FULL	52.2	66.1	82.8	97.5	72.2	57.2	56.9	53.6

(b) Claude-Opus-5

Benchmark	VS-GPT	VS-Opus	VS-Gemini	Text	Block	Position	Color	PQ Score
Design2Code	44.6	67.4	85.3	98.6	94.5	95.9	91.3	76.1
Design2Code-HARD	40.6	65.8	85.1	**96.3**	84.4	92.8	90.5	70.8
Vision2Web	40.9	64.3	85.2	–	–	–	–	67.7
ABPQ-HARD	**39.5**	**61.3**	**77.8**	97.4	83.2	**54.3**	**51.8**	**55.9**
ABPQ-FULL	39.7	61.7	78.9	97.6	**81.7**	57.0	56.1	57.3

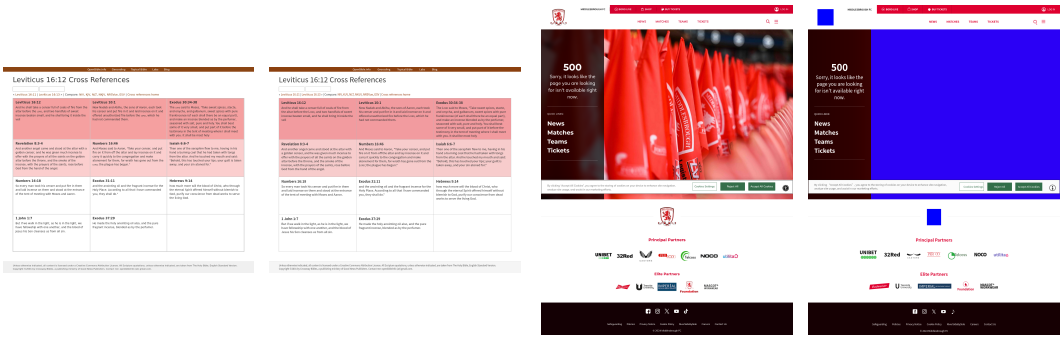
(c) Kimi-K3

Benchmark	VS-GPT	VS-Opus	VS-Gemini	Text	Block	Position	Color	PQ Score
Design2Code	47.5	69.3	89.0	98.6	94.2	96.8	93.9	72.4
Design2Code-HARD	41.6	65.4	87.2	97.8	87.5	93.4	92.0	71.5
Vision2Web	42.4	66.8	88.5	–	–	–	–	62.4
ABPQ-HARD	40.9	60.0	81.2	**92.5**	**75.7**	**51.2**	**46.6**	**49.4**
ABPQ-FULL	**39.8**	**58.9**	**79.2**	92.8	76.3	53.8	51.4	51.1

correlations with the three VS metrics, Position, and Color. This indicates that selecting difficult samples based on PQ Score also implicitly performs joint filtering across the dimensions captured by the other metrics.

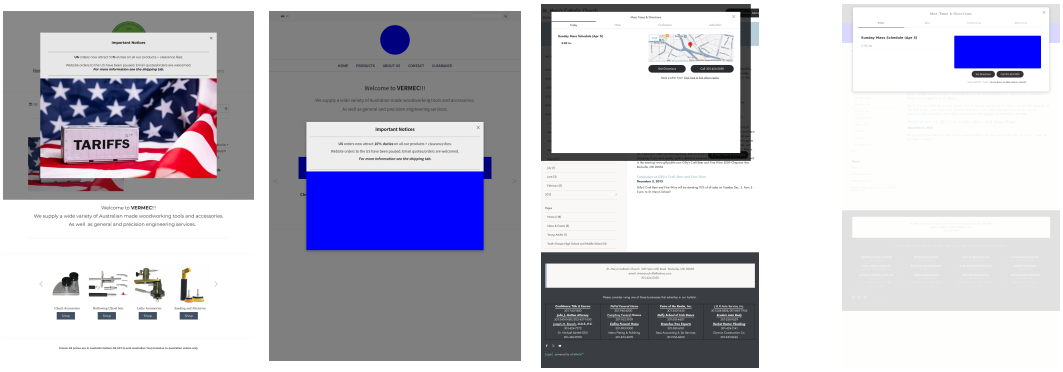
5.2 Alignment of PQ Score and Other Automatic Metrics with Human Evaluation

We use Spearman's rank correlation coefficient to measure the consistency between each automatic metric and human evaluation in terms of model rankings. This coefficient allows nonlinear monotonic relationships and is suitable for comparing model rankings produced by different evaluation methods. As shown in Table 5, the VS metrics obtained from different judges exhibit strong correlations with human evaluation ($\rho = 0.65$–0.80), as both assess the overall fidelity of the reconstructed page. Text also shows a strong correlation ($\rho = 0.66$), indicating that text completeness substantially influences human judgments of overall UI quality. PQ Score has a moderate correlation with human evaluation ($\rho = 0.42$): human evaluation considers visual factors such as content, style, and color, whereas PQ Score places greater emphasis on page layout and structural information, making the two complementary. In contrast, Block, Color, and Position measure only specific dimensions and therefore exhibit weaker correlations with overall human judgments ($\rho = 0.07$–0.19).



(a) PQ Score = 100.00

(b) PQ Score = 50.00



(c) PQ Score = 14.32

(d) PQ Score = 7.62

Figure 3: Qualitative comparisons across four examples. In each panel, the reference image is shown on the left and the result generated from P is shown on the right. As the PQ Score decreases, the visual discrepancy between A and B becomes increasingly pronounced. The examples also reveal common errors in layout and style reproduction, component counts, and page structure.

5.3 Difficulty Comparison with Representative Benchmarks

Table 6 compares ABPQ with Design2Code, Design2Code-HARD, and Vision2Web. In the main text, we present results for three representative models, GPT-5.6-Sol, Claude-Opus-5, and Kimi-K3, while the results for the remaining seven models are provided in Appendix G. ABPQ-HARD and ABPQ-FULL yield the lowest scores for nearly all combinations of models and metrics, and all three VS judges consistently rate ABPQ as the most challenging benchmark, indicating that its difficulty does not depend on a specific judge or a single metric. On ABPQ, the models generally achieve relatively high Block scores but lower Position, Color, and PQ Scores, suggesting that the models can recover coarse grained regions and major components but still struggle substantially to preserve precise positions, sizes, spacing, colors, and layout hierarchies. In contrast, Text is nearly saturated across all benchmarks considered and thus provides limited discrimination. We attribute this to its primary focus on recognizing and reproducing page text, tasks for which current models already possess strong OCR capabilities.

5.4 Examples of Samples with Different PQ Scores

In the main text, we present four groups of samples with high, medium, and low PQ Scores, with more examples provided in Appendix H. As shown in Figure 3, for samples with high PQ Scores, the original screenshot A and the rendered result B of the generated code P are visually almost

identical. As the PQ Score decreases, A and B become increasingly different in visual appearance, indicating that PQ Score can effectively identify samples that are more challenging for models.

5.5 Common Failure Modes

We systematically analyze the model results and identify two major error patterns. First, the content is correct, but the layout and style are not faithfully reproduced. Models can usually recover the textual content but struggle to accurately reproduce the positions, sizes, spacing, and colors of elements. Second, the component counts and page structures are incorrect. Models often omit or duplicate images, cards, list items, table rows, and form controls, and this problem is more pronounced in less capable models. These two error patterns can be observed in the examples shown in Figures 3(b) and 3(c), respectively.

6 Discussion

6.1 Data Scale or Sample Complexity: Which Matters More for UI-to-Code?

During benchmark construction, we find that even Qwen3.5-9B can filter out 3,665,705 of the 4,771,476 samples, passing only 23.2% of them to the next stage. This indicates that, after basic training, current models can already handle most conventional UI-to-Code samples reliably. Data scale remains important for establishing foundational capabilities, but once common patterns become saturated, the marginal benefit of adding more homogeneous data decreases substantially. Compared with simply increasing data scale, incorporating more samples that are highly complex, belong to the long tail, or are prone to model failure may be a more effective direction for further improving UI-to-Code capabilities.

6.2 What Types of Samples Should Future UI-to-Code Learn From?

Based on the analysis in Section 5.5, we believe that future UI-to-Code training data should place less emphasis on simple text recovery and layout arrangement, and instead provide greater coverage of the precise reconstruction of layouts, visual styles, and page structures. Specifically, training data should include more challenging pages featuring layouts with multiple columns, dense cards, long lists, complex tables, and forms, while strengthening supervision of component positions, sizes, spacing, colors, and quantities. In addition, difficult samples with similar text but different layouts or structures can be constructed to encourage UI-to-Code models to genuinely understand the spatial relationships between page elements.

7 Conclusion and Limitations

Conclusion. We introduce PQ Score and an ABPQ benchmark construction pipeline built around it. PQ Score measures UI understanding and reconstruction through round trip consistency, enabling automatic identification of difficult samples from large screenshot collections. Using the pipeline, we construct ABPQ-FULL and ABPQ-HARD from 4,771,476 real UI screenshots and evaluate ten representative proprietary and open source models using multiple automatic metrics, LLM-as-Judge, and human evaluation. Experiments show that ABPQ is more challenging than existing benchmarks and that advanced models remain far from saturation. PQ Score also correlates strongly with multiple evaluation metrics, supporting its effectiveness for difficult sample filtering. ABPQ provides a continually updated approach to constructing UI-to-Code benchmarks with minimal human effort.

Limitations. ABPQ has two main limitations. First, the current hierarchical filtering process primarily uses Qwen models, and the final selected samples may therefore exhibit difficulty patterns specific to this model family. We conduct a preliminary evaluation across different model families in Appendix J. Although our experiments indicate that the effect of model family bias is limited, whether this observation holds at substantially larger scales requires further validation. Second, PQ Score primarily focuses on page layout and structural information and is relatively less sensitive to textual content, colors, and styles. Future work could incorporate these elements to improve the ability of PQ Score to comprehensively assess sample difficulty.

AI USE STATEMENT

In this work, we used generative AI tools for tasks requiring disclosure, including automated processing and filtering of benchmark data, execution of model-based components in the benchmark construction pipeline, and automated evaluation of model-generated webpages. Specifically, Qwen3.5-4B was used for comprehensive screenshot quality assessment; Qwen3.5-9B and Qwen3.8-27B were used for hierarchical PQ Score filtering; Qwen3.8-27B was used for garbled-text detection and UI complexity rating; and GPT-5.6-Sol, Claude-Opus-5, and Gemini-3.7-Flash were used as judges for Visual Score evaluation. We did not use generative AI tools to develop theoretical models or conceptual frameworks, formulate mathematical claims, assist with mathematical proofs, propose or refine hypotheses, design or provide feedback on the research methodology or experiments, assist with translation, or draw scientific conclusions from the experimental results. The remaining required-disclosure tasks are not applicable to this work. Additionally, we used generative AI tools to generate webpage code as experimental outputs and to perform limited language polishing to improve the clarity and readability of the paper. We have reviewed all AI-assisted work. To reduce dependence on any single AI judge, Visual Score was evaluated using three different models. The final benchmark candidates were manually reviewed by three reviewers with doctoral degrees in computer science and were retained only when at least two reviewers considered them suitable for evaluation. The reported human evaluation was conducted independently by three reviewers. All AI-assisted language edits were reviewed and approved by the authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

The data used in this work were collected from publicly accessible websites and data owned by the authors' organization. The collection, processing, and planned release of these data comply with applicable legal requirements, website terms, and licensing conditions. To protect privacy and ensure data safety, all samples underwent a rigorous multi-stage screening process, including automated NSFW and multilingual sensitive-content filtering, comprehensive assessment using Qwen3.5-4B, and final manual review by multiple reviewers. Samples containing private, sensitive, inappropriate, or otherwise unsuitable information were excluded. The released benchmark will include the selected UI screenshots and their corresponding HTML and is intended to support research and evaluation in the UI-to-Code community.

The human review process involved three reviewers. One reviewer was a non-first-author member of the author team whose role in this work was limited to the review process, while the other two were external reviewers who were compensated for their time. All reviewers participated with informed consent. To reduce the influence of individual judgments, each candidate was reviewed by multiple reviewers and was retained only when at least two reviewers considered it suitable for evaluation. The authors take responsibility for the ethical collection, processing, review, and release of the benchmark.

REPRODUCIBILITY STATEMENT

To support reproducibility, Section 3.1 and Appendix A provide the definition and implementation details of PQ Score, including DOM tree construction, node preprocessing, and tree edit distance computation. Section 3.2 and Appendix B document the complete ABPQ construction pipeline, including data processing, hierarchical filtering, selection thresholds, quality control, deduplication, and the prompts used at each stage. The experimental setup, evaluation metrics, human evaluation protocol, and complete results are reported in Sections 4–5 and Appendices D–G. Due to the file size limit for supplementary materials, source images cannot be included in supplement.zip. Instead, the archive provides the HTML files and experimental code, and corresponding screenshots can be regenerated by rendering the HTML files directly. We will publicly release the complete image set upon publication.

REFERENCES

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Yi Gui, Zhen Li, Yao Wan, Yemin Shi, Hongyu Zhang, Bohua Chen, Yi Su, Dongping Chen, Siyuan Wu, Xing Zhou, et al. Webcode2m: A real-world dataset for code generation from webpage designs. In *Proceedings of the ACM on Web Conference 2025*, pp. 1834–1845, 2025a.

Yi Gui, Yao Wan, Zhen Li, Zhongyi Zhang, Dongping Chen, Hongyu Zhang, Yi Su, Bohua Chen, Xing Zhou, Wenbin Jiang, et al. Uicopilot: Automating ui synthesis via hierarchical code generation from webpage designs. In *Proceedings of the ACM on Web Conference 2025*, pp. 1846–1855, 2025b.

Zehai He, Wenyi Hong, Zhen Yang, Ziyang Pan, Mingdao Liu, Xiaotao Gu, and Jie Tang. Vision2web: A hierarchical benchmark for visual website development with agent verification. *arXiv preprint arXiv:2603.26648*, 2026.

Xue Jiang, Yihong Dong, Lecheng Wang, Zheng Fang, Qiwei Shang, Ge Li, Zhi Jin, and Wenpin Jiao. Self-planning code generation with large language models. *ACM Transactions on Software Engineering and Methodology*, 33(7):1–30, 2024.

Yilei Jiang, Yaozhi Zheng, Yuxuan Wan, Jiaming Han, Qunzhong Wang, Michael R Lyu, and Xiangyu Yue. Screencoder: Advancing visual-to-code generation for front-end automation via modular multimodal agents. *arXiv preprint arXiv:2507.22827*, 2025.

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

Alexander H Liu, Kartik Khandelwal, Sandeep Subramanian, Victor Jouault, Abhinav Rastogi, Adrien Sadé, Alan Jeffares, Albert Jiang, Alexandre Cahill, Alexandre Gavaudan, et al. Ministral 3. *arXiv preprint arXiv:2601.08584*, 2026.

Yue Liu, Yue Yu, Yuanliang Zhang, Yu Jiang, Changjian Wang, Shanshan Li, et al. At which training stage does code data help llms reasoning? In *International Conference on Learning Representations*, volume 2024, pp. 36281–36300, 2024.

Mateusz Pawlik and Nikolaus Augsten. Tree edit distance: Robust and memory-efficient. *Information Systems*, 56:157–173, 2016.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026a. URL `https://qwen.ai/blog?id=qwen3.5`.

Qwen Team. Qwen3.8-Max: A new bar for coding and cowork, August 2026b. URL `https://qwen.ai/blog?id=qwen3.8`.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pp. 8748–8763. PmLR, 2021.

Chenglei Si, Yanzhe Zhang, Ryan Li, Zhengyuan Yang, Ruibo Liu, and Diyi Yang. 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)*, pp. 3956–3974, 2025.

Kimi Team, Tongtong Bai, Yifan Bai, Yiping Bao, Jianfeng Cai, Xinyuan Cai, Peizhou Cao, Yuxuan Cao, Ziwei Chai, Y Charles, et al. Kimi k3: Open frontier intelligence. *arXiv preprint arXiv:2607.24653*, 2026.

Yuxuan Wan, Chaozheng Wang, Yi Dong, Wenxuan Wang, Shuqing Li, Yintong Huo, and Michael Lyu. Divide-and-conquer: Generating ui code from screenshots. *Proceedings of the ACM on Software Engineering*, 2(FSE):2099–2122, 2025.

Weiyun Wang, Zhangwei Gao, Lixin Gu, Hengjun Pu, Long Cui, Xingguang Wei, Zhaoyang Liu, Linglin Jing, Shenglong Ye, Jie Shao, et al. Internvl3. 5: Advancing open-source multimodal models in versatility, reasoning, and efficiency. *arXiv preprint arXiv:2508.18265*, 2025.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better llm agents. *arXiv preprint arXiv:2402.01030*, 2024.

Zhen Yang, Wenyi Hong, Mingde Xu, Xinyue Fan, Weihan Wang, Jiale Cheng, Xiaotao Gu, and Jie Tang. Ui2codeˆ n: Ui-to-code generation as interactive visual optimization. *arXiv preprint arXiv:2511.08195*, 2025.

Sukmin Yun, Haokun Lin, Rusiru Thushara, Mohammad Q Bhat, Yongxin Wang, Zutao Jiang, Mingkai Deng, Jinhong Wang, Tianhua Tao, Junbo Li, et al. Web2code: A large-scale webpage-to-code dataset and evaluation framework for multimodal llms. *Advances in neural information processing systems*, 37:112134–112157, 2024.

Kechi Zhang, Huangzhao Zhang, Ge Li, Jia Li, Zhuo Li, and Zhi Jin. Toolcoder: Teach code generation models to use api search tools, 2023. URL https://arxiv.org/abs/2305.04032.

Xinlu Zhang, Zhiyu Zoey Chen, Xi Ye, Xianjun Yang, Lichang Chen, William Yang Wang, and Linda Ruth Petzold. Unveiling the impact of coding data instruction fine-tuning on large language models reasoning, 2024. URL https://arxiv.org/abs/2405.20535.

A PQ SCORE CALCULATION DETAILS

PQ Score measures the structural consistency between the generated HTML code P and the regenerated HTML code Q using the ordered tree edit distance between their DOM tag trees. Specifically, we first parse P and Q separately and construct ordered DOM tag trees rooted at the <body> element. If no <body> element is found, the entire document is used as the root. When constructing the DOM trees, we remove HTML comments and ignore style, script, noscript, and template nodes because they do not directly describe the visible structure of the page. For all remaining nodes, we retain only lowercase HTML tag names as node labels, excluding element attributes, CSS styles, textual content, and other nonstructural information. Therefore, the resulting ordered trees primarily represent the parent-child relationships, nesting hierarchies, and ordering of page elements.

Modern CSS may produce visually similar layouts from different DOM hierarchies. Consequently, PQ Score may assign a lower score to visually equivalent implementations that use different structures. However, all generations in our experiments use deterministic decoding, thereby excluding structural variation caused by stochastic sampling. With the model and inference configuration fixed, if the model still generates substantially different DOM structures from the original screenshot and its rendered reconstruction, this indicates that its understanding and reconstruction of the page structure are not stable across the round trip. PQ Score treats such structural instability as a loss of information.

We then use the APTED algorithm (Pawlik & Augsten, 2016) to compute the minimum tree edit distance between the two ordered DOM tag trees. The permitted edit operations include node insertion, node deletion, and node label replacement. The costs of insertion and deletion are both set to 1. For label replacement, the cost is 0 when the labels of the two nodes are identical and 1 otherwise. Let T_P and T_Q denote the DOM tag trees constructed from P and Q, respectively, and let $|T_P|$ and $|T_Q|$ denote their respective numbers of nodes. The DOM structure similarity is defined as:

$$S_{\mathrm{DOM}}(P, Q) = \max\left(0,\, 1 - \frac{\mathrm{TED}(T_P, T_Q)}{\max\left(|T_P|, |T_Q|\right)}\right). \tag{3}$$

The larger node count of the two trees is used for normalization to reduce the influence of differences in DOM tree size on the edit distance. When the two trees are identical, their tree edit distance is 0, and thus $S_{\mathrm{DOM}} = 1$. As the number of node insertions, deletions, or label replacements increases, the tree edit distance increases and S_{DOM} decreases accordingly. PQ Score is finally defined as:

$$\text{PQ Score}(P, Q) = 100 \times S_{\mathrm{DOM}}(P, Q). \tag{4}$$

PQ Score ranges from 0 to 100. When code P cannot be generated, or when P cannot be successfully rendered and subsequently used to generate Q, we do not artificially assign a PQ Score of 0. Instead, such cases are separately marked as generation failures and retained during hierarchical filtering as samples that are difficult for the model to handle.

B ABPQ PIPELINE

Web-Scale Screenshot Collection. For image hash deduplication, we remove exact duplicate images based on their image hashes. For rule based filtering, we remove screenshots in which either white or black pixels account for more than 99% of all pixels. We use HunyuanOCR as the lightweight OCR model. For filtering based on the ViT NSFW classifier and multilingual sensitive words, we follow exactly the procedure described in WebCode2M (Gui et al., 2025a). The prompt used by Qwen3.5-4B (Qwen Team, 2026a) for comprehensive page assessment is shown in Figure 4a.

Hierarchical PQ Score Filtering. We use the same prompt for Qwen3.5-9B (Qwen Team, 2026a) and Qwen3.8-27B (Qwen Team, 2026b), as shown in Figure 4b. Code generation for both models uses deterministic decoding to eliminate run-to-run variation caused by stochastic sampling. We render the generated HTML using Playwright. The rendering width is set to exactly match the width of the original screenshot, while the page height is determined by the generated code. We enable `fullPage=true` to capture the complete rendered page. One potential failure mode is that a model may collapse to the same simple, blank, or single-container page in both generation

```
You are a strict binary quality filter for a WebUI-to-Code screenshot dataset.
Your task is to decide whether the given image should be kept or dropped as a sample for webpage screenshot-to-code generation.
The image does not need to show a complete webpage. Partial webpage screenshots, cropped webpage regions, page sections, cards, forms, navigation areas, banners, product blocks, article fragments, and individual Web UI components may all be kept if they are visually valid and useful for WebUI-to-Code.
OCR text extracted from the image is provided as auxiliary evidence. It may be incomplete or inaccurate, so use it only as supporting evidence and not as the sole basis for your decision.
OCR text:
"""
{ocr_text}
"""
Evaluation criteria:
•The image must be a webpage, Web UI, web app, web component, or partial/cropped Web UI screenshot.
•The image must be sufficiently clear, readable, and visually usable for WebUI-to-Code.
•The UI must appear normally rendered, without severely missing styles, broken CSS, loading-only states, blank or black screens, or large-scale resource failures.
•The image must not contain severe element overlaps, major layout breakage, severe truncation, or an unusable visual structure.
•The image must not contain obvious high-risk private or sensitive content, such as passwords, verification codes, API keys, access tokens, cookies, bank or payment credentials, government IDs, private messages, or private account/session details.
•External images, logos, avatars, product images, and videos must be replaceable with placeholders while preserving useful Web UI layout information.
Important principles:
•Keep valid Web UI screenshots, including partial or cropped UI regions.
•Do not drop an image merely because it is simple, minimal, whitespace-heavy, or does not show a complete webpage.
•Do not require navigation bars, cards, buttons, forms, lists, grids, headers, or footers.
•External visual assets may be replaced with placeholders later.
•Drop the image if it is not a Web UI screenshot, has unacceptable visual quality, is broken or incorrectly rendered, has a severe layout issue, contains privacy-sensitive content, or is incompatible with placeholder replacement.
Decision rule:
Return "keep" only if every evaluation criterion is satisfied. Otherwise, return "drop".
Output only valid JSON. Do not include explanations, Markdown, code fences, or any additional fields.
The output must exactly follow this schema:
{"decision":"keep"}
or
{"decision":"drop"}
```

(a) Comprehensive screenshot quality assessment.

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

(b) UI-to-Code generation for PQ Score filtering.

```
You are a strict quality inspector for webpage screenshots. Your task is to determine whether the screenshot contains many missing-glyph placeholder boxes.
Important: Carefully inspect all visible text in the screenshot. Base your judgment only on visible evidence.
Screenshot: {screenshot}
Evaluation rules:
- Set "has_many_missing_glyph_boxes" to true only when the screenshot contains many repeated missing-glyph placeholder boxes, usually hollow square or tofu glyphs.
- "Many" means approximately 10 or more missing-glyph boxes across multiple lines or regions, or missing-glyph boxes affecting roughly 20% or more of the visible text.
- If there are only a few isolated missing-glyph boxes, set the result to false.
- Do not count checkboxes, input borders, card borders, icons, QR-code cells, decorative geometric squares, or the normal Chinese character "□".
- Do not judge the language of the visible text.
- First provide one short English sentence explaining the visible evidence.
- Output only a valid JSON object in the following structure:
{
"reason": "<one concise English sentence based on visible evidence>",
"answer": {
"has_many_missing_glyph_boxes": true
}
}
The value of "has_many_missing_glyph_boxes" must be either true or false.
Do not use Markdown and do not add any other fields or text.
```

(c) Garbled text detection.

```
You are a strict UI complexity evaluator for webpage screenshots. Your task is to inspect the provided screenshot and assign one overall UI complexity score.
Screenshot: {screenshot}
Important evaluation rules:
- Evaluate only the UI that is visibly present in the screenshot.
- Judge the interface holistically rather than using a mechanical checklist.
- The presence of a particular component does not automatically make the interface complex. For example, a simple table may still be visually simple.
- Do not predict whether another model could successfully reproduce the page.
- Do not judge aesthetics, usability, correctness, language, content topic, brand popularity, or hidden functionality.
- Base the score on the overall visible structure, amount of content, variety of visual elements, and level of visual detail.
- Do not calculate separate scores for individual dimensions.
- Set "is_valid_ui" to false only when there is not enough visible UI to evaluate. Otherwise, set it to true.
Use an integer complexity score from 1 to 10:
- 1–2: Very simple or sparse UI
- 3–4: Relatively simple UI
- 5–6: Moderately complex UI
- 7–8: Complex UI
- 9–10: Extremely complex or visually intricate UI
First provide one short English sentence explaining the visible evidence, using no more than 300 characters.
Output only a compact, single-line JSON object in the following format:
{"reason":"The interface has moderate visible complexity.","answer":{"is_valid_ui":true,"ui_complexity_score":5}}
The value of "is_valid_ui" must be either true or false.
The value of "ui_complexity_score" must be an integer from 1 to 10.
Do not use Markdown, indentation, line breaks, or additional fields.
```

(d) UI complexity rating.

Figure 4: Prompts used throughout the ABPQ pipeline. The four prompts are used for comprehensive screenshot assessment, hierarchical PQ Score filtering, garbled-text detection, respectively, and UI complexity rating.

passes, thereby obtaining an artificially high PQ Score. To examine this possibility, we randomly inspected 500 samples, independently sampling 250 samples from the Qwen3.5-9B filtering pool and 250 samples from the Qwen3.8-27B filtering pool. We found no cases in which both generations collapsed to the same degenerate structure and consequently received an artificially high PQ Score. In practice, the evaluated models generally produced DOM structures whose complexity reflected that of the input layouts, while differences between models primarily appeared in reconstruction fidelity rather than systematic collapse into trivial DOM structures. These results suggest that such degenerate self-consistent outputs are uncommon in the hierarchical filtering process.

Quality Control & Selection. We use Qwen3.8-27B for quality control. The prompts used for garbled text detection and complexity rating are shown in Figures 4(c) and 4(d), respectively. To prevent garbled text from affecting complexity ratings, we perform garbled text detection and complexity rating as separate steps. For deduplication based on CLIP similarity (Radford et al., 2021), we use a similarity threshold of 0.95. Manual review is conducted by three reviewers, all of whom hold PhD degrees in computer science. A sample is retained if at least two reviewers consider it suitable for evaluation.

C Benchmark Statistics

This section provides the detailed category definitions and supplementary distributions used to analyze ABPQ along three dimensions: domain coverage, page type, and user task.

C.1 Domain Coverage

As discussed in Section 3.3, we categorize the samples into 12 domains: Commerce, Education, Media, Arts, Recreation, Technology, Travel, Business, Health, Local Services, Society, and Others. Their definitions are provided in Table 7.

Table 7: Definitions of the domains.

Domain	Definition
Commerce	Retail, marketplaces, deals, ticketing, and transactions involving general consumer or specialty goods.
Education	Schools, universities, vocational learning, online courses, and academic research resources.
Media	Journalism, publishing, blogs, reviews, magazines, and other primarily editorial content.
Arts	Creative works and cultural activity across visual, performing, cinematic, and literary forms.
Recreation	Sports, fitness, digital games, gambling, and related leisure activities.
Technology	Software development, digital applications, internet infrastructure, and computing services.
Travel	Destinations, accommodation, transportation, navigation, and visitor services.
Business	Financial and professional services, manufacturing, agriculture, energy, and trade.
Health	Clinical medicine, health information, personal wellness, mental health, and caregiving.
Local Services	Property, household, automotive, dining, personal, pet, and other locally delivered services.
Society	Government, public services, nonprofit activity, religion, advocacy, and social communities.
Others	Pages whose available evidence does not support a responsible subject assignment.

C.2 Page Types

Each sample is assigned a single primary page type. We define 17 page types: Home Portal, Article, Listing Page, Entity Detail, Product Detail, Service Landing, Profile Page, Event Page, Reference Page, Discussion, Media Viewer, Interactive Tool, Form Flow, Access Gate, Legal Page, Status Page, and Other Page. Their definitions are provided in Table 8.

The page-type distributions of ABPQ-FULL and ABPQ-HARD are shown in Figures 5(a) and 5(b), respectively. ABPQ-FULL covers all 17 page types. Article is the most frequent type, with 124 samples, accounting for 24.8% of the dataset. It is followed by Home Portal, Listing Page, and Entity Detail, with 62, 55, and 53 samples, respectively. Product Detail and Media Viewer contain 33 and 28 samples, respectively, while the remaining page types are also represented.

ABPQ-HARD covers 16 of the 17 page types, with Other Page being the only absent category. Its distribution over the major page types largely follows that of ABPQ-FULL. Article remains the most frequent type, with 49 samples accounting for 24.5% of the dataset, while Home Portal, Entity Detail, Listing Page, and Product Detail contain 27, 21, 20, and 20 samples, respectively. The proportion of Product Detail increases from 6.6% in ABPQ-FULL to 10.0% in ABPQ-HARD, reflecting a localized distributional change after difficult sample filtering.

C.3 User Tasks

Each page may correspond to multiple user tasks. Therefore, the sum of the sample counts across task categories may exceed the total number of samples in the dataset. We define 19 user tasks, whose definitions are provided in Table 9.

Table 8: Definitions of the page types.

Page Type	Definition
Home Portal	The main entry page of a website or product.
Article	A news story, blog post, tutorial, review, or other continuous editorial page.
Listing Page	A directory, category, archive, or search-results page containing multiple entries.
Entity Detail	Structured information about one place, person, course, work, or organization record.
Product Detail	A page for one purchasable product or explicit product offer.
Service Landing	A page promoting a service, solution, program, or conversion proposition.
Profile Page	A profile of an organization, team, brand, or individual.
Event Page	Details for an event, performance, exhibition, class session, or competition.
Reference Page	Documentation, manuals, papers, reports, encyclopedia entries, or other reference material.
Discussion	A forum topic, question, issue, or community post with replies.
Media Viewer	A page centered on viewing or playing visual or audio media.
Interactive Tool	A calculator, editor, map, dashboard, configurator, or other interactive utility.
Form Flow	A booking, registration, application, checkout, contact, or submission flow.
Access Gate	A login, registration, verification, permission, or paid-access gate.
Legal Page	Privacy, cookie, terms, disclaimer, or other legal text.
Status Page	An error, unavailable resource, system status, or otherwise indeterminate page.
Other Page	A page with no reliably dominant structural type.

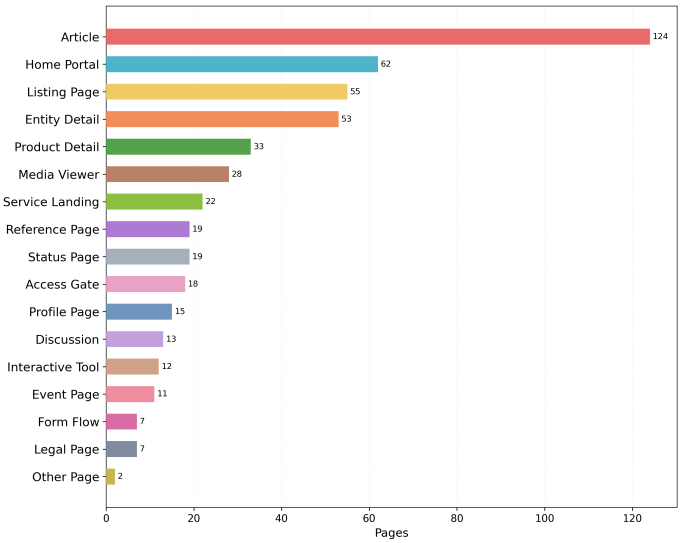
The user-task distributions of ABPQ-FULL and ABPQ-HARD are shown in Figures 6(a) and 6(b), respectively. In ABPQ-FULL, the most frequent tasks are Site Navigation, Page Controls, and Content Reading, covering 299, 279, and 256 samples, or 59.8%, 55.8%, and 51.2% of the dataset, respectively. Search and Filter and Item Inspection cover 205 and 175 samples, respectively. In addition, Contact Support, Subscription, Share and Export, and Media Viewing each cover more than 120 samples, indicating that ABPQ-FULL contains diverse scenarios involving navigation, reading, search, page interaction, and content operations.

ABPQ-HARD largely preserves the user-task distribution of ABPQ-FULL. Site Navigation and Page Controls each cover 117 samples, accounting for 58.5% of the dataset. Content Reading covers 95 samples, accounting for 47.5%, while Item Inspection and Search and Filter cover 81 and 70 samples, accounting for 40.5% and 35.0%, respectively. Compared with ABPQ-FULL, ABPQ-HARD contains higher proportions of Item Inspection, Map Navigation, Account Access, and Interactive Use, while the proportions of Content Reading, Search and Filter, and Community are slightly lower. Overall, despite its smaller size, ABPQ-HARD continues to cover a broad range of user tasks, from content browsing and search to shopping, applications, community interaction, and tool use.

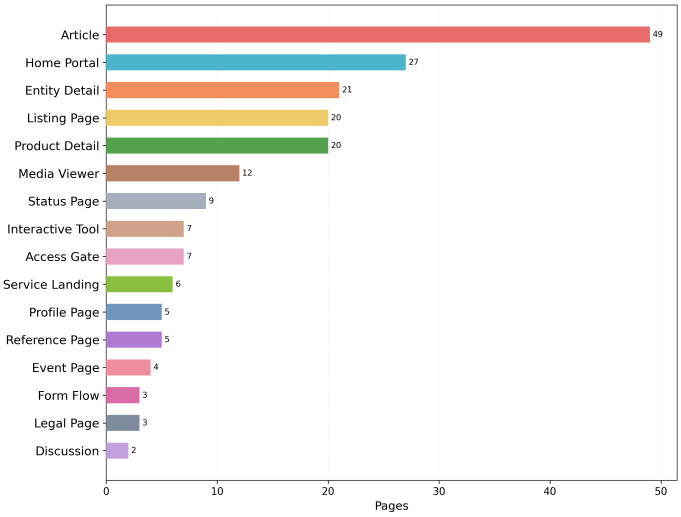
D Evaluation Metric Details

This section provides further details on the automatic and human evaluation methods used in our experiments. The calculation of PQ Score is presented in Appendix A, followed by descriptions of the other evaluation metrics. Note that our code generation process uses deterministic decoding. For samples where code generation fails or the generated code cannot be successfully rendered, the corresponding evaluation scores are set to 0.

For Block, Text, Position, and Color, we directly use the evaluation implementation provided by Design2Code (Si et al., 2025) without modifying their calculation methods. Given a reference screenshot I_R and a generated page screenshot I_G, the text detection module extracts the sets of visual elements $R = \{r_1, r_2, \ldots, r_m\}$ and $G = \{g_1, g_2, \ldots, g_n\}$, respectively. Each region contains textual content and bounding box coordinates. The optimal matching M between R and G is then obtained using the Jonker–Volgenant algorithm based on text similarity.



(a) ABPQ-FULL



(b) ABPQ-HARD

Figure 5: Page type distributions of ABPQ-FULL and ABPQ-HARD. ABPQ-FULL covers all 17 page types, while ABPQ-HARD retains 16 and largely preserves the overall distribution, with Article remaining the most frequent type.

Block. Block measures whether all visual elements in the reference page are reproduced and whether the generated page contains additional elements that do not exist in the reference page. It calculates the ratio of the total area of all matched regions to the total area of all regions, including unmatched regions in both the reference and generated pages. An unmatched region indicates either a missing original element or an additionally generated element and therefore lowers the score. The larger the unmatched region, the lower the score.

Text. For each pair of matched regions, text similarity is defined as twice the number of overlapping characters between the two text strings divided by their total number of characters, corresponding to the character-level Sørensen–Dice similarity. The final score is the average text similarity across all matched region pairs.

Table 9: Definitions of the user tasks.

User Task	Definition
Site Navigation	Browse home pages, sections, categories, menus, or recommendations.
Content Reading	Read articles, news, tutorials, documentation, reports, or other text.
Search and Filter	Search, filter, sort, or narrow a collection of site resources.
Item Inspection	Inspect details for a product, service, place, person, course, or other entity.
Media Viewing	View or play images, galleries, comics, video, audio, or streams.
Map Navigation	Use maps, locations, routes, distance, or transport information.
Save and Compare	Save, follow, bookmark, shortlist, or compare candidate items.
Account Access	Sign in, register, recover credentials, or complete identity verification.
Page Controls	Manage privacy, locale, accessibility, consent, or obstructing overlays.
Contact Support	Use contact, quotation, chat, customer service, or technical support channels.
Subscription	Subscribe to updates, newsletters, alerts, creators, or topics.
Share and Export	Share, print, download, copy, or export content and results.
Community	Comment, reply, post, message, chat, or otherwise join a community interaction.
Interactive Use	Operate a calculator, editor, configurator, visualization, or similar tool.
Shopping	Manage a cart, select options, purchase, subscribe, or complete checkout.
Booking Application	Book, register, enroll, reserve, inquire, or submit an application.
Financial Support	Donate, sponsor, contribute, or provide noncommercial financial support.
Gameplay	Play a browser game, puzzle, simulator, challenge, or prize draw.
Reviews	Read or submit ratings, reviews, or feedback.

Position. The positions of regions substantially affect the overall page layout. For each pair of matched regions, Position calculates their positional similarity using the normalized coordinates of their center points:

$$\mathrm{sim}_{\mathrm{pos}}(r_p, g_q) = 1 - \max\left(|x_q - x_p|, |y_q - y_p|\right), \tag{5}$$

where (x_p, y_p) and (x_q, y_q) are the center coordinates of the two regions normalized to the range from 0 to 1. The final score is the average positional similarity across all matched region pairs.

Color. Color uses the CIEDE2000 color difference formula to measure the perceptual difference between the text colors in a generated region and its corresponding reference region. This formula accounts for the complexity of human color perception. The final score is the average color similarity across all matched region pairs.

Visual Score. We adopt the evaluation prompt from Vision2Web (He et al., 2026), in which a judge model compares the reference screenshot with the generated page screenshot and evaluates the overall visual fidelity of the page. We make only one addition to the original prompt: blue placeholders used to replace external images should not be penalized for having image content different from the original images, but their positions and sizes should still be compared. The complete evaluation prompt is shown in Figure 7.

Human Evaluation. For human evaluation, reviewers are shown both the reference screenshot and the generated page screenshot and are provided with the following instruction: "Please carefully compare the two webpage screenshots, considering content completeness, layout accuracy, and visual style consistency. Blue placeholders represent external images and should be treated as equivalent to the corresponding images in the reference screenshot. Do not deduct points for their image content, but still evaluate their positions and sizes." The detailed rating criteria on a scale from 0 to 5 are provided in Table 10.

Table 10: Definitions of the human evaluation scores.

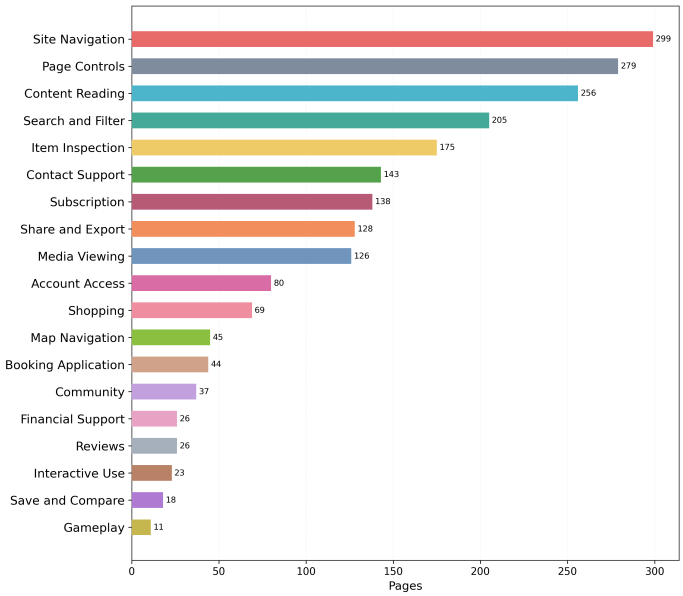
Score	Overall Replacement Quality
0	Rendering fails or no valid page is produced.
1	The result cannot serve as a replacement. Core content is substantially missing or incorrect, and the page structure, layout, or visual style differs markedly from the reference page.
2	The result has low replacement quality. Some content is retained, but important regions are missing, or there are severe deviations in layout, dimensions, hierarchy, or visual style. Substantial revision is required.
3	The result can partially serve as a replacement. The core content and overall structure are recognizable, but obvious omissions, misalignment, size deviations, or visual-style differences remain. Some revision is required.
4	The result can generally serve as a replacement. The content is relatively complete, and the layout and visual style closely resemble those of the reference page, with only a few local omissions, positional deviations, or styling differences.
5	The result can serve almost directly as a replacement. Visible text, images, and components are complete and accurate, while the structure, position, dimensions, spacing, fonts, colors, and decorative details closely match those of the reference page. Only negligible rendering differences remain.

Table 11: Automatic evaluation results on ABPQ-FULL. Scores are generally higher than on ABPQ-HARD, while model rankings remain highly consistent across the two sets. Higher is better, and bold indicates the best result within each model group.

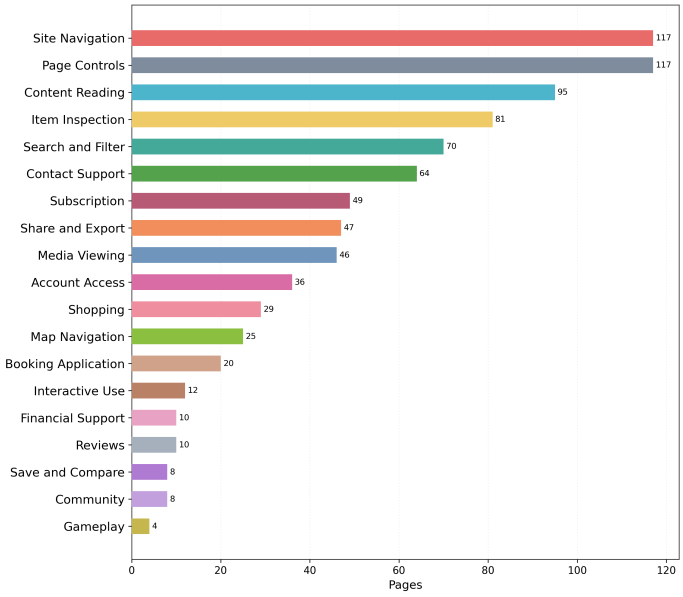
Model	VS-GPT	VS-Opus	VS-Gemini	Text	Block	Position	Color	PQ Score
Closed-source								
GPT-5.6-Sol High	**52.2**	**66.1**	**82.8**	97.5	72.2	57.2	**56.9**	53.6
Claude-Opus-5	39.7	61.7	78.9	**97.6**	**81.7**	57.0	56.1	**57.3**
Claude-Opus-4.7	32.4	49.8	63.6	97.3	81.1	57.2	54.9	54.4
Gemini-3.7-Flash	31.6	52.2	74.0	95.5	75.0	55.5	56.2	51.5
Doubao-Seed-2.0-Pro	32.2	48.7	67.4	97.2	80.4	**57.4**	56.7	50.4
Open-source								
Ministral-3-14B-2512	25.0	31.4	40.0	89.4	65.5	52.8	49.8	40.4
Qwen3.8-27B	28.1	42.5	57.2	85.2	65.7	49.5	46.3	41.8
InternVL3.5-38B	20.5	22.2	27.1	93.8	76.0	**58.2**	**57.0**	45.4
Qwen3.5-397B-A17B	32.6	50.0	68.9	**97.5**	**79.3**	56.3	54.4	**51.8**
Kimi-K3	**39.8**	**58.9**	**79.2**	92.8	76.3	53.8	51.4	51.1

Table 12: Pearson correlations between PQ Score and other automatic metrics on ABPQ-HARD. PQ Score is strongly correlated with Text and Block and moderately with most others.

Metric	Pearson's r	Correlation Strength
VS-GPT	0.16	Weak
VS-Opus	0.41	Moderate
VS-Gemini	0.43	Moderate
Text	0.72	Strong
Block	0.81	Very Strong
Position	0.39	Moderate
Color	0.42	Moderate



(a) ABPQ-FULL



(b) ABPQ-HARD

Figure 6: User task distributions of ABPQ-FULL and ABPQ-HARD. Each page may involve multiple tasks, so the category counts can exceed the dataset size. Both sets cover all 19 user tasks, while ABPQ-HARD largely preserves the broad task distribution of ABPQ-FULL.

E Automatic Evaluation Results on ABPQ-FULL

In Section 4.2.1, we report the automatic evaluation results of the models on ABPQ-HARD. This section provides the complete results of all ten models on ABPQ-FULL, as shown in Table 11. ABPQ-FULL and ABPQ-HARD support the same overall conclusions: current advanced models still have substantial room for improvement across multiple metrics, and different models exhibit strengths on different metrics. Across the 80 model–metric combinations, scores on ABPQ-HARD are lower than those on ABPQ-FULL in 43 cases, indicating that ABPQ-HARD is generally more

```
You are a senior QA automation engineer. Your task is to compare a prototype image with an actual page
screenshot and score the appearance of the page.
Important: Carefully observe all differences between the prototype and the actual page. Do not overlook any
discrepancies, however small. Score strictly according to the rules below.
1. Prototype Image : {prototype_page}
2. Actual Page Image : {actual_page}
Note: You need to automatically segment the page into meaningful UI components based on visual and functional
layout.
- Divide the page into logical functional blocks, not too granular or too coarse.
- Example functional blocks:
- Navigation bar / header
- Hero section / main banner
- Product listing / catalog area
- Checkout / payment section
- Footer
- Any other visually distinct functional sections
- For each block, treat it as a single component for scoring purposes.
- In the Actual Page Image, a flat solid blue rectangle may be the intentional rick.jpg placeholder for an image or
other visual-media region in the prototype.
- When such a blue placeholder corresponds to an image/media region in the Prototype Image, do not deduct points
because the placeholder's color, subject, texture, or internal image content differs from the prototype. Treat the
intended image content as matched for scoring purposes.
- Still evaluate and score the placeholder's geometry and layout normally: its presence, count, position, width, height,
aspect ratio, alignment, spacing, cropping, and relationship to surrounding elements must match the corresponding
prototype region.
- A missing placeholder, an extra placeholder, a placeholder in the wrong region, or incorrect placeholder geometry
must still reduce the score.
- Do not apply this exemption to genuine blue backgrounds, buttons, decorations, or other non-image UI elements.
Each component/block gets an independent score:
- Output only a JSON object in the following structure:
[
{{
"name": "<component name>",
"score": <0 | 0.25 | 0.5 | 0.75 | 1>,
"reason": "<brief explanation of why this score was given>"
}}
]
```

Figure 7: Visual Score evaluation prompt adapted from Vision2Web.

challenging. Meanwhile, the relative model rankings are highly consistent across the two sets, with only minor changes in order among a few models with similar scores. This consistency demonstrates that ABPQ-HARD can provide representative evaluation results at a smaller scale.

Furthermore, we observe that InternVL3.5-38B achieves a higher PQ Score than GPT-5.6-Sol High on ABPQ-HARD (47.0 vs. 44.6) in Table 2, despite receiving substantially lower Visual Scores. On ABPQ-FULL, GPT-5.6-Sol High achieves a higher PQ Score than InternVL3.5-38B (53.6 vs. 45.4), but the gap of 8.2 points is substantially smaller than the gaps of 31.7–55.7 points across the three Visual Score judges. This difference reflects that PQ Score places greater emphasis on structural information, whereas Visual Score focuses on overall visual fidelity. PQ Score therefore provides an effective complementary perspective, and combining it with other evaluation metrics enables a more comprehensive assessment of UI-to-Code capabilities.

F Correlation Between PQ Score and Other Metrics on ABPQ-HARD

As shown in Section 5.1, PQ Score exhibits strong positive correlations with all other metrics on ABPQ-FULL. This section additionally presents the results on ABPQ-HARD, as shown in Table 12. Compared with ABPQ-FULL, the correlations with some metrics decrease. One possible reason is that the primary errors in difficult samples may lie in different dimensions, such as overall visual quality, textual content, or layout structure. Consequently, different metrics may assess the same generated result more divergently, weakening the linear relationships among their scores. Nevertheless, PQ Score remains strongly correlated with Text and Block and moderately correlated with most other metrics. ABPQ-FULL reflects the overall associations among different metrics, whereas ABPQ-HARD further reveals their complementarity on difficult samples.

G Additional Comparison Results with Representative Benchmarks

Section 5.3 presents results on Design2Code, Design2Code-HARD, Vision2Web, ABPQ-FULL, and ABPQ-HARD for three representative models: GPT-5.6-Sol, Claude-Opus-5, and Kimi-

K3 (Team et al., 2026). This section reports results for the remaining seven models, as shown in Table 13. The additional results support the same overall conclusions as the main text. ABPQ-FULL and ABPQ-HARD yield the lowest scores for most combinations of models and metrics. In particular, they are more challenging for all models on Position and Color, while PQ Score exhibits the same overall trend. For most models, all three VS judges also identify ABPQ as the most challenging benchmark. The few exceptions mainly occur for relatively less capable open source models, whose scores are already low on the other benchmarks, making the differences in difficulty among benchmarks less pronounced. In addition, Text remains generally high across all benchmarks, further demonstrating its limited discriminative ability. Overall, these additional results show that the greater difficulty of ABPQ extends across models of different scales and capability levels, rather than being limited to the three representative models selected in the main text.

H More Examples of Samples with Different PQ Scores

Section 5.4 presents several samples with different PQ Scores. This section provides additional examples, as shown in Figures 8–12. For each example, the original screenshot A is shown on the left, and the rendered result B of the generated code P is shown on the right. Overall, A and B are visually similar for samples with high PQ Scores, while their visual differences gradually increase as the PQ Score decreases. These additional examples further demonstrate that PQ Score can effectively identify samples that are more challenging for models.

I Scalability of the ABPQ Pipeline

The scalability of ABPQ primarily stems from the fact that its challenging-sample filtering process does not require reference HTML. Depending on data availability, the pipeline can be applied in two different scenarios.

When paired UI–HTML data are available, as is often the case within companies that have accumulated large collections of webpage screenshots and their corresponding code, ABPQ can automatically mine challenging samples using only the screenshots while retaining the corresponding HTML for subsequent evaluation. A benchmark constructed in this manner supports not only PQ Score and Visual Score, but also metrics that rely on reference code, such as Block, Text, Position, and Color, thereby enabling a more comprehensive multidimensional evaluation.

When only UI screenshots are available without corresponding HTML, researchers can still run the models and identify challenging samples by following the automated ABPQ pipeline. Although metrics that rely on reference code cannot be computed in this setting, PQ Score does not require reference code, while Visual Score only requires comparisons between the original screenshots and the rendered results of the generated pages. These two metrics can evaluate models from the perspectives of structural information preservation and overall visual fidelity, respectively. Human evaluation can also be incorporated when necessary.

Overall, ABPQ does not require paired HTML in every application scenario. When reference code is available, the pipeline supports a more comprehensive evaluation using multiple metrics. When reference code is unavailable, PQ Score and Visual Score can still support automated challenging-sample filtering and model evaluation. By continuously collecting new UI screenshots and rerunning the filtering pipeline, practitioners can construct and update challenging evaluation sets as model capabilities evolve.

J Evaluation of Model Family Bias in ABPQ

To assess the robustness of the easy sample removal step, we sampled 1,000 instances assigned a PQ Score above 40 by Qwen3.8-27B and independently evaluated them using InternVL3.5-38B. Among these samples, 928 also received a PQ Score above 40 from InternVL, corresponding to a confirmation rate of 92.8%. This result indicates that the samples discarded as easy by Qwen are largely recognized as easy by a model from a different family. For samples with a PQ Score below 40, subsequent filtering primarily relies on page component complexity analysis and human evaluation, and is therefore minimally affected by model preferences.

Table 13: Cross-benchmark comparison of seven additional models. ABPQ yields the lowest scores for most model–metric combinations, confirming its greater difficulty across models of different scales and capabilities. Lower scores indicate greater difficulty, and bold marks the lowest value in each column for each model. "–" indicates that the metric is unavailable because Vision2Web does not provide the reference code required for its computation.

(a) Claude-Opus-4.7

Benchmark	VS-GPT	VS-Opus	VS-Gemini	Text	Block	Position	Color	PQ Score
Design2Code	36.5	56.9	72.5	98.8	95.0	93.4	87.0	72.7
Design2Code-HARD	35.9	59.8	78.7	**97.0**	83.1	92.9	87.8	70.0
Vision2Web	34.9	54.5	77.0	–	–	–	–	66.3
ABPQ-HARD	**32.2**	**48.9**	**62.1**	97.7	**80.1**	**54.7**	**50.5**	54.6
ABPQ-FULL	32.4	49.8	63.6	97.3	81.1	57.2	54.9	**54.4**

(b) Gemini-3.7-Flash

Benchmark	VS-GPT	VS-Opus	VS-Gemini	Text	Block	Position	Color	PQ Score
Design2Code	35.7	59.3	82.3	97.4	91.1	92.6	92.5	73.3
Design2Code-HARD	37.9	60.2	85.7	97.0	77.9	90.0	92.5	73.0
Vision2Web	35.7	58.3	86.4	–	–	–	–	63.3
ABPQ-HARD	31.8	53.0	76.0	96.5	76.9	**54.3**	**52.9**	53.2
ABPQ-FULL	**31.6**	**52.2**	**74.0**	**95.5**	**75.0**	55.5	56.2	**51.5**

(c) Doubao-Seed-2.0-Pro

Benchmark	VS-GPT	VS-Opus	VS-Gemini	Text	Block	Position	Color	PQ Score
Design2Code	37.5	60.8	83.3	98.6	93.4	93.7	93.4	71.8
Design2Code-HARD	36.7	57.9	78.3	**96.4**	**74.3**	90.6	90.0	69.6
Vision2Web	34.6	53.4	76.3	–	–	–	–	62.1
ABPQ-HARD	**32.0**	**47.4**	**66.0**	97.1	81.3	**55.3**	**51.0**	**49.0**
ABPQ-FULL	32.2	48.7	67.4	97.2	80.4	57.4	56.7	50.4

(d) Ministral-3-14B-2512

Benchmark	VS-GPT	VS-Opus	VS-Gemini	Text	Block	Position	Color	PQ Score
Design2Code	29.9	42.6	53.2	97.7	90.2	86.3	80.0	62.5
Design2Code-HARD	**23.3**	**30.3**	**38.4**	**82.4**	**50.1**	72.4	71.8	45.0
Vision2Web	29.1	36.4	48.2	–	–	–	–	53.5
ABPQ-HARD	25.6	33.2	41.7	91.7	69.9	**52.7**	**48.8**	43.2
ABPQ-FULL	25.0	31.4	40.0	89.4	65.5	52.8	49.8	**40.4**

(e) Qwen3.8-27B

Benchmark	VS-GPT	VS-Opus	VS-Gemini	Text	Block	Position	Color	PQ Score
Design2Code	35.8	58.4	78.5	98.4	91.7	92.4	91.0	75.6
Design2Code-HARD	37.0	58.1	81.5	96.4	75.6	86.6	88.3	73.5
Vision2Web	35.5	53.4	75.3	–	–	–	–	70.6
ABPQ-HARD	29.4	44.0	59.4	90.0	70.7	51.7	**45.6**	42.4
ABPQ-FULL	**28.1**	**42.5**	**57.2**	**85.2**	**65.7**	**49.5**	46.3	**41.8**

(f) InternVL3.5-38B

Benchmark	VS-GPT	VS-Opus	VS-Gemini	Text	Block	Position	Color	PQ Score
Design2Code	26.3	34.7	41.8	97.6	91.3	81.8	78.3	63.7
Design2Code-HARD	24.2	29.8	37.5	**91.8**	**62.6**	79.9	76.5	57.8
Vision2Web	**20.4**	**20.4**	**26.7**	–	–	–	–	**42.4**
ABPQ-HARD	20.7	22.7	27.7	95.2	80.5	**57.1**	**54.9**	47.0
ABPQ-FULL	20.5	22.2	27.1	93.8	76.0	58.2	57.0	45.4

(g) Qwen3.5-397B-A17B

Benchmark	VS-GPT	VS-Opus	VS-Gemini	Text	Block	Position	Color	PQ Score
Design2Code	37.7	59.7	79.3	98.7	92.4	92.6	95.0	72.1
Design2Code-HARD	37.2	61.0	82.0	98.0	**78.1**	87.8	90.6	71.7
Vision2Web	36.3	54.7	78.3	–	–	–	–	67.5
ABPQ-HARD	32.8	**49.1**	69.0	98.1	80.7	**55.2**	**49.9**	52.7
ABPQ-FULL	**32.6**	50.0	**68.9**	**97.5**	79.3	56.3	54.4	**51.8**

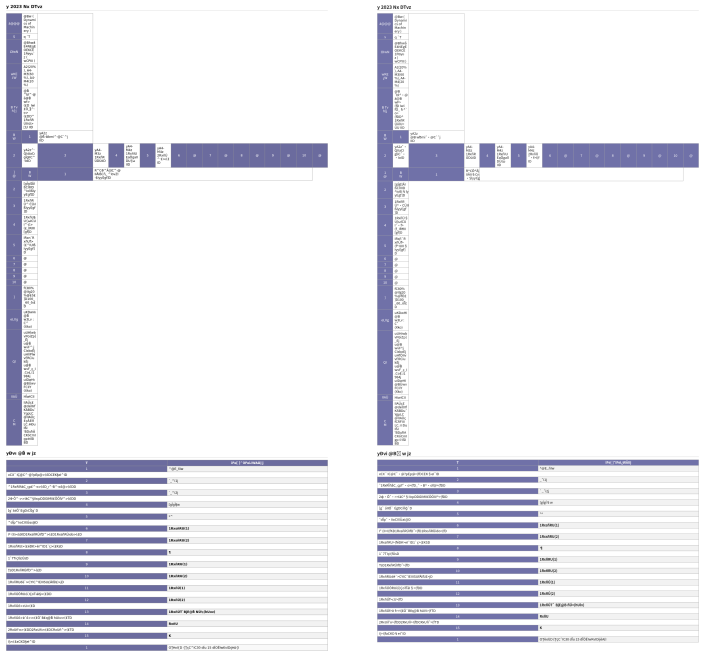


Figure 8: Sample with a PQ Score of 99.25. The original screenshot A is shown on the left and the rendered result B of generated code P on the right.

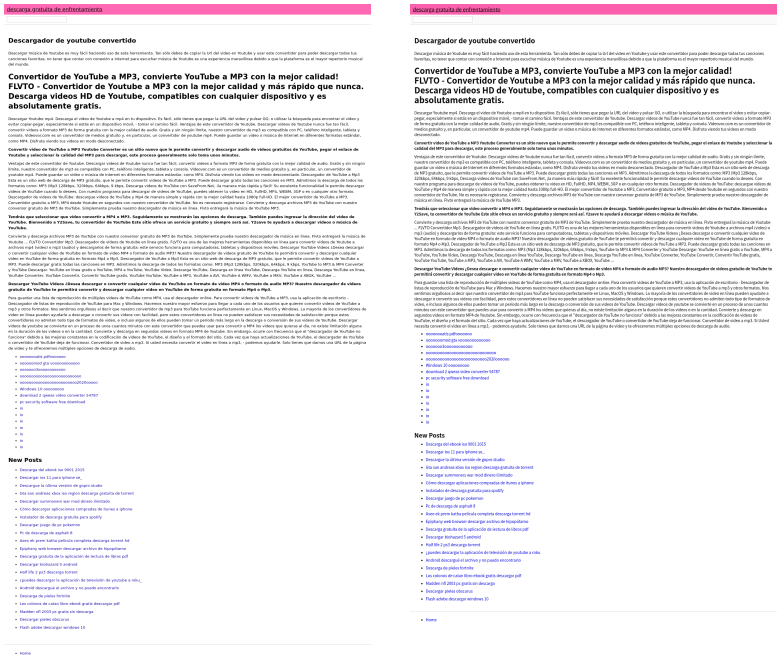


Figure 9: Sample with a PQ Score of 98.99. The reference image is shown on the left and the generated result is shown on the right.

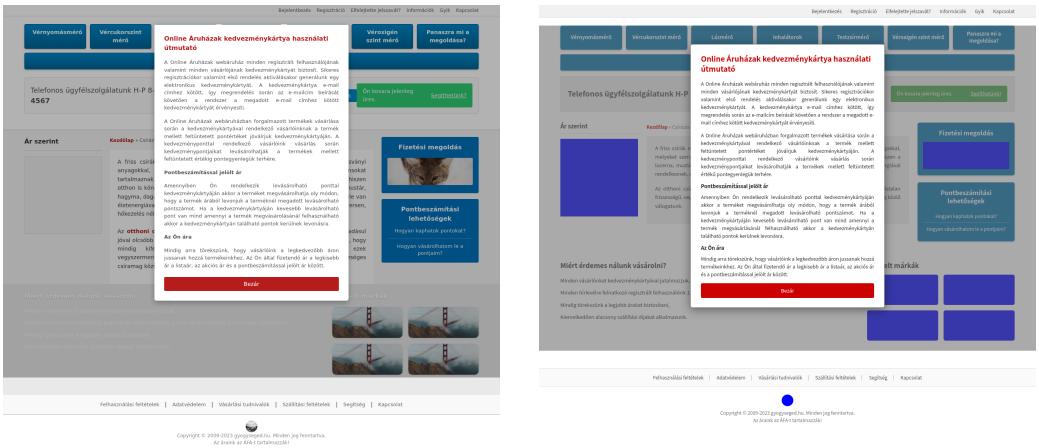


Figure 10: Sample with a PQ Score of 50.00. The reference image is shown on the left and the generated result is shown on the right.

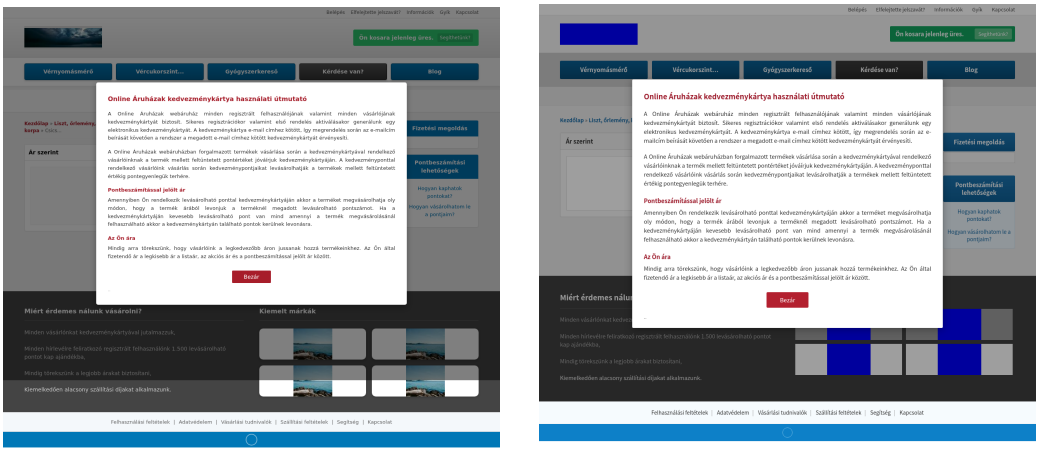


Figure 11: Sample with a PQ Score of 50.00. The reference image is shown on the left and the generated result is shown on the right.

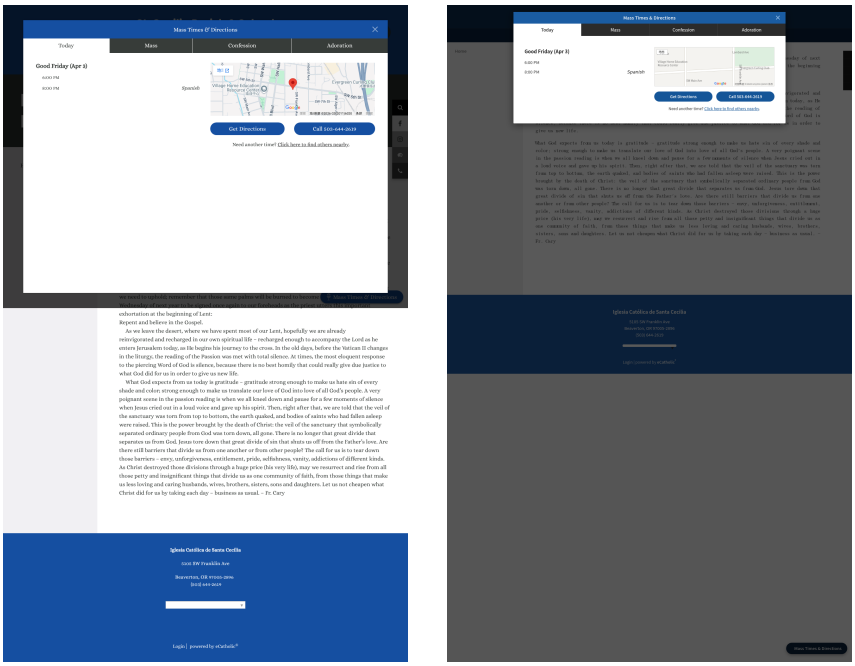


Figure 12: Sample with a PQ Score of 7.14. The reference image is shown on the left and the generated result is shown on the right.