A11yn: Aligning LLMs for Accessible Web UI Code Generation

Anonymous ACL submission

Abstract

Large language models can generate functional and visually appealing web UIs from natural language instructions. However, these models frequently produce interfaces that fail to meet accessibility requirements, excluding users with diverse needs and contexts. We address this by introducing A11yn, a policy alignment framework for accessibility-aware UI code generation. A11yn utilizes a severity-aware WCAG reward as a training signal, prioritizing the correction of high-impact accessibility failures. To support model training and evaluation, we contribute two resources: UIReq-6.8K, a dataset of 6,800 diverse UI generation instructions, and RealUIReq-300, a benchmark of 300 real-world tasks grounded in public web pages. Experimental results show that A11yn significantly improves accessibility compliance over strong baselines. Critically, these gains are achieved while maintaining the semantic fidelity and visual quality of the generated UIs, demonstrating that code-generating LLMs can be successfully aligned with objective accessibility requirements.

1 Introduction

Large language models (LLMs) with programming capabilities can generate web interfaces directly from natural language instructions, ranging from simple static pages to complex interactive components (Zhou et al., 2025). In this setting, recent work guides such models toward specific notions of interface quality, emphasizing adherence to contemporary design conventions, preservation of semantic structure, and visual coherence (Xiao et al., 2025; Lu et al., 2025).

However, *accessibility* is largely absent from the notions of interface quality considered in LLM-based web UI generation. Accessibility determines whether web interfaces can be perceived and operated by users with diverse abilities; for example, whether content is interpretable by screen readers

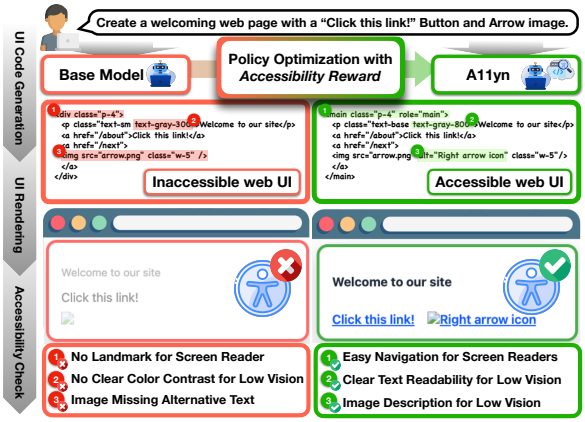


Figure 1: **A11yn improves accessibility in UI-generative LLMs.** Compared to base models, A11yn produces web UIs with enhanced accessibility features, including improved screen reader compatibility, smoother navigation, and clearer image descriptions.

for blind users or navigable via keyboard-only interaction for users with motor impairments. When these requirements are unmet, otherwise functional interfaces can become unusable. Importantly, accessibility is not a subjective criterion. The W3C formalizes accessibility requirements through the Web Content Accessibility Guidelines (WCAG), which define concrete and testable standards.

Despite this, large-scale audits show that over 90% of public web pages contain detectable WCAG violations (Mowar et al., 2025), indicating that accessibility failures are pervasive. LLMs, trained on web corpora with such flaws, often reproduce them in generated UIs. Prior studies (Suh et al., 2025; Mowar et al., 2025; Aljedaani et al., 2024; Guriţă and Vatavu, 2025) confirm that LLMs omit key accessibility elements, such as alternative text, semantic landmarks, and properly labeled form controls. This raises a core research question: *Can we align LLMs to natively generate web UIs that are more accessible?*

In this work, we introduce **A11yn** (pronounced

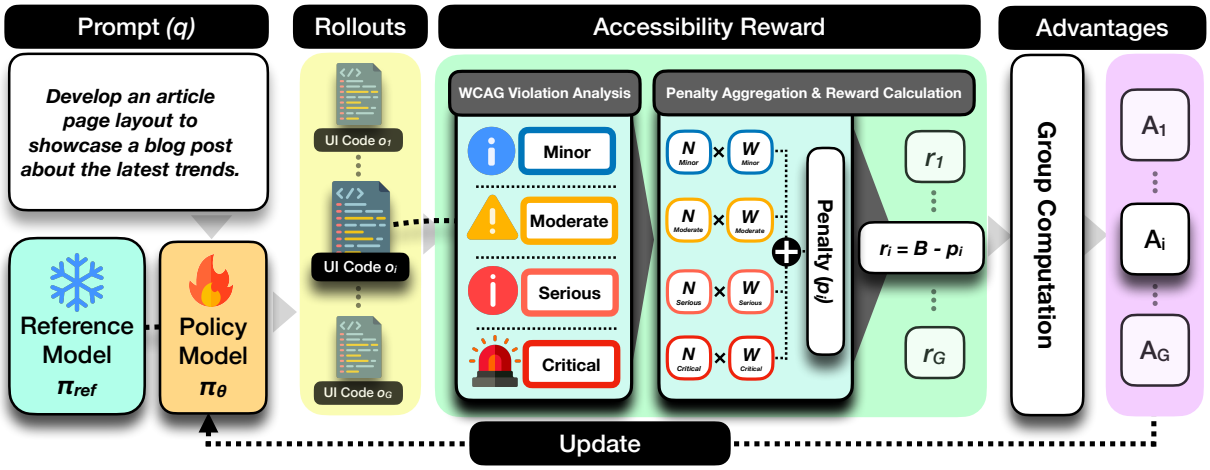


Figure 2: **A11yn aligns UI-generative LLMs using a severity-aware accessibility reward.** For an instruction q, the policy LLM π_θ generates candidate UI codes $\{o_1, \ldots, o_G\}$, each evaluated by a WCAG-based accessibility reward $\{r_1, \ldots, r_G\}$ that explicitly weights violations by severity. Rewards are normalized within the candidate set to compute advantages, which guide policy updates of π_θ via GRPO toward outputs that prioritize high-impact accessibility improvements.

align), a framework for aligning code-generating LLMs toward accessibility-aware web UI generation. A11yn operationalizes accessibility as a learnable objective by defining a **severity-aware reward** derived from WCAG violations detected using Axe Core (Deque Systems, 2015), a widely adopted accessibility auditing tool. Detected violations are mapped to severity-weighted penalties and transformed into a bounded reward signal, which is used to directly optimize the code LLM policy via Group-Relative Policy Optimization (GRPO) (Shao et al., 2024).

To enable reinforcement learning without relying on expensive supervised accessibility annotations, we construct **UIReq-6.8K**, an instruction-only dataset of 6,800 natural language UI generation requests spanning diverse domains and component requirements. For evaluation, we curate **RealUIReq-300**, a benchmark of 300 real-world web UI generation tasks, each annotated with structured metadata including page purpose, application domain, page type, and required UI components.

Experimental results show that A11yn substantially reduces accessibility violations, achieving a 60% decrease in Inaccessibility Rate compared to the base model, while preserving both visual appearance and semantic fidelity. These findings demonstrate that accessibility can be integrated as a first-class, learnable behavior in LLM-based UI generation, advancing toward more inclusive and standards-compliant web interfaces.

2 Related Work

LLM-based UI Code Generation. Prior research has applied specialized models to automate the translation of designs or descriptions into code. Early work like ReDraw (Moran et al., 2018) used a learned model to assemble mobile UI code from image mock-ups. With the advent of large language models (LLMs), generating UI code directly from high-level natural language descriptions has become feasible. For instance, UICoder (Wu et al., 2024) iteratively fine-tunes a pre-trained model with SFT on a self-generated SwiftUI training dataset, which is filtered in scale with automatic compiler feedback and a CLIP-based model. On the web UI generation side, WebGen-Bench (Lu et al., 2025) provides a benchmark designed to evaluate LLM-based agents in generating fully functional, multi-page web applications, featuring diverse application generation instructions and automated web navigation tests to assess functionality.

Post-hoc accessibility guidance. Real-world web data often contains accessibility violations, leading LLMs trained on such data to reproduce accessibility flaws in generated UI code (Martins and Duarte, 2024; Guriţă and Vatavu, 2025; Ahmed et al., 2025; Aljedaani et al., 2024). While LLMs can sometimes surpass human-written code in accessibility, they still struggle in compliance (Suh et al., 2025). Novice developers using AI assistants also frequently omit key practices, underscoring current limitations (Mowar et al., 2025). To address

these issues, practical tools like CodeA11y (Mowar et al., 2025), a VS Code plugin (Calì et al., 2025), and real-time in-DOM correction (Huang et al., 2024) provide LLM-based accessibility support. Feeda11y (Suh et al., 2025) further improves accessibility by applying feedback loops to iteratively prompt LLMs for better compliance. Yet such methods remain costly because the inference overhead often exceeds the training cost. This motivates training models that natively generate accessible code by design.

3 Methodology

A11yn incorporates a novel accessibility reward for web UI code-generating policy optimization. Below, we outline (1) the preliminary of our approach, (2) the reward design, and (3) the training pipeline.

3.1 Preliminary

We adopt Group Relative Policy Optimization (GRPO) (Shao et al., 2024), a policy gradient method that optimizes policies by comparing multiple sampled completions for the same prompt. Given a UI request q, the policy samples a group of candidates $\{o_1, \ldots, o_G\}$, each assigned a scalar reward r_i. Rewards are normalized within the group to compute relative advantages, emphasizing comparative improvements rather than absolute scores. GRPO updates the policy using a clipped surrogate objective with KL regularization against a frozen reference model:

$$J_{\text{GRPO}}(\theta) = \mathbb{E}_{q,\{o_i\}\sim\pi_{\theta_{\text{old}}}}\Bigg[\frac{1}{G}\sum_{i=1}^{G}\frac{1}{|o_i|}\sum_{t=1}^{|o_i|}L_t^{(i)}(\theta)\Bigg] - \beta\, D_{\text{KL}}(\pi_\theta\|\pi_{\text{ref}}). \tag{1}$$

Unlike PPO (Schulman et al., 2017)'s critic-based absolute value optimization, GRPO's relative reward comparisons better accommodate sparse accessibility reward in guiding policy toward accessible outcomes.

3.2 Accessibility Reward

To guide the policy towards generating accessible web UI code, we design a reward function with Web Content Accessibility Guidelines (WCAG) auditing tool. After current policy model π_θ generating each web UI code output, we run Axe-core (Deque Systems, 2015), a widely used open-source accessibility engine, to detect violations of the WCAG. For each response completion, Axe-core returns a list of affected DOM nodes, where each violation is classified by severity $v \in \{\text{Minor, Moderate, Serious, Critical}\}$. For a UI output o_i, we let $V(o_i)$ denote the set of severity levels detected in that output. The number of nodes associated with each violation level is counted and denoted as N_v. The total penalty p_i for a UI output o_i is computed by aggregating the affected DOM nodes, weighting each by its severity:

$$p_i = \sum_{v\in\mathcal{V}(o_i)} \text{N}_v \cdot w_v \tag{2}$$

where severity weights are $w_v \in \{0.1, 0.2, 0.3, 0.4\}$ corresponding respectively to Minor, Moderate, Serious, and Critical violations. This design prioritizes the mitigation of more severe accessibility failures; we empirically validate the effectiveness of severity-weighted penalties in 6.4.1. We then convert this into a bounded reward by subtracting the penalty from a base score B, where we use $B = 2.0$, and clip the reward to zero for negative values.

$$r_i = \max(0,\ B - p_i) \tag{3}$$

Under this quantitative reward signaling scheme, a violation-free output converges toward $r_i \approx B$, and each violation proportionally lowers the reward. By assigning larger negative weights to more severe issues, the policy is encouraged to eliminate severe failures first. In practice, the policy model receives a solid numerical score that reflects the accessibility testing environment that is appropriate for giving RL feedback.

3.3 Training pipeline

We instantiate A11yn as a GRPO based reinforcement learning pipeline with *rollout-reward–update* cycle that repeatedly steers the policy toward accessibility-compliant code as illustrated in fig. 2. We use Qwen2.5-Coder-7B-Instruct (Hui et al., 2024) as our policy model π_θ and simultaneously use its frozen copy as the reference model π_{ref}, since it is pre-trained and capable of generating web contents based on natural language request. In each iteration, a textual UI request prompt q from training prompt set (section 4.1) is retrieved. The current policy π_θ then generates a group of G candidate completions $\{o_i\}_{i=1}^{G}$, producing diverse web UI code alternatives for the same prompt. Each completion is evaluated with Axe-core (Deque Systems, 2015), where generated web contents are rendered and analyzed for WCAG violations. The

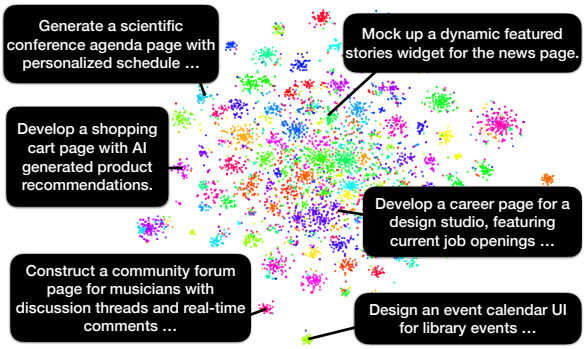


Figure 3: **t-SNE visualization of UIReq-6.8K demonstrating dataset diversity.** Each point corresponds to a UI request, colored by application category, highlighting broad coverage across domains and interaction types.

detected violations are converted into scalar penalties using the severity-weighted mapping described in section 3.2, yielding an accessibility reward r_i for each completion. Group statistics, mean $\bar{r}$ and standard deviation σ are computed to form normalized advantages. Then, these group-normalized advantages focus updates on relative improvements among sampled completions, favoring codes with minimal WCAG violations in the same group.

4 Data

4.1 Training: UIReq-6.8K

To train A11yn, we construct UIReq-6.8K, a reinforcement learning training dataset of 6,800 UI generation instructions. As shown in fig. 3, the dataset spans a wide range of domains and interaction patterns, supporting broad coverage of instruction types. Unlike supervised datasets, UIReq-6.8K does not fix target UIs for each request, which enables exploration and reward optimization without imposing stylistic bias.

Each instruction prompt in UIReq-6.8K describes a desired user interface in natural language, specifying page type, application domain, specific web UI components, or stylistic intent (e.g. a dark-themed login screen with email and password inputs). The instruction prompts are generated using GPT-4o-mini (OpenAI et al., 2024) and guided to reflect diversity and semantic richness. Diversity is achieved by covering 68 application categories (section A). Semantic richness is enforced through detailed requirements in instruction prompt synthesis, where every request specifies its page type, application domain, specific web UI components, and stylistic intent.

4.2 Evaluation: RealUIReq-300

We assess the accessibility of web UI within the broader scope of natural language to web UI code generation flow. To achieve this, a realistic request-style benchmark dataset was required, one that could capture authentic user intents and interface specifications instead of relying on fully synthetic or overly simplified captions.

In prior UI generation research, evaluation often relies on datasets built for natural language descriptions of user interfaces, with Screen2Words (Wang et al., 2021) being a popular example. Such datasets frame the task around short, descriptive summaries of existing UI screens. The descriptions are terse and taxonomic (e.g., "sign-in page of a social app" or "page displaying data status"), rather than request-oriented specifications that encode user intent or concrete requirements. Consequently, evaluation based on such summaries emphasizes surface-level correspondence over faithful satisfaction of UI specifications, limiting their suitability for assessing end-to-end UI generation quality.

To address these limitations, we introduce RealUIReq-300, a benchmark of 300 realistic web UI requests derived from manually collected webpage screenshots. As illustrated in fig. 4, each example is constructed through a multi-stage, human-in-the-loop pipeline (section D) that includes screenshot collection, metadata extraction, and request formulation. GPT-4.1 (OpenAI et al., 2024) is used in an assistive role for metadata extraction and drafting candidate requests, helping to expand coverage beyond manual annotation alone. All generated metadata and requests are subsequently reviewed and refined by human annotators to correct truncation, ambiguity, and missing context.

As summarized in table 1, RealUIReq-300 provides multi-sentence, request-style instructions that explicitly specify user intent, page type, required UI components, and application domain. This design yields a semantically rich and structurally aligned evaluation set, well suited for assessing natural language–driven web UI generation and accessibility.

5 Experimental Setup

We evaluate A11yn and baseline LLMs by measuring the accessibility violations in their generated web UIs. In addition, we assess whether A11yn can balance accessibility with semantic alignment

Dataset	Query Style	UI Intent Coverage	Component Details	UI Source	Query Length	Number of Queries
Screen2Words (Wang et al., 2021)	Single sentence, Taxonomic	✗	✗	RICO dataset (Android UI)	6 words	112k
RealUIReq-300 (Ours)	Multi-sentence, Request-oriented	✓	✓	Real-world Web UI	87 words	300

Table 1: **Comparison of REALUIREQ-300 with Screen2Words.** REALUIREQ-300 offers multi-sentence requests with structured intent, detailed UI component specification, and realistic phrasing grounded in real-world web UIs.

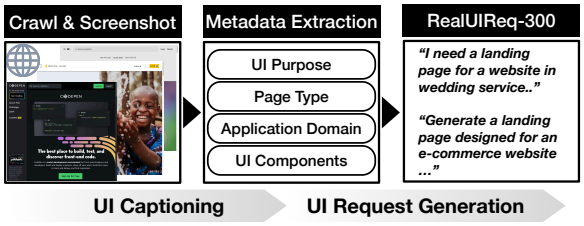


Figure 4: **RealUIReq-300 is curated from real web UIs with diverse use-case domains.** User requests are inversely generated from screenshots and metadata extracted, then refined to produce realistic instructions aligned with the original UIs.

and visual appeal, as analyzed in section 6.4.3. Each model is tested on RealUIReq-300 benchmark section 4.2, an evaluation set of 300 web UI request prompts, designed to ensure consistent and controlled comparisons. We use a low sampling temperature (0.1) for all models to balance reproducibility with limited exploration that mitigates repetitive failure modes.

5.1 Metrics

To assess the accessibility of model-generated responses, we adopt a comprehensive set of evaluation metrics informed by the principles of the Web Content Accessibility Guidelines (WCAG) from the accessibility auditing tool (Deque Systems, 2015). Our evaluation comprises three main metrics designed for robustness and fairness. First, we measure the average DOM counts with accessibility violations detected across the evaluation set, categorized by severity: *Minor*, *Moderate*, *Serious*, and *Critical*. Each severity level reflects the impact of the violation on user experience, ranging from minor impact issues to critical barriers that largely hinder accessibility.

To account for the varying severity of accessibility violations, we propose **Weighted Violation Score (WVS)**, which quantifies accessibility violations by assigning severity-based weights to affected DOM nodes at each severity category. The WVS is formally defined as:

$$\begin{aligned}\text{WVS} = \lambda_{\text{Minor}} \cdot N_{\text{Minor}} + \lambda_{\text{Moderate}} \cdot N_{\text{Moderate}} \\ + \lambda_{\text{Serious}} \cdot N_{\text{Serious}} + \lambda_{\text{Critical}} \cdot N_{\text{Critical}}\end{aligned} \tag{4}$$

where N_{Minor}, N_{Moderate}, N_{Serious}, and N_{Critical} represent the number of violated DOM counts at each severity level from the generated code with RealUIReq-300 request prompts. The corresponding weights λ reflect the relative impact of each category, with values of 1 for *Minor*, 2 for *Moderate*, 3 for *Serious*, and 4 for *Critical*. This formulation provides a single interpretable metric that captures both the frequency and severity of issues.

Inspired by the *Inaccessibility Rate* proposed in Feeda11y (Suh et al., 2025), we adopt the normalized metric to fairly compare models that generate web contents with varying lengths. Specifically, we calculate the ratio of WVS to the total number of DOM elements produced.

$$\text{IR} = \frac{\text{WVS}}{\text{No. of Total DOM Elements}} \tag{5}$$

This metric captures the normalized, severity-adjusted density of accessibility violations, allowing us to evaluate the true accessibility in proportion to UI complexity.

5.2 Baselines

We compare A11yn against five baselines: (1) Qwen2.5-Coder-7B-Instruct serves as the backbone from which A11yn is policy optimized. It reflects the model's raw web UI code generation capability in zero shot setting without any accessibility optimization. (2) Qwen2.5-Coder-7B-Instruct *(+Feeda11y)* is used to examine the impact of accessibility-aware prompting. It incorporates Feeda11y (Suh et al., 2025) prompts using a three-step iterative ReAct prompting (Yao et al., 2023) method with violation feedback from Axe-core. (3)

Model	Average Violated DOM Counts (↓)				WVS (↓)	IR (↓)
	Minor	Moderate	Serious	Critical		
Qwen2.5-Coder-7B-Instruct	1 (±1)	1149 (±36)	978 (±48)	40 (±0)	5392 (±35)	0.38 (±0.0)
+ Feeda11y	11 (±1)	461 (±37)	841 (±33)	30 (±5)	3576 (±64)	0.21 (±0.0)
Qwen2.5-Coder-14B-Instruct	3 (±2)	846 (±35)	1491 (±32)	49 (±8)	6365 (±158)	0.43 (±0.0)
GPT-4.1	45 (±5)	1925 (±34)	1424 (±21)	105 (±7)	8588 (±100)	0.27 (±0.0)
Claude Sonnet 4	2 (±3)	3388 (±31)	1435 (±14)	282 (±10)	12210 (±117)	0.29 (±0.0)
A11yn (Ours)	**0** (±0)	**231** (±16)	**481** (±23)	**24** (±3)	**1918** (±65)	**0.15** (±0.0)

Table 2: **Accessibility measures across models.** We report Average Violated DOM Counts at different severity levels. Weighted Violation Score (WVS) and Inaccessibility Rate (IR) provide severity-adjusted and normalized aggregate measures, respectively. Lower values indicate better performance. Best results are shown in **bold**, and second-best in underline.

Qwen2.5-Coder-14B-Instruct is included to assess the effect of model scaling, offering a larger candidate from the same model family. In addition, we evaluate two frontier models: (4) GPT-4.1 (OpenAI et al., 2024) and (5) Claude Sonnet 4 (Anthropic, 2025), which represent strong baselines in general-purpose code and web UI generation.

6 Results

6.1 Quantitative Results

Table 2 summarizes the accessibility performance of A11yn and five baselines. Despite their overall strength, frontier models such as GPT-4.1 and Claude Sonnet 4 exhibit relatively high inaccessibility rates (0.27 and 0.29), indicating that model scale does not guarantee accessible UI generation.

A11yn achieves the best performance across all metrics, attaining the lowest Weighted Violation Score (WVS) and Inaccessibility Rate (IR). Compared to the base model, A11yn reduces critical violations from 40 to 24 (40% ↓), serious violations from 978 to 481 (50.8% ↓), and moderate violations from 1149 to 231 (79.9% ↓). Overall, this corresponds to a 64.4% reduction in WVS (5392 → 1918) and a 60.5% reduction in IR (0.38 → 0.15), demonstrating substantial gains in accessibility compliance.

Prompt-based improvement via Feeda11y yields moderate benefits, achieving a WVS of 3576 and an IR of 0.21, but still underperforms A11yn. Moreover, Feeda11y incurs significant computational overhead due to iterative prompting, averaging 4,584 intermediate tokens per request.

6.2 Category-level violation analysis

Figure 5 shows the distribution of WCAG violation categories and highlights which types of accessibility failures are mitigated by A11yn. The most substantial reductions occur in structural violations related to semantic landmarks. In particular, region-related violations decrease from 894 to 143, indicating more consistent use of semantic landmarks such as <main>, <nav>, and <header>, which provide navigable structure for screen readers. Violations involving multiple or missing main landmarks are also reduced markedly, from 164 to 17, reflecting clearer document hierarchies.

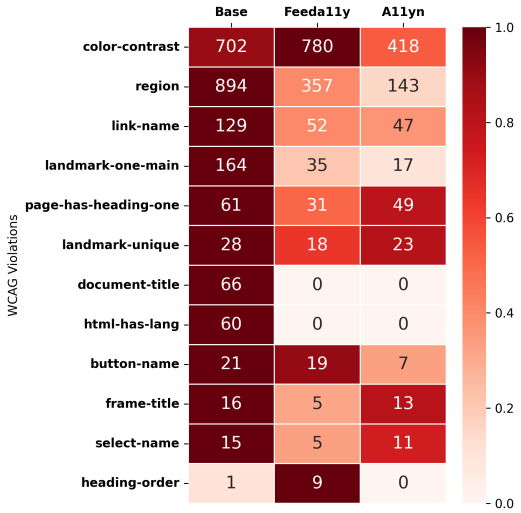


Figure 5: **Per-category violation counts across models.** Colors are normalized per row for visualization. (lighter = fewer violations, darker = more violations).

A11yn further improves visual accessibility by reducing color contrast violations from 702 to 418. These violations commonly hinder low-vision users when text fails to meet the WCAG-recommended 4.5:1 contrast ratio, and their reduction indicates improved adherence to visual accessibility standards. In addition, link-name violations decrease from 129 to 47, suggesting more consistent generation of descriptive link text that is interpretable by

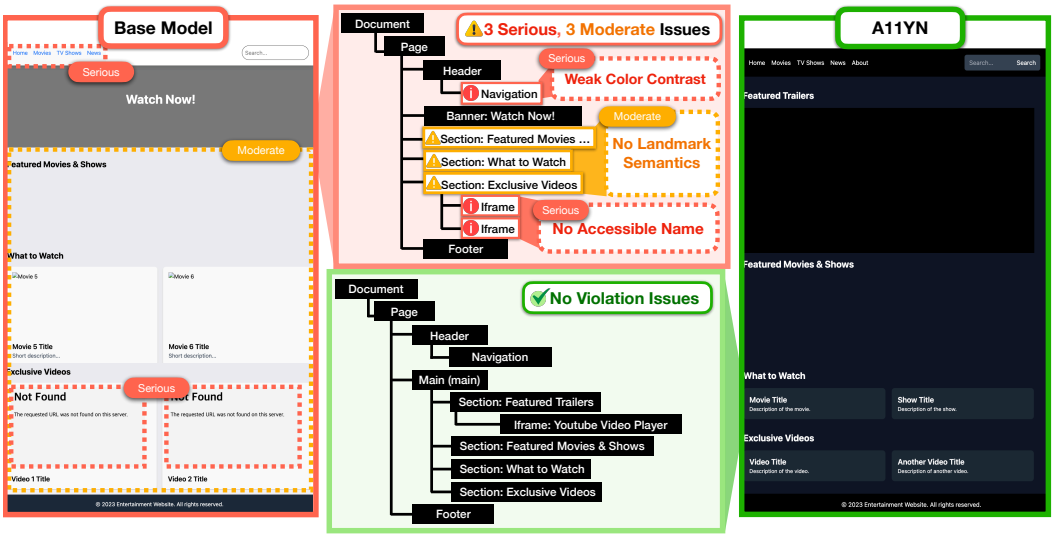


Figure 6: **Comparison of the base model (left) and A11yn (right) using both Rendered UIs and their corresponding Accessibility Trees (middle).** The accessibility tree reveals the base model's weak color contrast, missing landmarks, and unlabeled media nodes, while A11yn provides clear landmarks, stronger contrast, and descriptive accessible names, resulting in a clean, fully interpretable structure.

screen readers.

Overall, these distributional shifts demonstrate that A11yn systematically improves both structural and visual aspects of accessibility rather than targeting isolated failure modes.

6.3 Qualitative Example

Figure 6 illustrates a representative example pair comparing the rendered UI and its corresponding accessibility tree for the base model and A11yn. Beyond UI visuals, the *accessibility tree* provides a structural view of how assistive technologies like screen readers interpret and navigate the UI page.

The base model (left) exhibits violations across severity levels as reflected in its accessibility tree. The weak color contrast (serious) issue of the text in the navigation bar hinders reliable perception for low-vision users. The missing landmark semantics (moderate) issue leaves sections such as "Featured Movies & Shows" without clear structural roles, disrupting screen reader navigation. The iframe elements lacking accessible names cause media contents like embedded videos to be exposed without meaningful descriptions for non-sighted users.

By contrast, A11yn (right) produces UI with strong color contrast, sections clearly defined in Header–Main–Footer landmarks, and iframe elements annotated with descriptive names. The tree reflects accessibility-directed fixes consistent with the reward optimization objective, demonstrating that A11yn enhances both the visual presentation and the assistive-tool interpretation of the web UI.

6.4 Analysis

6.4.1 Ablation Study

Scheme	Violated DOM Counts (↓)			
	Minor	Moderate	Serious	Critical
Base	0	1130	786	46
Uniform	3	422	474	30
Weighted	**0**	**244**	**472**	**23**

Table 3: **Ablation of reward weighting strategies**. *Weighted* penalties more effectively affect high-impact WCAG violations than *Uniform* weighting.

To evaluate the impact of severity weighting, we compare two A11yn variants: a *weighted* reward that penalizes WCAG violations by severity and a *uniform* reward that assigns equal penalties. The weighted variant uses coefficients $(0.1, 0.2, 0.3, 0.4)$ for Minor, Moderate, Serious, and Critical violations, while the uniform variant uses $(0.25, 0.25, 0.25, 0.25)$. As shown in Table 3, severity-weighted rewards consistently reduce accessibility violations more effectively than uniform weighting, validating the design in section 3.2.

6.4.2 Validation across accessibility auditors

Accessibility auditing tools differ in rule coverage and detection heuristics, raising concerns that results may depend on a specific auditor. To

Model	Achecker Violations (↓)	Violations per Page (↓)
Base	2891	9.64
Feeda11y	2576	8.59
A11yn (Ours)	**1347**	**4.49**

Table 4: **Cross-tool validation with AChecker.** A11yn also shows a consistent reduction in AChecker violations compared to the base model and Feeda11y, confirming robustness across auditing tools.

assess robustness, we follow prior accessibility and HCI work (Suh et al., 2025; Pool, 2023) and evaluate A11yn using a second, independent tool, AChecker (Gay and Li, 2010). While Axe-core provides severity-level breakdowns, AChecker reports aggregate violation counts.

Table 4 shows that A11yn consistently reduces accessibility violations under AChecker as well. The relative improvements over both the base model and the prompting-based Feeda11y baseline closely mirror those observed with Axe-core, indicating that A11yn's gains are robust and not artifacts of a particular auditing tool.

6.4.3 Retaining semantics and aesthetics

Model	Inaccessibility Rate (↓)	Appearance Score (↑)
Base model	0.38	3.6/5
Feeda11y	0.21	**3.7/5**
A11yn (Ours)	**0.15**	3.6/5

Table 5: Comparison of models in terms of accessibility (Inaccessibility Rate) and aesthetics (Appearance Score on a 5-point Likert scale).

Web UI generation is inherently multi-dimensional, requiring trade-offs among accessibility, semantic fidelity, and visual quality. Although accessibility and aesthetics are often viewed as conflicting (Anthony, 2019), prior work suggests that they should be balanced rather than treated as opposing objectives (Kurosu and Kashimura, 1995; Mbipom and Harper, 2011; Le-Cong et al., 2021). Accordingly, while our work focuses on improving accessibility, we also evaluate whether these gains preserve semantic fidelity and visual appeal. This analysis ensures that accessibility improvements do not come at the expense of other key interface qualities.

Here, we adopt the Appearance Score metric from WebGen-Bench (Lu et al., 2025), a 5-point Likert scale rated by GPT-4.1 on rendering quality, content relevance, layout harmony, and modernity. The Appearance Score serves as a core metric capturing both semantic fidelity and aesthetics. As shown in table 5, A11yn achieves the lowest inaccessibility rate, a 60.5% reduction over the base model, with appearance score of 3.6 maintained. This shows that A11yn substantially improves accessibility while preserving aesthetics and fidelity, achieving a balanced outcome.

By comparison, Feeda11y achieves a higher appearance score (3.7) but retains a relatively high inaccessibility rate (0.21). This indicates that Feeda11y's improvements incidentally enhance visual quality rather than systematically addressing accessibility, reflecting a shifted emphasis. In contrast, A11yn achieves lower inaccessibility rate (0.15) while maintaining the Appearance Score intact (3.6), offering stronger evidence of accessibility enhancement with balance. Moreover, A11yn attains such outcome in a single forward pass, whereas Feeda11y relies on iterative prompting.

7 Conclusion

Accessibility remains a critical yet under-optimized aspect of LLM-based web UI generation. We introduced **A11yn**, a reward-driven alignment framework that trains code-generating LLMs to natively produce more accessible web interfaces using a severity-aware WCAG-based reward. Through reinforcement learning, A11yn significantly reduces accessibility violations while preserving semantic correctness and visual fidelity.

Our results show that accessibility can be integrated as a learnable behavior within the LLM generation pipeline rather than enforced through external tooling or post-hoc correction. More broadly, our findings suggest that formal, standards-driven objectives such as accessibility can be effectively incorporated into LLM alignment. We believe this approach extends beyond web UI generation and may generalize to other human-computer interaction domains, including mobile applications, AR/VR systems, and multimodal interactive environments.

8 Limitations & Future Directions

Reliance on Accessibility Audits. This work evaluates accessibility using automated auditing

tools such as Axe-core and AChecker, without direct human-centered assessment. Fully capturing the diverse and context-dependent accessibility needs of users through human evaluation remains a challenging and open problem in HCI research and is left for future work.

Scope of HTML-based Web UIs. The current study focuses on HTML-based web UI generation, which covers a broad and practical class of real-world interfaces. Nevertheless, extending accessibility-aligned optimization to additional frontend frameworks and UI technologies remains an important direction as accessibility requirements may vary across platforms.

Targeting of Static UI. Our analysis considers static UI renderings derived from screenshots and does not explicitly model interactive behaviors. As some accessibility challenges arise from runtime interactions, future work may explore dynamic and interactive web interfaces, including accessibility considerations under UI state changes and user interactions.

References

Ammar Ahmed, Margarida Fresco, Fredrik Forsberg, and Hallvard Grotli. 2025. From code to compliance: Assessing chatgpt's utility in designing an accessible webpage – a case study. *Preprint*, arXiv:2501.03572.

Wajdi Aljedaani, Abdulrahman Habib, Ahmed Aljohani, Marcelo Eler, and Yunhe Feng. 2024. Does chatgpt generate accessible code? investigating accessibility challenges in llm-generated source code. In *Proceedings of the 21st International Web for All Conference*, pages 165–176.

Anthony. 2019. The aesthetic-accessibility paradox. https://uxmovement.com/thinking/the-aesthetic-accessibility-paradox/. Accessed: 2025-07-29.

Anthropic. 2025. Claude opus 4 & claude sonnet 4: System card. Technical report, Anthropic. Accessed: 2025-07-28.

Elisa Calì, Tommaso Fulcini, Riccardo Coppola, Lorenzo Laudadio, and Marco Torchiano. 2025. A prototype vs code extension to improve web accessible development. In *2025 IEEE/ACM Second IDE Workshop (IDE)*, pages 52–57. IEEE.

Deque Systems. 2015. axe-core Accessibility Engine. https://github.com/dequelabs/axe-core. Accessed: 2025-07-28.

Greg Gay and Cindy Qi Li. 2010. Achecker: open, interactive, customizable, web accessibility checking. In *Proceedings of the 2010 International Cross Disciplinary Conference on Web Accessibility (W4A)*, W4A '10, New York, NY, USA. Association for Computing Machinery.

Alexandra-Elena Guriţă and Radu-Daniel Vatavu. 2025. When llm-generated code perpetuates user interface accessibility barriers, how can we break the cycle. In *Proceedings of the 22nd International Web for All Conference (W4A'25)*.

Calista Huang, Alyssa Ma, Suchir Vyasamudri, Eugenie Puype, Sayem Kamal, Juan Belza Garcia, Salar Cheema, and Michael Lutz. 2024. Access: Prompt engineering for automated web accessibility violation corrections. *Preprint*, arXiv:2401.16450.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, Kai Dang, Yang Fan, Yichang Zhang, An Yang, Rui Men, Fei Huang, Bo Zheng, Yibo Miao, Shanghaoran Quan, and 5 others. 2024. Qwen2.5-coder technical report. *Preprint*, arXiv:2409.12186.

Masaaki Kurosu and Kaori Kashimura. 1995. Apparent usability vs. inherent usability: experimental analysis on the determinants of the apparent usability. In *Conference companion on Human factors in computing systems*, pages 292–293.

Thanh Le-Cong, Xuan Bach D Le, Quyet Thang Huynh, and Phi Le Nguyen. 2021. Usability and aesthetics: Better together for automated repair of web pages. In *2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*, pages 173–183. IEEE.

Zimu Lu, Yunqiao Yang, Houxing Ren, Haotian Hou, Han Xiao, Ke Wang, Weikang Shi, Aojun Zhou, Mingjie Zhan, and Hongsheng Li. 2025. Webgen-bench: Evaluating llms on generating interactive and functional websites from scratch. *Preprint*, arXiv:2505.03733.

Beatriz Martins and Carlos Duarte. 2024. A large-scale web accessibility analysis considering technology adoption. *Universal Access in the Information Society*, 23(4):1857–1872.

Grace Mbipom and Simon Harper. 2011. The interplay between web aesthetics and accessibility. In *The proceedings of the 13th international ACM SIGACCESS conference on Computers and accessibility*, pages 147–154.

Kevin Moran, Carlos Bernal-Cárdenas, Michael Curcio, Richard Bonett, and Denys Poshyvanyk. 2018. Machine learning-based prototyping of graphical user interfaces for mobile apps. *IEEE transactions on software engineering*, 46(2):196–221.

Peya Mowar, Yi-Hao Peng, Jason Wu, Aaron Steinfeld, and Jeffrey P Bigham. 2025. Codea11y: Making ai

coding assistants useful for accessible web development. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, pages 1–15.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2024. Gpt-4 technical report. *Preprint*, arXiv:2303.08774.

Jonathan Robert Pool. 2023. Accessibility metatesting: Comparing nine testing tools. In *Proceedings of the 20th International Web for All Conference*, W4A '23, page 1–4, New York, NY, USA. Association for Computing Machinery.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *Preprint*, arXiv:1707.06347.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *Preprint*, arXiv:2402.03300.

Hyunjae Suh, Mahan Tafreshipour, Sam Malek, and Iftekhar Ahmed. 2025. Human or llm? a comparative study on accessible code generation capability. *Preprint*, arXiv:2503.15885.

Bryan Wang, Gang Li, Xin Zhou, Zhourong Chen, Tovi Grossman, and Yang Li. 2021. Screen2words: Automatic mobile ui summarization with multimodal learning. *Preprint*, arXiv:2108.03353.

Jason Wu, Eldon Schoop, Alan Leung, Titus Barik, Jeffrey Bigham, and Jeffrey Nichols. 2024. UICoder: Finetuning large language models to generate user interface code through automated feedback. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 7511–7525, Mexico City, Mexico. Association for Computational Linguistics.

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. *Preprint*, arXiv:2506.06251.

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

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

A UIReq-6.8K: Application Domains

The training dataset covers 68 diverse application categories. Examples include:

Training Set Application domains

- Business & Enterprise
- Health & Wellness
- Education & E-Learning
- Data & Analytics
- Communication & Social
- E-Commerce & Retail
- Finance & FinTech
- Real Estate & Property
- Media & Entertainment
- Food & Beverage
- Travel & Hospitality
- Developer Tools & Technology
- Science & Research
- Legal & Compliance
- Automotive & Mobility
- Government & Public Services
- Environment & Sustainability
- Security & Identity
- Non-Profit & Social Impact
- AI & Machine Learning
- Books & Reference
- Comics
- Dating
- Entertainment
- Events
- Finance
- Food & Drink
- Health & Fitness
- House & Home
- Libraries & Demo
- Lifestyle
- Maps & Navigation
- Medical
- Music & Audio
- News & Magazines
- Parenting
- Personalization
- Photography
- Productivity
- Shopping
- Social
- Sports
- Tools
- Travel & Local
- Video Players & Editors
- Weather
- Auto & Vehicles
- Beauty
- Art & Design
- Board
- Card
- Casino
- Casual
- Educational (Games)
- Music (Games)
- Puzzle
- Racing

- Role Playing
- Simulation
- Sports (Games)
- Strategy
- Trivia
- Word
- Augmented Reality
- Developer Tools
- Magazines & Newspapers
- Utilities
- Graphics & Design

B RealUIReq-300 Domain Distribution

61 Domain Distribution of RealUIReq-300 dataset

Education 30

E-commerce 19

Non-profit / Humanitarian 13

Social media 12

Web development 10

Entertainment / Media 9

Business consulting 6

Academic publishing 6

Personal development 5

Finance / Investing 4

Government services 4

Environmental activism 4

Community forum 4

B2B marketing / Business services 4

Business Collaboration 3

Medical research 3

Financial Services 3

Crowdfunding 2

Music streaming 2

Wedding planning 2

Creative networking 2

Parenting / Community 2

Advocacy / Activism 2

Cloud computing / AI 2

Search engine 1

Culture / History 1

Freelance marketplace 1

Service industry 1

Microfinance 1

Space exploration 1

Religious / Charitable Org 1

Technology / Software 23

News / Media 16

Software development 12

Tourism 10

SaaS 9

Entertainment ticketing 8

Professional services 6

Food / Cooking 5

Travel / Lifestyle 5

Tech discovery platforms 4

Academic research 4

Health and wellness 4

Art and design 4

Creative / Portfolio 3

Content publishing 3

Personal blog 3

Consumer electronics 2

Adventure / Outdoor sports 2

Education / Interactive media 2

Events / Entertainment 2

Film / Photography 2

Job board / Recruitment 2

Healthcare tech 2

Retail / Fashion 1

Dev social 1

Women empowerment 1

Science museum 1

Gaming / Media 1

Cloud storage 1

Humor / Lifestyle 1

C Training Configuration

We trained *Qwen/Qwen2.5-Coder-7B-Instruct* on 8 NVIDIA A6000 GPUs (48GB VRAM each) using GRPO with vLLM-based sampling and reward modeling. Training was conducted in bfloat16 mixed-precision with gradient checkpointing enabled. Each prompt was expanded into $G = 6$ sampled completions, with a per-device batch size of 2 and gradient accumulation set to 6. To stabilize optimization, KL divergence regularization was applied with $\beta = 0.001$. Below is a summary of the key configurations used; for full details, please refer to the provided training scripts and configuration files.

Component	**Configuration**
Framework	HF Transformers; TRL (GRPOTrainer); Accelerate; DeepSpeed
Learning rate	5×10^{-5}
Optimizer	Adam (β_1=0.9, β_2=0.99, ϵ=10^{-8})
Weight decay	0.1
LR scheduler	Cosine (warmup ratio 0.01)
Gradient accumulation	6
Batch size	96 (2 per device $\times$ 8 processes $\times$ 6 accumulation)
Epochs	2
Gradient checkpointing	Enabled
Precision	bfloat16
Loss type	GRPO
KL β	0.001
PEFT	Enabled

Table 6: Optimization and training parameters.

Parameter	**Value**
Engine	vLLM (colocated mode)
GPU memory utilization	60%
Min-p	0.1
Top-p	1.0
Top-k	-1
Temperature	0.7
Repetition penalty	1.1
Stop sequence	`</answer>`
Max tokens	3072
Number of generations	6
Seed	3407

Table 7: GRPO sampling parameters.

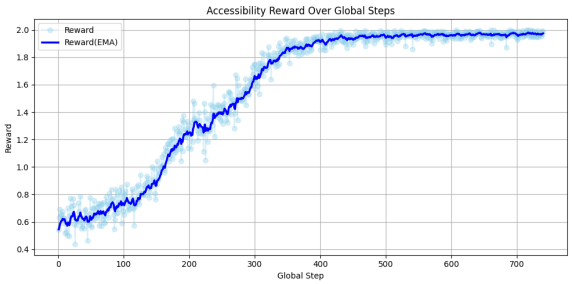


Figure 7: **Accessibility reward curve throughout training**, showing a steady increase that indicates reduced WCAG violation occurrences over time.

D Data Annotation of RealUIReq-300

The construction of RealUIReq-300 involved human-in-the-loop intervention to ensure semantic accuracy and realism of evaluation requests. Human contributors consisted of the paper authors. Human involvement occurred at three stages:

- **UI Captioning Validation.** LLM-generated metadata extracted from webpage screenshots (e.g., UI purpose, page type, domain, and key components) was manually reviewed to verify that notable and functionally important UI elements were correctly captured. Minor corrections were applied to address omissions or inaccuracies.
- **UI Request Draft Editing.** Initial request drafts were edited to improve clarity, specificity, and naturalness, resolving vague phrasing and ensuring alignment with the underlying UI content.
- **Consistency and Coherence Checking.** Final requests were reviewed to ensure consistency and semantic coherence between the request texts and the corresponding real-world UIs, including alignment with the intended UI purpose, structure, and static web UI scope.

E Use of AI Assistants

We employed AI assistants to enhance the clarity and accuracy of our writing, particularly in identifying and correcting grammatical errors, typographical mistakes, and rephrasing sentences for improved readability.

F Icon Attribution

Icons in the figures are sourced from Flaticon (`https://www.flaticon.com`) and are credited to their respective creators in accordance with Flaticon's licensing requirements.

G WCAG Violations

Below is the full list of WCAG violations.

Rule ID	Impact	Description
aria-allowed-attr	Serious, Critical	ARIA attributes are allowed for an element's role
aria-command-name	Serious	Every ARIA button, link, and menuitem has an accessible name
aria-hidden-body	Critical	`aria-hidden='true'` is not present on the <body> element
aria-hidden-focus	Serious	`aria-hidden` elements are not focusable nor contain focusable elements
aria-input-field-name	Moderate, Serious	Every ARIA input field has an accessible name
aria-meter-name	Serious	Every ARIA meter node has an accessible name
aria-progressbar-name	Serious	Every ARIA progressbar node has an accessible name
aria-required-attr	Critical	Elements with ARIA roles have all required ARIA attributes
aria-required-children	Critical	Elements with ARIA roles that require child roles contain them
aria-required-parent	Critical	Elements with ARIA roles that require parent roles are contained by them
aria-roles	Minor–Critical	Elements with a `role` use a valid value
aria-toggle-field-name	Moderate, Serious	Every ARIA toggle field has an accessible name
aria-tooltip-name	Serious	Every ARIA tooltip node has an accessible name
aria-valid-attr-value	Serious, Critical	All ARIA attributes have valid values
aria-valid-attr	Critical	Attributes beginning with `aria-` are valid ARIA attributes

Table 8: WCAG 2.0 — ARIA(Accessible Rich Internet Applications) Rules

Rule ID	Impact	Description
area-alt	Critical	<area> elements of image maps have alternate text
image-alt	Critical	<img> has alt text or role of none/presentation
input-image-alt	Critical	<input type=image> has alternate text
object-alt	Serious	<object> elements have alternate text
role-img-alt	Serious	`role="img"` elements have alternate text
svg-img-alt	Serious	<svg> with img/graphics roles have accessible text
video-caption	Critical	<video> elements have captions

Table 9: WCAG 2.0 — Text Alternatives & Captions

Rule ID	Impact	Description
bypass	Serious	Page has a mechanism to bypass navigation
nested-interactive	Serious	Interactive controls are not nested
scrollable-region-focusable	Serious	Scrollable content regions are keyboard accessible
server-side-image-map	Minor	Server-side image maps are not used

Table 10: WCAG 2.0 — Keyboard, Focus & Navigation

Rule ID	Impact	Description
frame-focusable-content	Serious	`<frame>/<iframe>` with focusable content do not have `tabindex=-1`
frame-title-unique	Serious	`<iframe>/<frame>` contain a unique `title` attribute
frame-title	Serious	`<iframe>/<frame>` have an accessible name

Table 11: WCAG 2.0 — Frames & Embeds

Rule ID	Impact	Description
button-name	Critical	Buttons have discernible text
input-button-name	Critical	Input buttons have discernible text
label	Minor–Critical	Every form element has a label
select-name	Minor–Critical	`<select>` has an accessible name
form-field-multiple-labels	Moderate	Form field does not have multiple label elements

Table 12: WCAG 2.0 — Forms & Names

Rule ID	Impact	Description
definition-list	Serious	<dl> elements are structured correctly
dlitem	Serious	<dt> and <dd> are contained by a <dl>
list	Serious	Lists are structured correctly
listitem	Serious	elements are used semantically
document-title	Serious	Each HTML document contains a non-empty <title>

Table 13: WCAG 2.0 — Structure & Semantics

Rule ID	Impact	Description
duplicate-id-active	Serious	Every id of active elements is unique
duplicate-id-aria	Critical	Every id used in ARIA and in labels is unique
duplicate-id	Minor	Every id attribute value is unique

Table 14: WCAG 2.0 — Parsing & Uniqueness

Rule ID	Impact	Description
color-contrast	Serious	Foreground/background colors meet WCAG 2 AA contrast thresholds
link-in-text-block	Serious	Links are distinguishable from surrounding text without relying on color
meta-viewport	Critical	<meta name="viewport"> does not disable text scaling and zooming
blink	Serious	<blink> elements are not used
marquee	Serious	<marquee> elements are not used
link-name	Serious	Links have discernible text

Table 15: WCAG 2.0 — Color & Visual Presentation

Rule ID	Impact	Description
html-has-lang	Serious	Document has a `lang` attribute
html-lang-valid	Serious	`lang` attribute on `<html>` has a valid value
html-xml-lang-mismatch	Moderate	`lang` and `xml:lang` agree on base language
valid-lang	Serious	`lang` attributes have valid values

Table 16: WCAG 2.0 — Language

Rule ID	Impact	Description
td-headers-attr	Serious	Cells using `headers` refer only to cells in the same table
th-has-data-cells	Serious	`<th>` and header roles have data cells they describe

Table 17: WCAG 2.0 — Data Tables

Rule ID	Impact	Description
meta-refresh	Critical	`<meta http-equiv="refresh">` is not used for delayed refresh
no-autoplay-audio	Moderate	`<video>` or `<audio>` elements do not autoplay audio for more than 3 seconds without a control mechanism to stop or mute the audio

Table 18: WCAG 2.0 — User Control & Timing

Rule ID	Impact	Description
aria-allowed-role	Minor	`role` attribute has an appropriate value for the element
aria-dialog-name	Serious	ARIA dialog/alertdialog nodes have accessible names
aria-text	Serious	`role=text` used only on elements with no focusable descendants
aria-treeitem-name	Serious	ARIA treeitem nodes have accessible names
presentation-role-conflict	Minor	Presentational elements do not have global ARIA or `tabindex`
label-title-only	Serious	Every form element has a visible label and is not solely labeled using hidden labels, or the title or aria-describedby attributes
tabindex	Serious	`tabindex` attribute values are not greater than 0

Table 19: Best Practices — ARIA(Accessible Rich Internet Applications)

Rule ID	Impact	Description
landmark-banner-is-top-level	Moderate	Banner landmark is top level
landmark-complementary-is-top-level	Moderate	Complementary/aside landmark is top level
landmark-contentinfo-is-top-level	Moderate	Contentinfo landmark is top level
landmark-main-is-top-level	Moderate	Main landmark is top level
landmark-no-duplicate-banner	Moderate	At most one banner landmark
landmark-no-duplicate-contentinfo	Moderate	At most one contentinfo landmark
landmark-no-duplicate-main	Moderate	At most one main landmark
landmark-one-main	Moderate	Document has a main landmark
landmark-unique	Moderate	Landmarks have unique role/name/title combinations
region	Moderate	All page content is contained by landmarks
skip-link	Moderate	All skip links have a focusable target

Table 20: Best Practices — Landmarks & Regions

Rule ID	Impact	Description
empty-heading	Minor	Headings have discernible text
heading-order	Moderate	Heading order is semantically correct
page-has-heading-one	Moderate	Page (or a frame) contains a level-one heading
empty-table-header	Minor	Table headers have discernible text
accesskeys	Serious	Every accesskey attribute value is unique
image-redundant-alt	Minor	Image alternative is not repeated as text
meta-viewport-large	Minor	`<meta name="viewport">` can scale a significant amount

Table 21: Best Practices — Headings & Structure

Rule ID	Impact	Description
scope-attr-valid	Moderate, Critical	`scope` attribute is used correctly on tables
table-duplicate-name	Minor	`<caption>` text differs from `summary` attribute
frame-tested	Critical	`<iframe>` and `<frame>` elements contain the axe-core script

Table 22: Best Practices — Tables

H Prompt Details

We provide the details of the prompt used.

H.1 Prompts for Dataset Synthesis

Prompt for Generating Training Data

```
Your task is to generate 100 different possible UI request
queries for a certain application domain category.

The application category is:
{category}
Requirements for queries:
1. Each query should include different page types possible
    within the app in the application domain
2. Each queries should be specific about widget requirements
    and context rich (side navigation, collapsible menus, etc.)
3. Each query must be different from the others
4. Queries should be realistic and natural like real user
    requests
5. Queries should be in some length and semantically rich,
    not just a few words
6. Queries should have solid use case or purpose, not random
    requests
7. Do not consider interactivity or animations or hovering
    effects focus on static UI elements
8. Queries require some ambience or style requests like
    color scheme, typography, etc.
```

Prompt for Captioning metadata for evaluation set curation

```
Caption given UI screenshot with
1. Main purpose and intent of the UI, regarding the target audience and
actual use case.
2. Page type of the UI, such as a landing page, blog post, product page, etc.
(in short-answer format)
3. Domain of the UI, such as e-commerce, social media, education, etc. (in short-
answer format)
4. Top 5 important visual elements in the UI design, that are crucial for more
user engagement and usability. But you should EXCLUDE any elements about the
EXACT IMAGES in the UI.
Answer in such format:
1. <Purpose/Intent>
2. <Page Type>
3. <Domain>
4. (a) <Element (a)>
(b) <Element (b)>
(c) <Element (c)>
...
```

Prompt for Generating RealUIReq-300 evaluation set queries

```
SYSTEM PROMPT:
You are a helpful assistant that generates realistic user
requests for web UI development.

USER PROMPT:
Based on the following web page specifications, generate a
user request that mentions EVERY DETAIL provided, including
the purpose, page type, domain, and all listed components.

Purpose: {purpose}
Page Type: {page_type}
Domain: {application_domain}
Required Components: {required_components}

The request should be 3-5 sentences long and sound realistic.
```

H.2 Prompts for Inference and Evaluation

Prompt for Inference of models for web UI generation

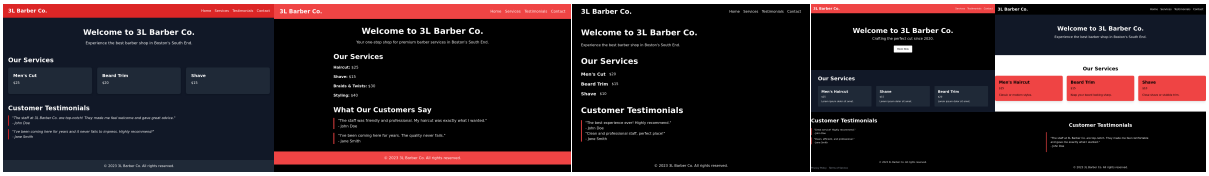
````
You are an expert UI designer assistant.
You should plan the design based on the user request.
Show the plan in the `<think>` tag.
    - You must think about the html structure and widgets
      needed to fulfill the user request.
    - You must think about the Tailwind CSS classes to use
      for styling.
Then, you should generate a complete HTML document that
includes:
    - A `<head>` section with a `<meta charset="UTF-8">` tag
    - A `<meta name="viewport" content="width=device-width,
      initial-scale=1.0">` tag
    - A proper tailwind css link tag to load Tailwind CSS
      from CDN
    - A `<body>` section that contains the complete HTML
      structure and content
The HTML document should be visually appealing, well-
structured, and content/semantically-rich.
You must strictly follow the output format shown below:
<think>
...
</think>

<answer>
```html
<html>
<head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width,
    initial-scale=1.0" />
    ...
</head>
<body>
...
</body>
</html>
```
</answer>

User: {user_request}
Assistant:
````

WebGen-Bench Appearance Score Evaluation Prompt (Lu et al., 2025)

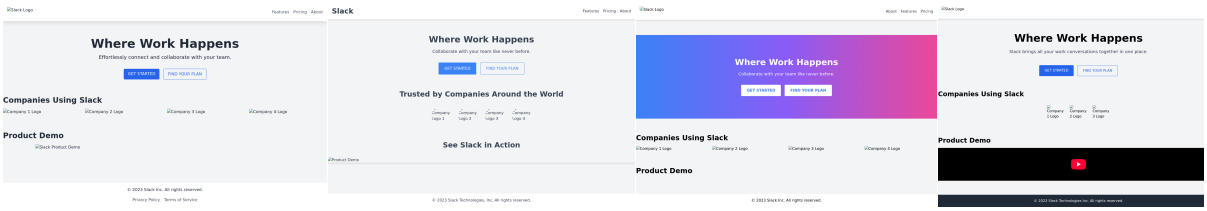
```
Instruction: You are tasked with evaluating the functional
design of a webpage that had been constructed based on the
following instruction: {instruction}

Grade the webpage's appearance on a scale of 1 to 5 (5 being highest), considering
the following criteria:
   - Successful Rendering: Does the webpage render correctly without visual
     errors? Are colors, fonts, and components displayed as specified?
   - Content Relevance: Does the design align with the webpage's purpose
     and user requirements? Are elements (e.g., search bars, report formats)
     logically placed and functional?
   - Layout Harmony: Is the arrangement of components (text, images, buttons)
     balanced, intuitive, and clutter-free?
   - Modernness & Beauty: Does the design follow contemporary trends (e.g.,
     minimalism, responsive layouts)? Are colors, typography, and visual
     hierarchy aesthetically pleasing?
Grading Scale:
   - 1 (Poor): Major rendering issues (e.g., broken layouts, incorrect colors).
     Content is irrelevant or missing. Layout is chaotic. Design is outdated or
     visually unappealing.
   - 2 (Below Average): Partial rendering with noticeable errors. Content is
     partially relevant but poorly organized. Layout lacks consistency. Design
     is basic or uninspired.
   - 3 (Average): Mostly rendered correctly with minor flaws. Content is relevant
     but lacks polish. Layout is functional but unremarkable. Design is clean but
     lacks modern flair.
   - 4 (Good): Rendered well with no major errors. Content is relevant and
     logically organized. Layout is harmonious and user-friendly. Design is modern
     and visually appealing.
   - 5 (Excellent): Flawless rendering. Content is highly relevant, intuitive,
     and tailored to user needs. Layout is polished, responsive, and innovative.
     Design is cutting-edge, beautiful, and memorable.

Task: Review the provided screenshot(s) of the webpage. Provide a detailed
analysis and then assign a grade (from 1 to 5) based on your analysis. Highlight
strengths, weaknesses, and how well the design adheres to the specifications, but
don't mind the absence of images or cards for specific data because they are not
the target for evaluation.
```

```
IMPORTANT: Please end your response with a clear grade in the format "Grade: X"
where X is a number from 1 to 5.

Your Response Format:
  Analysis: [from 2 to 4 paragraphs addressing all criteria, referencing the
  instruction]
  Grade: [from 1 to 5]
  Your Response:
```

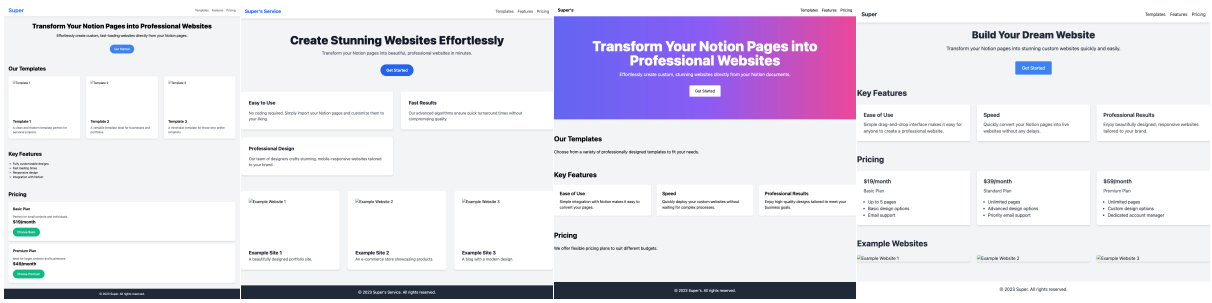
I Qualitative Web UI Samples of A11yn



I want a landing page for 3L Barber Co., a barbershop located in Boston's South End. The page should attract and inform potential customers about our services and atmosphere, featuring a bold, contrasting color scheme of black, white, and red. I'd like to see large, eye-catching typography for the main headline, along with a clear navigation menu that includes essential sections. Additionally, please include a detailed list of services with prices and a section for customer testimonials and ratings.



Please make a landing page designed to promote Slack as a collaborative workspace platform for teams. The page should feature a bold, attention-grabbing headline that says "Where Work Happens," along with prominent "GET STARTED" and "FIND YOUR PLAN" call-to-action buttons. Additionally, please include logos of well-known companies that use Slack, a clean and minimalist design with ample white space, and an interactive product demo or screenshot that showcases key features. The domain should focus on business communication and collaboration.



I want a landing page for Super's service that creates custom websites from Notion pages. The purpose of the page is to attract and inform potential users about the ease of use, speed, and professional results of our service. It should include a clear, prominent headline explaining our core value proposition, a concise subheading detailing key benefits, and a prominent call-to-action button to get started. Additionally, please include a navigation menu with sections like Templates, Features, and Pricing, along with example website previews to showcase our capabilities. The domain is web development/SaaS.

Figure 8: Qualitative web UI samples generated by A11yn for representative RealUIReq-300 prompts, illustrating diverse layout and styling outcomes.