Video2Code: Generating Interactive Webpages from UI Videos via Action-Aware Revisit

Anonymous ACL submission

Abstract

UI videos provide a natural input for generating interactive webpages, as they capture both webpage appearance and action-triggered state transitions. However, directly applying video-capable vision-language models to this task remains insufficient. Existing models typically rely on sparse sampling or compressed temporal representations, which may miss short action boundaries and break the state-action-state transitions needed to implement webpage behavior. We formulate UI video-to-code generation as executable state-transition recovery from interaction videos, and identify this failure mode as state-transition misalignment. We introduce **Video2Code**, an action-aware video-to-code approach for recovering executable UI state transitions. Rather than allocating the visual budget uniformly across the video, Video2Code first performs coarse video understanding to locate action-critical regions, then invokes a temporal clipping tool to revisit these regions at higher temporal resolution before generating HTML/CSS/JavaScript code. We instantiate Video2Code with action-aligned video-code supervision and evaluate it under both visual and functional criteria. Experiments show that Video2Code substantially strengthens the underlying open-source model for UI video-to-code generation, improving functional correctness over direct video observation, especially on dense multi-step interactions.

1 Introduction

Vision-language models (VLMs) have recently shown strong capabilities in multimodal understanding and code generation (Liu et al., 2023; Wang et al., 2024a,b; Bai et al., 2025b; Hong et al., 2025; Zhu et al., 2025). This progress has led to growing interest in UI-to-code generation, where models produce executable front-end code from webpage screenshots or design images (Si et al., 2024; Yun et al., 2024; Jiang et al., 2025; Gui et al., 2025b,a; Xu et al., 2025). However, most existing methods still operate on static screenshots, with the goal of reconstructing the layout and visual appearance of a webpage. Such a formulation captures what an interface looks like, but provides little information about how it behaves under user interaction.

Modern webpages often reveal their behavior only through interaction. A screenshot may show a collapsed menu or an initial viewport, but it cannot specify what happens after a click, selection, or scroll. Even multiple screenshots remain ambiguous when the action order and transition process are missing. UI videos therefore provide a natural input for interactive webpage generation, as they record both interface states and the temporal evidence connecting them. As shown in Figure 1, the key gap is not only visual reconstruction, but also recovering the behavior triggered by user actions.

However, video input alone is not sufficient. Current video-capable VLMs usually process videos with sparse frame sampling (Cheng et al., 2024; Chen et al., 2024; Cheng et al., 2025; Liu et al., 2025; Zhang et al., 2026) or compressed temporal representations (Li et al., 2024a; Zhang et al., 2025, 2024; Li et al., 2024b). While effective for many natural videos, such budget-limited observation is fragile for UI interaction videos, where code-relevant evidence is concentrated around action boundaries. For executable webpage generation, the key evidence is not an isolated informative frame, but the complete state-action-state transition around a user operation.

We therefore formulate UI video-to-code generation as *executable state-transition recovery* from interaction videos. Under budget-limited video observation, a model may suffer from *state-transition misalignment*: the sampled observations fail to preserve the complete state-action-state transition needed to implement webpage behavior. To address this challenge, we introduce **Video2Code**,

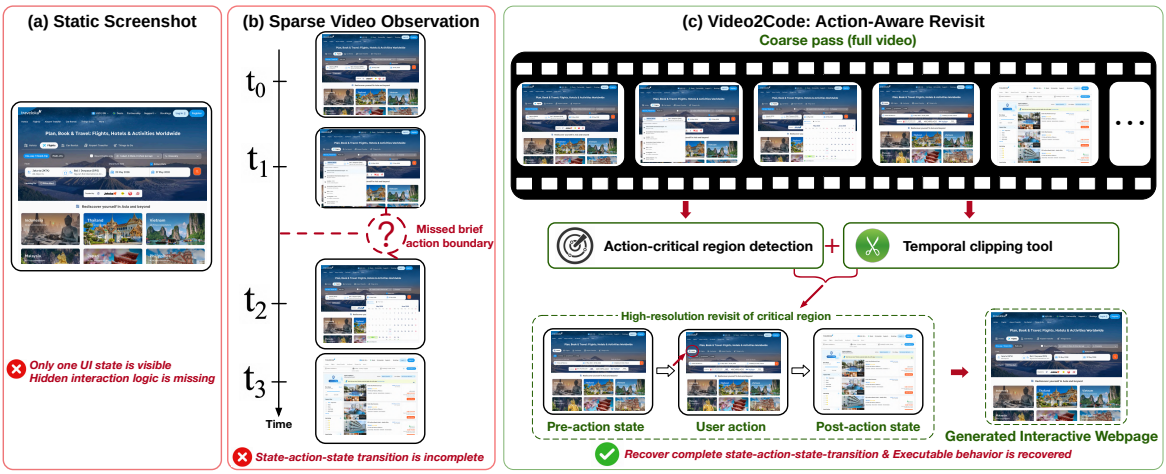


Figure 1: Motivating example of UI video-to-code generation. Static screenshots capture isolated UI states, while sparse video observation may miss short action boundaries. Video2Code revisits action-critical regions to recover complete state-action-state transitions for executable webpage behavior.

an action-aware video-to-code approach for recovering executable UI state transitions. Rather than allocating observation uniformly across the video, Video2Code first performs a coarse pass to locate action-critical regions, then invokes a temporal clipping tool to revisit these regions at higher temporal resolution before generating executable HTML/CSS/JavaScript code. This coarse-to-fine design preserves global context while recovering the local transition evidence needed for behavior-faithful webpage generation.

We instantiate Video2Code with action-aligned video-code supervision. Each UI interaction video is paired with temporal action anchors and an executable webpage implementation, enabling the model to learn both revisit planning and code generation from recovered transition evidence. We evaluate Video2Code under both visual and functional criteria, including browser-based interaction verification. Experiments show that Video2Code improves functional correctness and interaction coverage over single-pass video observation, especially on dense multi-step interactions.

Our contributions are summarized as follows:

- We formulate UI video-to-code generation as executable state-transition recovery from interaction videos, and identify *state-transition misalignment* as a key challenge of budget-limited video observation.
- We propose **Video2Code**, an action-aware video-to-code approach that uses a temporal clipping tool to revisit action-critical regions at higher temporal resolution before generating executable webpage code.
- We instantiate Video2Code with action-aligned video-code supervision and show that it improves functional correctness and interaction coverage over single-pass video observation.

2 Video2Code

In this section, we present **Video2Code**, an action-aware video-to-code approach for generating executable front-end code from UI interaction videos. As shown in Figure 2, Video2Code centers on *Action-Aware Revisit*: the model first performs a coarse pass over the full video to locate action-critical regions, then invokes a temporal clipping tool to revisit these regions at higher temporal resolution before generating the final webpage implementation.

2.1 Problem Setup

Given a UI interaction video $v = \{x_t\}_{t=1}^{T}$, where x_t denotes the frame at time t, Video2Code aims to generate an executable webpage implementation y, including HTML, CSS, and JavaScript. We view v as demonstrating a latent sequence of action-conditioned UI transitions:

$$\mathcal{Z}(v) = \{(s_i, a_i, s_{i+1}, \tau_i)\}_{i=1}^{N},$$

where s_i and s_{i+1} are the UI states before and after a user action a_i, and $\tau_i = (t_s^i, t_e^i)$ denotes the temporal window containing the transition.

A behavior-faithful implementation should reproduce both the visual states and the executable behavior: executing the same action a_i on the generated webpage should lead to a state matching

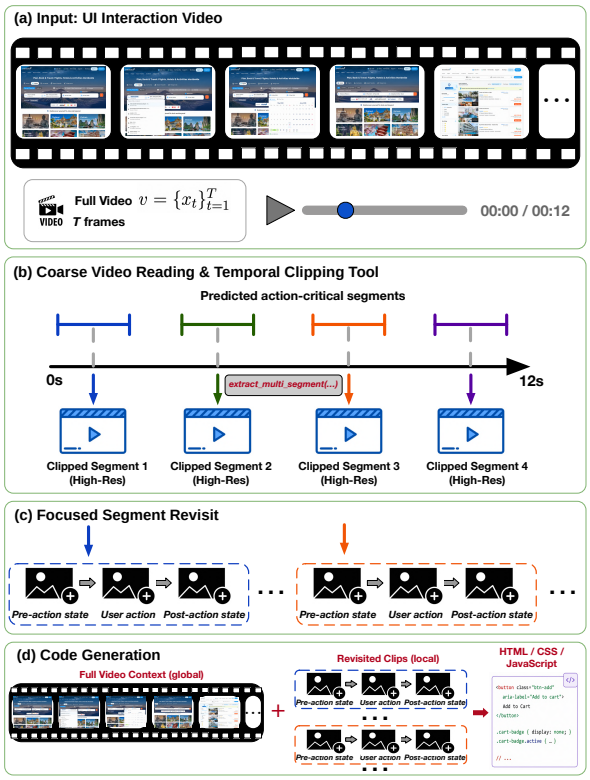


Figure 2: Overview of Video2Code. Action-Aware Revisit locates action-critical segments, revisits them with a temporal clipping tool, and generates executable webpage code from global and local evidence.

s_{i+1}. Thus, the key observation unit for UI video-to-code generation is not an isolated frame, but a complete state-action-state transition around a user operation.

2.2 Temporal Clipping Tool

To support action-aware revisit, Video2Code uses a temporal clipping tool that extracts model-specified video segments for focused observation. Given an input UI interaction video v, the model can issue a tool call

$$a_t = \texttt{extract_multi_segment}(v, \hat{\tau}_t),$$

where $\hat{\tau}_t = \{(\hat{t}_s^j, \hat{t}_e^j)\}_{j=1}^{K}$ denotes a set of temporal segments predicted by the model at step t. Each segment $(\hat{t}_s^j, \hat{t}_e^j)$ specifies a video interval that is expected to contain action-critical transition evidence. Executing the tool call returns the corresponding clipped video observations:

$$o_t = E(a_t) = \{c_1, c_2, \ldots, c_K\},$$

where each c_j is the clipped segment associated with $(\hat{t}_s^j, \hat{t}_e^j)$.

The returned clips are incorporated into the model context for subsequent reasoning:

$$\mathcal{H}_{t+1} = \mathcal{H}_t \oplus o_t,$$

where $\mathcal{H}_t$ denotes the current multimodal context and $\oplus$ denotes context augmentation. Since each clipped segment covers a much shorter temporal range than the original video, the model can inspect it at a higher temporal resolution without uniformly densifying the full video. The tool itself does not decide which regions are important; it only executes the temporal intervals specified by the model.

2.3 Action-Aware Revisit

Video2Code follows a two-round revisit process: it first builds a coarse understanding of the full interaction video to locate action-critical regions, and then revisits these regions with the temporal clipping tool to recover local transition evidence before code generation.

Coarse Video Reading. In the first round, the model observes the full UI interaction video at a coarse temporal resolution to infer the global interaction flow and produce a revisit plan. Specifically, it predicts action-critical temporal segments $\hat{\tau}$ that are likely to contain executable transition evidence around user operations such as clicks, text inputs, selections, and scrolls. The predicted segments are then passed to the temporal clipping tool for focused extraction.

Focused Segment Revisit. In the second round, the returned clips are reintroduced into the model context as focused observations. The model combines the global context from the coarse pass with the local evidence from the revisited clips to generate the final HTML/CSS/JavaScript implementation. This process preserves global interaction structure while recovering local transition evidence needed for behavior-faithful code generation.

2.4 Defining Action-Critical Segments

A key question in Action-Aware Revisit is what should be revisited. For UI video-to-code generation, an action-critical segment is not simply a visually salient frame or an informative video moment. Instead, it should provide sufficient temporal evidence for recovering an executable transition. Thus, we define an action-critical segment as a video interval that covers three elements: the UI state before a user operation, the operation itself, and the resulting UI state after the operation. This

definition ensures that the revisited segment provides the local state-action-state evidence needed for generating event handlers, state updates, and dynamic UI behavior.

Pure Action Segments. A pure action segment contains a single user operation whose effect can be understood locally, such as a click, text input, selection, or scroll. For such actions, the segment should include the pre-action state, the action execution process, and the post-action state. For example, when a click opens a dropdown menu, the segment should cover the collapsed menu, the click operation, and the expanded menu state. This allows the model to infer both the triggering event and the UI change that should be implemented in code.

Action-Scroll Segments. Some webpage behaviors cannot be recovered from isolated actions because the resulting state is revealed only after a subsequent scroll. For example, clicking a navigation link may load new content, while scrolling exposes the newly loaded section. If the click and scroll are clipped independently, the model may miss the contextual dependency between the triggering action and the visible post-action state. Therefore, we group such action-scroll sequences into a single segment when the scroll is necessary to observe the result of the preceding action.

This transition-centered definition distinguishes Video2Code from general frame or clip selection. The goal is not merely to retain visually informative content, but to preserve complete transition evidence that can be translated into executable webpage behavior.

3 Action-Aligned Supervision and Training

To train Video2Code, we construct action-aligned supervision that follows the same two-round structure as Action-Aware Revisit. Figure 3 illustrates the overall supervision and training pipeline.

3.1 UI Interaction Video Collection

We collect UI interaction videos from executable webpages through automated browser exploration. Given a webpage URL or an HTML file, a web exploration agent interacts with the page by executing common user operations such as clicks, text inputs, selections, and scrolls. The exploration is performed over webpage states to expose visual states and action-triggered transitions, while the

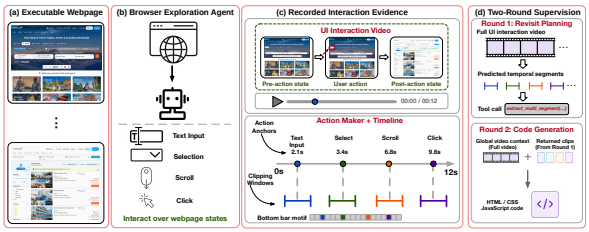


Figure 3: Action-aligned supervision for Video2Code. Interaction videos and action timelines supervise temporal clipping, while webpage implementations supervise executable code generation from the global context and returned clips.

browser viewport is recorded as a UI interaction video.

To obtain temporal anchors for user actions, we inject a lightweight visual marker into the webpage during recording. Specifically, a 12-pixel-high bar is added to the bottom of the page and changes color when an action is executed. This marker does not affect the webpage layout or interaction logic, but provides an explicit signal for localizing action boundaries. We then derive action timelines that specify when each operation occurs and which temporal windows should be considered for clipping.

For each webpage, the collected data consists of a UI interaction video, an action timeline with temporal anchors, and the corresponding webpage implementation. The video provides the input observation, the timeline provides supervision for action-aware temporal clipping, and the implementation provides the target executable code.

3.2 Constructing Action-Aligned Video-Code Supervision

Given a UI interaction video, its action-aligned timeline, and the corresponding webpage implementation, we construct supervision that follows the same two-round structure as Video2Code inference.

Round 1: Revisit Planning. The first round teaches the model to locate action-critical regions from the full interaction video. We convert the action-aligned timeline into target tool-call supervision. Specifically, each action anchor is expanded into a temporal clipping window that covers the pre-action state, the action execution process, and the post-action state. The model is trained to predict these temporal windows and invoke the temporal clipping tool accordingly. This supervision encourages the model to focus on executable transition evidence rather than treating all video moments as

equally informative.

Round 2: Code Generation after Revisit. The second round teaches the model to generate executable webpage code after the clipped segments are returned. For each example, the model receives the global video context together with the revisited clips extracted in the first round. The target output is the corresponding webpage implementation, including HTML, CSS, and JavaScript. This supervision encourages the model to ground code generation in both the overall interaction flow and the local transition evidence recovered through revisit.

Annotation and Filtering. We use Gemini-3.1-Pro to annotate the two-round supervision. For the first round, the action-aligned clipping timelines are converted into target tool-call trajectories, including the expected temporal segments and tool calls. For the second round, each clipped segment is represented as a sequence of frames and used to produce the corresponding reference webpage implementation. After annotation, we apply a function judge filtering step to remove low-quality examples. The filtering mainly checks code quality and reasoning/tool-call consistency. After filtering, we obtain *WebVidCoding*, a dataset of 12K fully annotated action-aligned video-code pairs for training Video2Code. Further details of *WebVidCoding* construction are provided in Appendix A.1.

3.3 Training

We fine-tune Video2Code on *WebVidCoding* with supervised fine-tuning, using GLM-4.6V-100B as the base model. Each training example follows the two-round format of Action-Aware Revisit: the model first predicts tool calls for action-critical temporal segments from the full video, and then generates the executable webpage implementation conditioned on the returned clips. This training format matches the inference-time revisit process. Training details are provided in Appendix A.2.

4 Experiments

In this section, we evaluate Video2Code under both visual and functional criteria. Our experiments examine whether Video2Code can preserve visual states and action-triggered behaviors from UI videos, whether Action-Aware Revisit improves over direct video-to-code generation and strong VLM baselines, and whether the gains hold across real-world, synthetic, and complex interaction scenarios.

4.1 Experimental Setup

Evaluation Models. We compare Video2Code with closed-source and open-source multimodal models that support native video input and can generate code. Closed-source models include Gemini-3.1, Gemini-2.5, and Kimi-K2.6 accessed via official APIs, while open-source models include representative Qwen3.5 (Team, 2026) and Qwen3-VL (Bai et al., 2025a). All baselines are prompted to directly generate an executable HTML/CSS/-JavaScript webpage from the input UI interaction video. We focus on code-capable VLMs rather than video-understanding-only models to avoid conflating UI video understanding with basic code generation ability.

Benchmarks. We construct *WebVideo2Code-Bench* for UI video-to-code generation, consisting of two subsets that cover real-world and controlled webpage scenarios. *WebVideo2Code-Real* contains 85 real-world webpages selected from 100 manually recorded websites. For each website, we operate the webpage in a browser, record an interaction video, annotate user-action timestamps, and filter out recordings with overly sparse, overly dense, or ambiguous interactions. *WebVideo2Code-Syn* contains 76 synthetic webpages built from trajectories collected by an exploration agent. We filter low-quality trajectories and remove redundant or repeated operations to obtain concise benchmark videos. Both subsets cover common operations such as click, text inputs, selections, and scrolls. Each webpage contains multiple interaction tasks, and we report final scores by first averaging over tasks within each webpage and then over webpages. Detailed benchmark construction procedures and statistics are provided in Appendix A.3.

Metrics. We evaluate generated webpages using two complementary metrics: *visual similarity* and *functional correctness*. Visual similarity measures whether the rendered states of the generated webpage match the corresponding target states in the input video. Functional correctness measures whether the generated webpage can reproduce the expected state changes after executing the demonstrated user actions. Both metrics are computed at the task level and then aggregated page-wise: for each webpage, we first average scores over its in-

Table 1: Main results on *WebVideo2Code-Bench*. Visual similarity and functional correctness are computed page-wise by first averaging over interaction tasks within each webpage and then averaging over webpages.

Model	WebVid2Code-Real		WebVid2Code-Syn	
	Visual Sim.	Func. Corr.	Visual Sim.	Func. Corr.
Closed-source VLM				
Gemini-3.1-Pro	82.53	71.64	86.28	58.30
Gemini-2.5-Pro	79.99	61.81	86.83	56.17
Kimi-K2.6	86.11	73.14	92.45	71.27
Open-source VLM				
Qwen3.5-122B-A10B-Thinking	78.01	43.21	87.01	50.44
Qwen3-VL-235B-A22B-Thinking	71.75	35.29	80.80	33.06
GLM-4.6V	70.94	29.85	79.53	53.59
Video2Code	**82.06**	**61.99**	**88.53**	**58.06**

teraction tasks, and then average page-level scores over all webpages. This prevents webpages with more tasks from dominating the final results.

4.2 Browser-Based Functional Verification

To evaluate whether a generated webpage reproduces the behavior shown in the input video, we perform browser-based interaction verification. Given an annotated interaction video, we first split it into task-wise frame sequences according to the user-action timestamps, where each task corresponds to one operation and its surrounding state transition. For each task, we launch the generated webpage in a browser and provide the verifier with the current webpage screenshot, DOM tree, and the target task frames. The verifier then executes the operation that best matches the demonstrated action, with an action space including click, text entry, selection, and scroll.

During execution, the browser records the resulting webpage trace as a sequence of frames. The verifier compares this execution trace with the target task frames and produces two judgments: whether the expected state change occurs, and how visually similar the resulting state is to the target state. These judgments are used to compute functional correctness and visual similarity, respectively.

For real-world webpages, tasks are executed sequentially following the order of the manually recorded interaction video. To reduce cascading failures, we use an initial-page retry mechanism: when a task cannot be executed from the current state, we reload the generated webpage and retry the same task from the initial page. If the retry succeeds, the new browser state is used for subsequent tasks; otherwise, the task is marked as failed. Further details of the evaluation protocol, task-level scoring, and page-wise aggregation are provided in Appendix A.4.

4.3 Main Results

Table 1 reports the main results on *WebVideo2Code-Bench*, including *WebVideo2Code-Real* and *WebVideo2Code-Syn*. Closed-source VLMs such as Kimi-K2.6 and Gemini-3.1-Pro achieve strong performance, showing that UI video-to-code generation benefits from powerful general-purpose multimodal reasoning. However, direct generation with open-source models remains limited, especially in functional correctness. Compared with the GLM-4.6V direct generation baseline, Video2Code improves visual similarity from 70.94 to 82.06 and substantially improves functional correctness from 29.85 to 61.99 on *WebVideo2Code-Real*. On *WebVideo2Code-Syn*, Video2Code also improves over GLM-4.6V from 79.53 to 88.53 in visual similarity and from 53.59 to 58.06 in functional correctness. These results suggest that Action-Aware Revisit and action-aligned supervision help recover executable state transitions across both real-world and controlled webpage scenarios, substantially strengthening the underlying open-source model for behavior-faithful UI video-to-code generation.

4.4 Ablation Study

We conduct an ablation study on *WebVideo2Code-Real* to isolate the effect of Action-Aware Revisit in real-world webpage scenarios. The *w/o Revisit* variant uses the same fine-tuned model as Video2Code, but disables the temporal clipping tool and directly generates code from the coarse video observation. All other settings are kept unchanged. As shown in Table 2, removing Action-Aware Revisit leads

Table 2: Ablation study of Action-Aware Revisit on *WebVideo2Code-Real*. The *w/o Revisit* variant disables the temporal clipping tool.

Model	*WebVideo2Code-Real*	
	Visual Sim.	Func. Corr.
Video2Code w/o Revisit	77.17	47.84
Video2Code	**82.06**	**61.99**

to clear performance drops. Video2Code improves visual similarity from 77.17 to 82.06, and more substantially improves functional correctness from 47.84 to 61.99. The larger gain in functional correctness suggests that focused revisit is particularly important for recovering executable transition evidence around user actions.

4.5 Effect of Frame-Based Observation

We further examine whether sparse frame-based observation is sufficient for UI video-to-code generation. For models without native video input, we convert each input video into sampled frames and provide them as multi-image context. Specifically, we evaluate GPT-5.5 with 1 FPS sampling, which provides relatively dense visual observations without native video processing. As shown in Table 3, this setting achieves high visual similarity, but its functional correctness remains lower than Video2Code. This suggests that sampled frames can provide strong cues for webpage appearance, but may still miss action-critical temporal evidence needed for executable behavior.

4.6 Further Analysis

We further analyze Video2Code under different interaction conditions on *WebVideo2Code-Real*, including interaction count, interaction-type diversity, and video length.

Effect of Interaction Count. Figure 4a groups samples by the number of user interactions in each video. The results show that Video2Code performs similarly on videos with 1–8 and 9–16 interactions. Visual similarity remains almost unchanged, while functional correctness shows only a moderate decrease from 62.9 to 59.6. This suggests that Video2Code remains relatively stable as the number of interaction steps increases within the evaluated range.

Effect of Interaction Type Diversity. We measure the diversity of interaction types using Shannon entropy over clicks, text inputs, selections, and scrolls. For each video, let $\mathcal{A} = \{\text{click}, \text{input}, \text{select}, \text{scroll}\}$ denote the set of interaction types, and let p_a be the proportion of interactions of type a in the video. The interaction-type entropy is defined as:

$$H(\mathcal{A}) = -\sum_{a \in \mathcal{A}} p_a \log_2 p_a, \tag{1}$$

where interaction types with $p_a = 0$ are ignored in the summation. A higher entropy indicates that the video contains a more diverse mixture of interaction types, while a lower entropy indicates that the interaction process is dominated by one or a few action types. As shown in Figure 4b, visual similarity remains stable across entropy groups, while functional correctness decreases from 68.9 in the low-entropy group to around 58 in higher-entropy groups. This indicates that mixed-action videos are more challenging for executable behavior recovery.

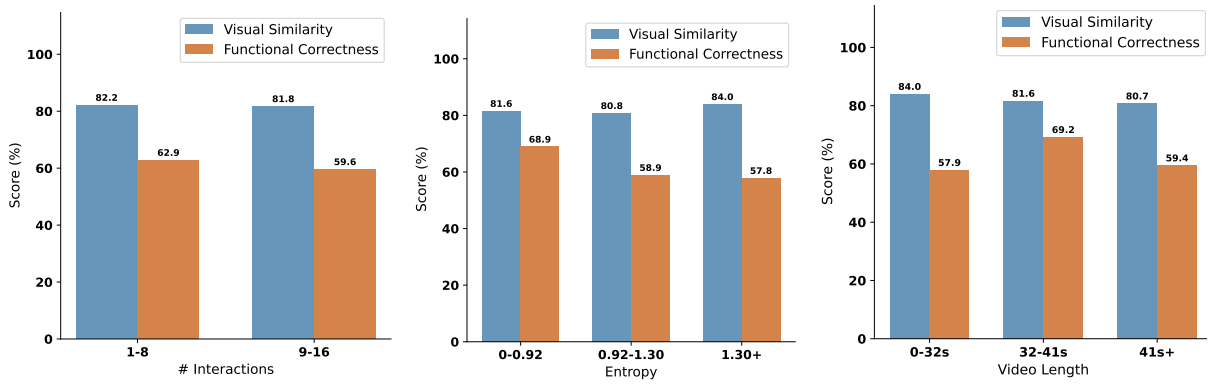
Effect of Video Length. Figure 4c shows that visual similarity slightly decreases as video length increases, while functional correctness does not show a monotonic drop and is highest in the medium-length group. Overall, Video2Code maintains relatively stable performance across different video lengths, although longer videos may introduce additional temporal grounding challenges.

Table 3: Effect of frame-based observation on *WebVideo2Code-Real*. For GPT-5.5, we sample frames at 1 FPS and provide them as multi-image context.

Model	*WebVideo2Code-Real*	
	Visual Sim.	Func. Corr.
GPT-5.5 with 1 FPS Frames	86.99	60.23
Video2Code	**82.06**	**61.99**

4.7 Human Evaluation of Automatic Verification

To assess the reliability of our browser-based automatic verification, we compare verifier outputs with human judgments on sampled interaction tasks from *WebVideo2Code-Real*. For functional correctness, human annotators inspect the target video segment and generated webpage execution trace to judge whether the demonstrated action is successfully reproduced. The verifier achieves a 96.0% agreement rate with human judgments. For visual similarity, we compute score differences only on cases where both human and automatic judgments mark the action as successful; cases where either side marks the action as failed are skipped. Under



(a) Interaction count (b) Interaction type (c) Video length

Figure 4: Further analysis of Video2Code on *WebVideo2Code-Real* under different interaction conditions, including interaction count, interaction-type entropy, and video length.

this protocol, 70.2% of valid cases have an absolute visual-score difference smaller than 10 points, indicating reasonable alignment between automatic and human visual assessment.

5 Related Work

Video Understanding with Vision-Language Models. Recent vision-language models have extended multimodal understanding from images to videos, enabling models to process temporal visual inputs and answer questions about dynamic scenes (Cheng et al., 2024; Chen et al., 2024; Li et al., 2024a; Zhang et al., 2024, 2025; Bai et al., 2025b). To handle limited context budgets, video-capable VLMs often rely on sparse frame sampling or compressed visual representations (Cheng et al., 2024; Chen et al., 2024; Zhang et al., 2024, 2025; Li et al., 2024b; Weng et al., 2024; Shen et al., 2024). Other methods further introduce hierarchical processing or adaptive frame/clip selection to preserve informative temporal regions (Cheng et al., 2025; Hannan et al., 2025; Shu et al., 2025; Tang et al., 2025; Tan et al., 2026). These strategies are effective for many natural video understanding tasks, where the goal is often to capture high-level semantic content over time. In contrast, UI interaction videos contain temporally sparse but functionally decisive events: a short click, selection, or scroll may determine the executable behavior that must be implemented in code. Thus, Video2Code focuses not on general video understanding, but on recovering action-conditioned UI state transitions for executable webpage generation.

Visual UI-to-Code Generation. Generating front-end code from visual inputs has attracted increasing attention, from early screenshot-to-code systems such as pix2code (Beltramelli, 2018) to recent VLM-based benchmarks and datasets, including WebSight (Laurençon et al., 2024), Vision2UI (Gui et al., 2024), Design2Code (Si et al., 2024), Web2Code (Yun et al., 2024), and WebCode2M (Gui et al., 2025a). Most existing methods focus on reconstructing webpage layout, style, and visual appearance from static screenshots or design images, with recent improvements from decomposition, hierarchical generation, or modular agentic pipelines (Wan et al., 2024; Gui et al., 2025b; Jiang et al., 2025). Beyond one-shot static reconstruction, UI2Code^{N} formulates UI-to-code as iterative visual optimization (Yang et al., 2025), while WebVIA studies interactive and verifiable UI-to-code generation from multi-state interaction evidence (Xu et al., 2025). Video2Code complements these efforts by using UI interaction videos as input, where user actions and state transitions provide temporal evidence for preserving both visual appearance and executable behavior.

6 Conclusion

In this paper, we propose **Video2Code**, an action-aware approach for generating executable webpages from UI interaction videos. We formulate UI video-to-code generation as executable state-transition recovery and introduce *Action-Aware Revisit*, which identifies action-critical regions from coarse video observation and revisits them at higher temporal resolution. With action-aligned video-code supervision, Video2Code learns to generate code from both global video context and local transition evidence. Experiments under visual and functional evaluation show that Video2Code improves behavior-faithful UI video-to-code generation over direct video observation.

Limitations

Video2Code focuses on generating executable webpages from UI interaction videos, but its current scope does not cover the full diversity of modern web applications. While *WebVideo2Code-Bench* includes both real-world and synthetic webpages, it mainly focuses on publicly accessible pages and does not fully cover scenarios involving authentication, personalization, backend-dependent content, or highly asynchronous behavior. Our browser-based functional verification focuses on common webpage operations, including clicks, text entry, selection, and scroll. Although these operations cover many frequent UI behaviors, they do not yet include more complex interactions such as drag-and-drop, file upload, canvas manipulation, or long-horizon workflows involving external services. Expanding the benchmark and verifier to support richer interaction primitives would enable a more comprehensive evaluation of behavior-faithful UI video-to-code generation.

Ethical Statement

This work studies UI video-to-code generation for browser-interactable webpages. We recognize potential concerns related to privacy, copyright, and misuse, especially when real-world webpages are recorded or when generated code is used to imitate existing interfaces. Video2Code is intended as a research framework for studying behavior-faithful webpage generation, rather than for cloning deployed services or collecting private user data.

For data construction and evaluation, we use controlled synthetic webpages and publicly accessible real-world webpages. We do not use authentication-required, paywalled, or private pages. Real-world recordings are limited to common and reversible interactions such as clicks, text entry, selections, and scrolls, and examples with unstable or ambiguous behavior are filtered. We do not intentionally collect personal information, credentials, or private user-generated content.

We use AI-assisted annotation during supervision construction, and all generated supervision is filtered before training. Human annotation is limited to action timestamp labeling and verification reliability assessment, without collecting sensitive annotator information. Released artifacts will include documentation describing intended research use, construction procedures, and limitations.

References

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, and 1 others. 2025a. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*.

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, and 1 others. 2025b. Qwen2. 5-vl technical report. *arXiv preprint arXiv:2502.13923*.

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

Lin Chen, Xilin Wei, Jinsong Li, Xiaoyi Dong, Pan Zhang, Yuhang Zang, Zehui Chen, Haodong Duan, Bin Lin, Zhenyu Tang, and 1 others. 2024. Sharegpt4video: Improving video understanding and generation with better captions. *Advances in Neural Information Processing Systems*, 37:19472–19495.

Chuanqi Cheng, Jian Guan, Wei Wu, and Rui Yan. 2025. Scaling video-language models to 10k frames via hierarchical differential distillation. *arXiv preprint arXiv:2504.02438*.

Zesen Cheng, Sicong Leng, Hang Zhang, Yifei Xin, Xin Li, Guanzheng Chen, Yongxin Zhu, Wenqi Zhang, Ziyang Luo, Deli Zhao, and 1 others. 2024. Videollama 2: Advancing spatial-temporal modeling and audio understanding in video-llms. *arXiv preprint arXiv:2406.07476*.

Yi Gui, Zhen Li, Yao Wan, Yemin Shi, Hongyu Zhang, Bohua Chen, Yi Su, Dongping Chen, Siyuan Wu, Xing Zhou, and 1 others. 2025a. Webcode2m: A real-world dataset for code generation from webpage designs. In *Proceedings of the ACM on Web Conference 2025*, pages 1834–1845.

Yi Gui, Zhen Li, Yao Wan, Yemin Shi, Hongyu Zhang, Yi Su, Shaoling Dong, Xing Zhou, and Wenbin Jiang. 2024. Vision2ui: A real-world dataset with layout for code generation from ui designs. *arXiv preprint arXiv:2404.06369*, 5.

Yi Gui, Yao Wan, Zhen Li, Zhongyi Zhang, Dongping Chen, Hongyu Zhang, Yi Su, Bohua Chen, Xing Zhou, Wenbin Jiang, and 1 others. 2025b. Uicopilot: Automating ui synthesis via hierarchical code generation from webpage designs. In *Proceedings of the ACM on Web Conference 2025*, pages 1846–1855.

Tanveer Hannan, Md Mohaiminul Islam, Jindong Gu, Thomas Seidl, and Gedas Bertasius. 2025. Revision-llm: Recursive vision-language model for temporal grounding in hour-long videos. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 19012–19022.

Wenyi Hong, Wenmeng Yu, Xiaotao Gu, Guo Wang, Guobing Gan, Haomiao Tang, Jiale Cheng, Ji Qi,

Junhui Ji, Lihang Pan, and 1 others. 2025. Glm-4.1 v-thinking: Towards versatile multimodal reasoning with scalable reinforcement learning. *arXiv e-prints*, pages arXiv–2507.

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

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

Bo Li, Yuanhan Zhang, Dong Guo, Renrui Zhang, Feng Li, Hao Zhang, Kaichen Zhang, Peiyuan Zhang, Yanwei Li, Ziwei Liu, and 1 others. 2024a. Llava-onevision: Easy visual task transfer. *arXiv preprint arXiv:2408.03326*.

Yanwei Li, Chengyao Wang, and Jiaya Jia. 2024b. Llama-vid: An image is worth 2 tokens in large language models. In *European Conference on Computer Vision*, pages 323–340. Springer.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. 2023. Visual instruction tuning. *Advances in neural information processing systems*, 36:34892–34916.

Shuming Liu, Chen Zhao, Tianqi Xu, and Bernard Ghanem. 2025. Bolt: Boost large vision-language model without training for long-form video understanding. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 3318–3327.

Xiaoqian Shen, Yunyang Xiong, Changsheng Zhao, Lemeng Wu, Jun Chen, Chenchen Zhu, Zechun Liu, Fanyi Xiao, Balakrishnan Varadarajan, Florian Bordes, and 1 others. 2024. Longvu: Spatiotemporal adaptive compression for long video-language understanding. *arXiv preprint arXiv:2410.17434*.

Yan Shu, Zheng Liu, Peitian Zhang, Minghao Qin, Junjie Zhou, Zhengyang Liang, Tiejun Huang, and Bo Zhao. 2025. Video-xl: Extra-long vision language model for hour-scale video understanding. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 26160–26169.

Chenglei Si, Yanzhe Zhang, Ryan Li, Zhengyuan Yang, Ruibo Liu, and Diyi Yang. 2024. Design2code: Benchmarking multimodal code generation for automated front-end engineering. *arXiv preprint arXiv:2403.03163*.

Wenhui Tan, Ruihua Song, Jiaze Li, Jianzhong Ju, and Zhenbo Luo. 2026. Think-clip-sample: Slow-fast frame selection for video understanding. *arXiv preprint arXiv:2601.11359*.

Xi Tang, Jihao Qiu, Lingxi Xie, Yunjie Tian, Jianbin Jiao, and Qixiang Ye. 2025. Adaptive keyframe sampling for long video understanding. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 29118–29128.

Qwen Team. 2026. Qwen3. 5: Towards native multimodal agents. *URL https://qwen. ai/blog*.

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

Peng Wang, Shuai Bai, Sinan Tan, Shijie Wang, Zhihao Fan, Jinze Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, and 1 others. 2024a. Qwen2-vl: Enhancing vision-language model's perception of the world at any resolution. *arXiv preprint arXiv:2409.12191*.

Weihan Wang, Qingsong Lv, Wenmeng Yu, Wenyi Hong, Ji Qi, Yan Wang, Junhui Ji, Zhuoyi Yang, Lei Zhao, Song XiXuan, and 1 others. 2024b. Cogvlm: Visual expert for pretrained language models. *Advances in Neural Information Processing Systems*, 37:121475–121499.

Yuetian Weng, Mingfei Han, Haoyu He, Xiaojun Chang, and Bohan Zhuang. 2024. Longvlm: Efficient long video understanding via large language models. In *European Conference on Computer Vision*, pages 453–470. Springer.

Mingde Xu, Zhen Yang, Wenyi Hong, Lihang Pan, Xinyue Fan, Yan Wang, Xiaotao Gu, Bin Xu, and Jie Tang. 2025. Webvia: A web-based vision-language agentic framework for interactive and verifiable ui-to-code generation. *arXiv preprint arXiv:2511.06251*.

Zhen Yang, Wenyi Hong, Mingde Xu, Xinyue Fan, Weihan Wang, Jiele Cheng, Xiaotao Gu, and Jie Tang. 2025. Ui2codeˆ n: A visual language model for test-time scalable interactive ui-to-code generation. *arXiv preprint arXiv:2511.08195*.

Sukmin Yun, Haokun Lin, Rusiru Thushara, and 1 others. 2024. Web2code: A large-scale webpage-to-code dataset and evaluation framework for multimodal llms. *arXiv preprint arXiv:2406.20098*.

Boqiang Zhang, Kehan Li, Zesen Cheng, Zhiqiang Hu, Yuqian Yuan, Guanzheng Chen, Sicong Leng, Yuming Jiang, Hang Zhang, Xin Li, and 1 others. 2025. Videollama 3: Frontier multimodal foundation models for image and video understanding. *arXiv preprint arXiv:2501.13106*.

Yuanhan Zhang, Jinming Wu, Wei Li, Bo Li, Zejun Ma, Ziwei Liu, and Chunyuan Li. 2024. Llava-video: Video instruction tuning with synthetic data. *arXiv preprint arXiv:2410.02713*.

Zheyu Zhang, Ziqi Pang, Shixing Chen, Xiang Hao, Vimal Bhat, and Yu-Xiong Wang. 2026. One token per highly selective frame: Towards extreme compression for long video understanding. *Advances in Neural Information Processing Systems*, 38:57332–57358.

Jinguo Zhu, Weiyun Wang, Zhe Chen, Zhaoyang Liu, Shenglong Ye, Lixin Gu, Hao Tian, Yuchen Duan, Weijie Su, Jie Shao, and 1 others. 2025. Internvl3: Exploring advanced training and test-time recipes for open-source multimodal models. *arXiv preprint arXiv:2504.10479*.

A Appendix

A.1 Data Construction

We provide additional implementation details for constructing *WebVidCoding*, the action-aligned video-code dataset used to train Video2Code. Specifically, we describe browser-based webpage exploration, action timeline extraction, clipping window construction, and two-round video-code supervision generation.

Exploration protocol. We use an automated browser exploration agent to collect UI interaction videos from synthetic HTML documents and real-world URLs. Given a webpage URL or an HTML file, the agent first identifies candidate interactive elements from the rendered page and DOM tree, and then explores the page by executing common operations, including clicks, text entry, selections, and scrolls. The exploration is performed over webpage states rather than isolated elements: after each operation, the browser waits for the page to stabilize and records the resulting viewport state. We remove repeated operations that lead to visually identical or near-identical states, so that the collected trajectory focuses on distinct action-triggered transitions. During exploration, the browser viewport is recorded as a video, and the executed action sequence is stored together with action metadata.

Action-boundary localization. To localize user actions in the recorded video, we inject a lightweight visual marker during recording. The marker is a 12-pixel-high bar fixed at the bottom of the viewport and changes color when an action is executed. After recording, we detect color changes in the marker region and use them as temporal anchors for user operations. Each anchor is associated with the corresponding action type, such as click, text entry, selection, or scroll.

Action-critical clipping. Given an action anchor, we construct a clipping window by expanding the anchor with pre-action and post-action temporal margins. The pre-action margin preserves the UI state before the operation, while the post-action margin captures the resulting state after the page response. For actions whose effects are only visible after a subsequent scroll, we merge the triggering action and the required scroll into a single action-critical segment, so that the clipped segment preserves complete transition evidence.

Two-round supervision generation. We construct the two-round supervision data through an offline teacher-generation pipeline. For each training sample, we first use the action-aligned timeline to derive action-critical clipping windows, and sample the corresponding clips into frame sequences for teacher annotation. These frames are provided to Gemini-3.1-Pro to generate supervision targets for the two-round Action-Aware Revisit format. In the first round, the target output is a structured tool-call trajectory that invokes `extract_multi_segment` with the action-critical temporal windows. This teaches the model to produce revisit plans that are consistent with the inference-time clipping interface. In the second round, the returned clips are appended to the model context, and the target output consists of an interaction-aware observation and reasoning process enclosed by `<think></think>`, followed by the executable webpage implementation in HTML, CSS, and JavaScript. To improve supervision quality, we generate the second-round target with two separate teacher requests: one for visual observation and interaction reasoning, and the other for executable code generation. This separation reduces interference between high-level transition analysis and low-level code synthesis, producing cleaner supervision for both revisit-based reasoning and final webpage reconstruction.

Quality filtering. We apply filtering to remove examples with unreliable supervision. We discard examples with invalid tool-call format, invalid thinking format or incomplete codes. We also filter out examples where the target implementation fails to render into a valid webpage. The remaining examples form *WebVidCoding*, a 12K-example dataset consisting of 4K synthetic webpage examples and 8K real-world webpage examples.

A.2 Training Details

We provide implementation details of the supervised fine-tuning procedure used to train Video2Code.

Training format. Each training instance is constructed from an interaction video v, an action-aligned clipping timeline τ, the returned clips $E(v, \tau)$, and the corresponding executable webpage implementation y. The first round takes the full interaction video as input and supervises the model to predict a structured temporal clipping call:

$$\texttt{extract_multi_segment}(v, \tau).$$

Table 4: Training configuration of Video2Code.

Configuration	Value
Base model	GLM-4.6V-100B
Training dataset	*WebVidCoding*
Number of examples	12K
Training objective	Supervised fine-tuning
Input format	Full video + returned clips
Tool-call target	`extract_multi_segment`
Code target	HTML/CSS/JavaScript
Frames per clip	3 fps
Learning rate	2×10^{-5}
Batch size	32
Epochs	3
Hardware	H800 GPUs

The target segments in τ are derived from the action timeline and are expected to cover action-critical state transitions. The second round appends the returned clips to the model context and supervises the model to generate the final HTML/CSS/JavaScript implementation.

Tool-call target format. The tool-call target contains a list of temporal intervals, each represented by its start and end timestamps. Each interval is constructed to cover the pre-action state, the action execution process, and the post-action state. For action-scroll cases, a single interval may cover both the triggering action and the subsequent scroll when both are needed to observe the resulting state.

Code-generation target. The code-generation target is a self-contained webpage implementation. The output is required to include the HTML structure, CSS styling, and JavaScript logic needed to reproduce the visual states and action-triggered behaviors demonstrated in the video. This format ensures that the model learns to generate executable code from both global video context and local transition evidence recovered through revisit.

Training configuration. We fine-tune Video2Code for 3 epochs with a learning rate of 2×10^{-5} and a batch size of 32 on H800 GPUs. Table 4 summarizes the training configuration.

A.3 Benchmark Construction

We provide additional details for constructing *WebVideo2Code-Bench*. The benchmark is designed to evaluate whether a generated webpage can reproduce both the visual states and the action-triggered behaviors demonstrated in a UI interaction video. It contains two subsets, *WebVideo2Code-Real* and *WebVideo2Code-Syn*, covering real-world webpages and controlled synthetic webpages respectively.

WebVideo2Code-Real. We construct the real-world subset from manually recorded webpage interaction videos. We start from 100 publicly accessible websites covering diverse real-world webpage scenarios, including *education platforms*, *academic, legal, and database-style websites*, *health information and symptom-checking pages*, *finance calculators and report interfaces*, *shopping and e-commerce flows*, *news search*, *online tools such as translation and file conversion*, *job search platforms*, and *travel booking or itinerary search pages*. For each website, we operate the webpage in a browser and record a UI interaction video that contains common user operations, including clicks, text entry, selections, and scrolls. We manually annotate the timestamps of user actions and use them to segment the video into task-wise state transitions. We filter out recordings with overly sparse interactions, overly dense interactions, unstable page behavior, ambiguous action effects, or interaction processes that cannot be reliably verified. After filtering, *WebVideo2Code-Real* contains 85 real-world webpage examples.

WebVideo2Code-Syn. We construct the synthetic subset from executable webpages explored by the browser exploration agent. The agent records interaction trajectories during webpage exploration, and we further convert these trajectories into concise benchmark videos. Specifically, we remove repeated operations, redundant states, and transitions whose effects are visually ambiguous or difficult to verify. Compared with the real-world subset, the synthetic subset provides more controlled interaction structures and executable references, which allows us to evaluate UI video-to-code generation under cleaner webpage behavior. After filtering, *WebVideo2Code-Syn* contains 76 synthetic webpage examples.

Task segmentation. Each benchmark video is decomposed into multiple interaction tasks according to the annotated action timestamps. A task corresponds to one user operation and its surrounding state transition. For each task, we keep frames covering the pre-action state, the action execution process, and the post-action state. These task-wise

frame sequences are used by the browser-based verifier to select the corresponding operation on the generated webpage and judge whether the resulting execution trace matches the target transition.

Filtering criteria. We apply filtering to ensure that each benchmark example is clear and evaluable. Examples are removed if the webpage fails to load, the recording is incomplete, the interaction effect is not visually observable, the action timestamp is ambiguous, or the task requires unsupported operations beyond the verifier action space. For synthetic webpages, we additionally remove trajectories with repeated or redundant operations that do not introduce new UI states. This filtering process ensures that the benchmark focuses on observable and executable webpage behavior.

A.4 Evaluation Protocol and Metrics

We provide additional details of the browser-based evaluation protocol used in *WebVideo2Code-Bench*. Given a UI interaction video and a generated webpage implementation, the goal is to evaluate whether the generated webpage can reproduce both the visual states and the action-triggered behaviors demonstrated in the video.

Task-wise video segmentation. We first split each input interaction video into task-wise frame sequences according to the annotated user-action timestamps. Each task corresponds to one user operation and its surrounding state transition. For example, an input video can be decomposed into multiple tasks:

$$v \rightarrow \{T_1, T_2, \ldots, T_n\},$$

where each task T_i contains frames covering the pre-action state, the user operation, and the post-action state. These task-wise frames serve as the target reference for both action execution and result verification.

Generated webpage rendering. For each generated implementation, we launch a browser environment and render the generated webpage. Before executing each task, we record the current webpage screenshot and extract the DOM tree. The screenshot provides visual context, while the DOM tree provides structural information about interactive elements on the generated page.

Action selection and execution. For each task, the verifier receives the current webpage screenshot, the DOM tree, and the target task frames from the input video. It then selects the operation that best matches the demonstrated user action and executes it on the generated webpage. The supported action space includes click, text entry, selection, and scroll. During execution, the browser records the resulting webpage trace as a sequence of frames.

Task-level scoring. After executing each task, the verifier compares the recorded webpage trace with the target task frames and produces two task-level scores. (1) *Task-level functional correctness.* Functional correctness evaluates whether the generated webpage produces the expected state change after the demonstrated user action is executed. For each task, the verifier checks whether the operation can be successfully performed and whether the resulting webpage state matches the transition shown in the target video segment. The task receives a binary functional score of 1 if the expected state change occurs, and 0 otherwise. If the operation cannot be executed, the webpage fails to respond, or the resulting state is inconsistent with the target transition, the task is counted as functionally incorrect. After all tasks of a webpage have been verified, the page-level functional score is reported on a 100-point scale by multiplying the ratio of functionally correct tasks among all verified tasks by 100. In other words, it is computed as the number of tasks with a functional score of 1 divided by the total number of tasks for that webpage, and then multiplied by 100. (2) *Task-level visual similarity.* Visual similarity evaluates whether the resulting webpage state after execution visually matches the target post-action state. For each functionally correct task, the verifier compares the recorded execution trace of the generated webpage with the target task frames and assigns a visual similarity score on a 100-point scale. Failed functional executions are excluded from page-level visual similarity, since the generated webpage does not reach the intended post-action state and comparing it with the target state would not reflect the quality of the intended visual reconstruction. The page-level visual similarity is therefore computed as the average task-level visual similarity over all functionally correct tasks.

To ensure that visual verification also covers the initial page state, we introduce a separate initial-page comparison mechanism. For long webpages, this state is naturally covered by the initial scroll task: we first perform a scroll action to reveal the full initial page beyond the viewport, thereby ver-

ifying both the page's scrollability and its visual similarity to the original initial-page view. However, for smaller webpages whose initial page is fully contained within the viewport, no scroll task is needed. In such cases, the first task may directly evaluate the post-action state after a click, text entry, or selection, leaving the initial page without a corresponding visual similarity score. Therefore, we conduct an additional visual comparison between the rendered initial page and the first frame of the webpage video, and include the resulting score in page-level similarity.

Sequential execution. For *WebVideo2Code-Real*, tasks are executed sequentially following the order of the recorded interaction video. After one task is executed, the browser state is preserved and used as the starting state for the next task. This setting matches the real recording process, where user operations are performed continuously on the same webpage session. For *WebVideo2Code-Syn*, we follow the predefined task structure of the synthetic trajectory. When a task is annotated as requiring a fresh initial state, we reload the generated webpage before executing the next task; otherwise, we continue from the current browser state. This handles synthetic trajectories where some operations are designed to be evaluated independently from the initial page.

Initial-page retry for *WebVideo2Code-Real*. To reduce cascading failures, we use an initial-page retry mechanism. When a task cannot be executed from the current generated webpage state, we reload the generated webpage and retry the same task from the initial page. If the retry succeeds, the new browser state is used for subsequent tasks; otherwise, the task is marked as functionally incorrect. This retry mechanism reduces cascading failures, where one failed operation would otherwise cause later independently executable tasks to be skipped.

Page-wise aggregation. Both metrics are first computed at the task level and then aggregated at the page level. For a webpage p with task set $\mathcal{T}_p$, we compute

$$\mathrm{Vis}(p) = \frac{1}{|\mathcal{T}_p|} \sum_{t \in \mathcal{T}_p} v_{p,t},$$
$$\mathrm{Func}(p) = \frac{|\{t \in \mathcal{T}_p \mid f_{p,t} = 1\}|}{|\mathcal{T}_p|} \times 100.$$

where $v_{p,t}$ denotes the visual similarity score for task t, and $f_{p,t} \in \{0, 1\}$ denotes whether task t is functionally correct. Thus, $\mathrm{Func}(p)$ measures the percentage of successfully reproduced tasks on webpage p.

The final benchmark score is obtained by averaging webpage-level scores over all webpages:

$$\mathrm{Vis} = \frac{1}{|\mathcal{P}|} \sum_{p \in \mathcal{P}} \mathrm{Vis}(p),$$
$$\mathrm{Func} = \frac{1}{|\mathcal{P}|} \sum_{p \in \mathcal{P}} \mathrm{Func}(p).$$

This page-wise aggregation prevents webpages with more interaction tasks from dominating the overall result.

A.5 Prompts

We provide the prompt templates used in our data construction, model inference, and browser-based evaluation. These prompts cover the main stages of Video2Code, including synthetic webpage construction, temporal clipping tool-call generation, video-to-code supervision, baseline inference, action replay, and interaction verification.

Synthetic HTML document construction prompts. We use synthetic webpages in both data construction and evaluation. Each synthetic HTML document is constructed through a two-stage process. First, we prompt a teacher model to generate a webpage design instruction based on a given theme. Second, we provide the generated instruction to a code-generation prompt to produce an executable HTML document. This process increases the content and interaction diversity of the synthetic webpages. The complete prompt templates are shown in Figure 5 and Figure 6.

Tool-call segment generation prompt. During data construction, we construct the first round supervision by a given UI interaction video frames and its action timeline. The prompt asks the annotator/model to identify temporal regions that contain complete state-action-state evidence based on timeline and output structured calls to `extract_multi_segment`, where each argument specifies an action-critical temporal interval to revisit. The complete prompt is shown in Figure 7.

Video-to-code generation prompt. During the second-round supervision, we use two prompts to separate reasoning from implementation. The first prompt requires the model to analyze the extracted video clips and frames and produce its reconstruction reasoning enclosed within `<think>`

and `</think>` tags. The second prompt then asks the model to generate the final executable webpage code in HTML, CSS, and JavaScript. The generation prompt emphasizes both visual fidelity and interactive functionality, including event handlers, state updates, input binding, selections, and scroll-dependent behavior. The complete prompts are shown in Figure 8 and Figure 9.

Video-based baseline inference prompt. This prompt is used for closed-source and open-source VLM baselines that support native video input. Unlike Video2Code, these baselines are prompted to directly generate executable webpage code from the full UI interaction video, without tool-based temporal revisit. The complete prompt is shown in Figure 10.

Frame-based baseline inference prompt. This prompt is used for baselines evaluated under frame-based observation. We convert the input video into sampled frames and provide them as multi-image context. The model is then prompted to directly generate executable HTML/CSS/JavaScript code from these frames, using the same task objective as the video-based setting. The complete prompt is shown in Figure 11.

Interaction selection prompt for evaluation. During browser-based evaluation, each task is evaluated by selecting and executing the operation that best matches the demonstrated user action. This prompt asks the model to choose the appropriate action based on the target video frames, the current rendered webpage screenshot, and the page DOM tree. The complete prompt template is shown in Figure 12.

Interaction verification prompts for evaluation. We use two interaction verification prompts. The first prompt evaluates the generated webpage after the selected action is executed. It judges whether the webpage produces the expected state change, and assigns a visual similarity score for the resulting state against the target post-action state. The second prompt is used for initial-page visual verification, where the rendered initial page is compared with the first frame of the input video when the initial page is not otherwise covered by a task-level evaluation. The complete prompts are shown in Figure 13 and Figure 14.

Synthetic HTML Design Instruction Template

Please write a detailed prompt that will be used to instruct a text-to-image model to generate an interactive webpage HTML code with the theme **"<INSERT THEME FROM THEME LIST>"**.

Specific Requirements:

1. The webpage content should revolve around the specified theme and include a wide variety of theme-related modules.
2. The webpage must contain multiple interactive elements, limited to `buttons`, `input fields`, and `dropdown selectors`. Each interactive component should cause corresponding and reasonable changes on the webpage.
3. The webpage content should be rich, detailed, and contextually diverse.
4. The output should only contain the final prompt for the AI to generate the webpage—without explanations, metadata, or additional commentary.

Figure 5: Template used to construct webpage design prompts for generating interactive webpage HTML code.

Table 5: Error distribution on manually inspected failed cases from *WebVideo2Code-Real*.

Error Type	Ratio
Event binding & logic failures	64.1%
Structural & systemic errors	24.1%
HTML render errors	5.9%
Visual & layout anomalies	5.0%
Unknown	0.9%

A.6 Error Analysis

We provide a qualitative error analysis to better understand the remaining challenges in UI video-to-code generation. We manually inspect 220 failed cases from *WebVideo2Code-Real* and categorize them into five error types, as summarized in Table 5. The dominant failure mode is *event binding and interaction logic errors*, which account for 64.1% of the inspected failures. This suggests that the main bottleneck of UI video-to-code generation is not only reconstructing webpage appearance, but also correctly implementing executable behavior from action-conditioned state transitions.

Event binding and logic failures. The most common failure occurs when the generated webpage contains visually plausible interactive components but fails to implement the correct behavior.

Synthetic HTML Code Generation Prompt

You are a web development expert highly sensitive to details and interaction experience, proficient in React and Tailwind CSS. Please generate a highly interactive single-page application with reasonable layout and rich content for the specified theme according to the following requirements.

Basic Requirements:

1. Generate a complete interactive single-page website rendered using **React (v18)** and **Tailwind CSS (v3+)**.
2. Return only the full source code wrapped within `<html>...</html>` tags. **Do not** include markdown wrappers, explanations, or code comments.
3. Must include the following dependencies:

```
<script src="https://cdn.jsdelivr.net/npm/react@18.0.0/umd/react.
    development.js"></script>
<script src="https://cdn.jsdelivr.net/npm/react-dom@18.0.0/umd/react-dom.
    development.js"></script>
<script src="https://cdn.jsdelivr.net/npm/@babel/standalone/babel.js"></
    script>
<script src="https://cdn.tailwindcss.com"></script>
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-
    awesome/5.15.3/css/all.min.css"></link>
```

Interactivity and Functional Areas:

1. All interactive components (`input`, `button`, `select`) must trigger meaningful updates to the rendered page.
2. For editable content, use modals, dropdowns, or input forms with complete validation.
3. Use real pictures from `https://picsum.photos/`. Each image must have a fixed URL and remain constant across reloads.

Page Structure and Layout:

1. Include logical partitions (navigation, sidebar, main content, etc.) referencing modern app layouts.
2. Ensure all sections are populated; empty placeholders are not allowed.
3. The visual style must match the assigned theme (e.g., business, minimalism, tech, lifestyle).

Notes:

- Do not output explanations or text outside the code.
- Ensure all theme-related UI logic is complete and intuitive.

Webpage Description: `<INSERT DETAILED PROMPT FROM STAGE 1>`

Figure 6: Code Generation Prompt used for large-scale HTML synthesis.

For example, a button, tab, dropdown, or selection widget may be present, but clicking or selecting it does not trigger the expected state change, or updates the wrong region of the page. These errors indicate that behavior-faithful UI video-to-code generation requires more than identifying interactive elements: the model must also bind each demonstrated action to the correct event handler, state update, and post-action UI state.

Structural and systemic errors. Some failures arise from incorrect or incomplete webpage structure. In these cases, the generated implementation may miss important components, organize sections in the wrong hierarchy, or fail to preserve the global relationship among UI states. Such structural errors often affect multiple interactions at once, because later actions depend on page elements or states that were not correctly reconstructed.

HTML render errors. A smaller portion of failures are caused by invalid or unstable generated code. The generated HTML/CSS/JavaScript may fail to render, produce a blank page, or break during browser execution. These cases directly prevent browser-based interaction verification and

highlight the need for stronger executable-code constraints during generation.

Visual and layout anomalies. Some generated webpages are executable but visually inconsistent with the target video. Typical issues include incorrect spacing, misaligned components, missing visual details, inconsistent typography, or wrong component placement. Although these errors may not always prevent interaction execution, they reduce visual similarity and can make the generated post-action state difficult to match with the target transition.

Unknown failures. A small portion of failures, 0.9%, are categorized as unknown. These cases are difficult to attribute to a single cause, either because multiple error sources occur simultaneously or because the recorded execution trace does not provide enough evidence to identify the primary failure mode.

A.7 Demo Cases

We present qualitative examples of UI video-to-code generation with Video2Code. For each example, we visualize the action-critical segment revisited by Video2Code and the rendered states of the generated webpage before and after executing the corresponding user action. These examples cover representative interaction patterns, including click, text input, scroll, and selection, and illustrate how Video2Code uses focused temporal revisit to recover local state-action-state evidence and translate it into executable webpage behavior. Figures 15–18 show representative cases for these action types, demonstrating behaviors such as click-triggered updates, input-dependent changes, scroll-dependent state transitions, and selection-based content updates.

Figures 19–21 show scroll-based demo cases. Unlike click or selection interactions, scroll cases are evaluated by comparing the target video frames with the corresponding full-screen rendered states after scrolling. These examples show that Video2Code can preserve long-page layout structure and recover scroll-dependent viewport transitions.

Tool-Call Segment Generation Prompt for Data Construction

You are a video understanding expert proficient in user interface (UI) interaction analysis, and you also have a deep background in frontend engineering.

Task Background
I will provide you with a **URL of a long video file**, as well as a set of **short video screenshots (image sequences) that I have prepared for you**. These screenshots are key frames from the moments when the user performs key interaction operations, such as clicking, typing, selecting, and scrolling, in the long video, covering the initial state before the operation, during the operation, and the page changes after the operation. The user will also give you an approximate description of what operation was performed in the long video for each group of operations. This video URL: **<VIDEO_URL>**

Core Task
You need to map the operation list I send you to these short video screenshots in your mind, and then **fabricate a thinking process in which you are watching the complete long video**. In this process, you should use the visual page content, that is, the screenshots you refer to, to **describe in what time period each interaction occurred, what exactly was done, and what feedback the page produced**. Finally, you need to call a tool to batch-cut these segments.

Output Structure Specification
Please strictly follow the following requirements in your output, and do not include any greetings:

Output Requirement 1: Long Video Observation and Fabricated Thinking Process `<think></think>`
This is the key point. Please complete the above "pretending to independently discover and observe" process inside the `<think></think>` tags, pretending that you are studying the interaction time points in the webpage, and describe them in detail. You must strictly follow the following rules:

1. **Pretend to discover independently**: Do not mention words such as "reference operation list," "user-provided list," or "which image" in the thinking process. Your tone must sound like you are watching the complete video frame by frame and then independently discovering these operations.

   Incorrect example: "According to reference operation 1, I see..."

   Correct example: "While browsing the beginning of the video, I notice that around 4.5 seconds, the mouse cursor moves to the 'Login' button, and then the page jumps..."

2. **Describe the operation and visual feedback in detail**: For each time segment you mention, you may slightly modify the times based on the times in the following **user timeline and operation reference description** to make it look more like an independent discovery. You must precisely describe what happens on the screen during that period. This includes:

   - What specific button or link did the mouse click?
   - What text was entered into the input field?
   - How did the page respond to this, such as popup, color change, or page jump?
   - Use the visual information from these short video screenshots, such as which one is before, which one is after, or the operation process, to enrich your description.

   For example: "At 4.5 seconds, I see an initial page layout, then the mouse moves and clicks, and at 5.0 seconds the page finishes loading."

3. **Special treatment**: For operations marked as **scroll browsing**, there is no need to describe the specific operation details; you only need to describe that scrolling shows the overall structure of the entire page.

Output Requirement 2: Tool Call Statement and Tool Call `<tool></tool>`
After completing the thinking, briefly state in natural language that you have identified the key interaction segments and will call the tool to cut and extract them.
Example: After analysis, I have independently identified the key interaction operations in the video. I will call the `zai_extract_multi_video_segments` tool to precisely cut and extract the following key interaction segments: from 3.5 seconds to 5.5 seconds, from 11.2 seconds to 13.2 seconds.

Output Requirement 3: Tool Call `<tool></tool>`
Finally, inside the `<tool></tool>` tags, output JSON data that strictly conforms to the following format.
Note: The tool accepts two main parameters: `video_url`, which is the URL I pass to you, and `segments`, which is a list of objects containing `start` and `end`. The `video_url` part must be written as the exact same URL I pass to you.
JSON format example:
{ "video_url": "**<VIDEO_URL>**", "segments": [{ "start": 3.5, "end": 5.5 }, { "start": 11.2, "end": 13.2 }] }

Timeline information
<TIMELINE AND OPERATION INFORMATION>

Please strictly follow the above output structure specification and do not output any irrelevant information.

Figure 7: Prompt for generating tool-call segments from interaction videos.

Video-to-Code Thinking Reconstruction Prompt for Data Construction

You are a senior engineer proficient in frontend development (React + Tailwind).

Task Objective
Based on the **key short videos capturing the exact moments when interactions occur** that you observed in the previous round from a long webpage interaction video, you need to construct a **visualized video observation process**, while also **pretending that you are about to generate HTML code to replicate this interactive webpage**.

Core Task
Your task is to "reverse-render" this group of **operation lists** in your mind into a **visualized video observation process**.

<VIDEO FRAMES INFORMATION>

Output Structure Specification
Please strictly follow the requirements below when outputting, and do not include any irrelevant greetings or extra text.

Output Requirement: Interaction Logic Observation Flow `<think></think>`
This is the key point! Please complete the thinking process of observing each short video inside the `<think></think>` tags, pretending that you are studying the interaction points on the webpage and trying to replicate them.
You need to strictly follow the rules below:

1. **First observe the overall webpage layout**: Before watching the first interaction short video, first inspect the webpage layout in the first video. This is the initial layout of the webpage, and you should describe it in detail.

2. **Observe each operation**:

   - *Observe the mouse position*: For each interaction short video, please observe the **changes in the mouse position**, because it must be the mouse position interacting with some interactive element that causes the page to change.
   - *Confirm the clicked object*: By observing the feedback after the click, clearly confirm in your thinking exactly which element the mouse clicked.
   - *Construct the interaction narrative*: Combined with properties such as "scrolling after the operation" and "continuous interaction" in the prompt, describe the page state changes caused by the operation, for example: loading spinner, page navigation, adding a new row of data to a table, scrolling to reveal content, etc.

2.5 **Special treatment for scroll browsing**: When the operation property indicates that this operation is a **scroll browsing** operation, there is no need to describe the above details. You only need to piece together the screenshots in the scroll browsing process and describe the overall structure of the entire page.

3. **Summarize and pretend that you are about to write code**: At the end, summarize what functions you have observed that this webpage needs to have, and state that you are starting to write the code. You do not actually need to write it.

Important Constraints

- At this stage, do not mention any code terms, such as `div`, `state`, `onclick`, etc.; only describe the visuals and logic.
- You are not looking at images; you are watching videos. Do not output anything inside `<think></think>` suggesting that you are "pretending to watch videos" or reading "Figure 1". Instead, fully fabricate it: only say that you are watching these clipped videos yourself.
- Do not mention any video timestamps; only describe the video content.
- Do not mention the reference operation information sent by the user. Instead, disguise it as something you observed from the videos yourself. That is, do not directly output "Operation 1" or similar reference content provided by the user in the result.

Figure 8: Prompt used to guide the model to produce an interaction logic thinking flow from clipped webpage interaction video frames before code generation.

Video-to-Code Coding Reconstruction Prompt for Data Construction

You are highly skilled at building interactive web pages with React and Tailwind, and can precisely restore a complete HTML interactive webpage based on multiple webpage screenshots provided by the user.

Interface Requirements

1. Build the initial page according to the first webpage screenshot provided by the user, and it must be completely consistent with the content of the first screenshot given to you.
2. Do not omit any details, including background colors, fonts, font sizes, spacing, borders, icons, text, etc., all of which must strictly match the screenshots.
3. Every sentence of text in the screenshots must be presented exactly as it is.
4. For image content, please use real images from the `https://picsum.photos/` library, with URLs similar to `https://picsum.photos/id/.../.../...`. Each image must explicitly list its URL; do not use reusable image components. The image URL of each webpage component must be fixed, and do not use random numbers to regenerate it each time.

Core Task Requirements
Please bring out a "microscope"-level power of observation to achieve a 1:1 pixel-level reproduction of the webpage interactions:

1. **Observe the overall page changes**: Carefully observe the timeline screenshot sequence provided below. Focus on comparing the **initial screen before the operation** and the **final state/instant-change state**, and do not omit any details on the page, such as a popup notification appearing in the upper-right corner, a newly added list item, etc. For screenshot sequences marked as **pure scroll browsing**, treat them as structured puzzle pieces for obtaining the global layout and hidden content of the page, ensuring the completeness of the entire page.
2. **Precisely locate the source of interaction**: In the **initial screen before the operation** screenshot, find the mouse position, infer which component was clicked or typed into, and ensure that these components exist in the reproduced webpage and possess the same capabilities as the original. Make its functionality and visual feedback exactly the same as in the screenshots.
3. All interactive operations given to you must be perfectly reproduced in the generated HTML, meaning they must have complete functionality, and after completion the page must be consistent with the corresponding screenshot.

Please use the following libraries:

- `React 18`: `https://cdn.jsdelivr.net/npm/react@18.0.0/umd/react.development.js`
- `ReactDOM 18`: `https://cdn.jsdelivr.net/npm/react-dom@18.0.0/umd/react-dom.development.js`
- `Babel`: `https://cdn.jsdelivr.net/npm/@babel/standalone/babel.js`
- `Tailwind CSS`: `https://cdn.tailwindcss.com`
- `Font Awesome`: `https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.15.3/css/all.min.css`

Code Output Format

1. Only output the code inside the complete `<html></html>` tags.
2. Ensure the code is complete HTML webpage code that can be rendered directly, and do not omit anything with ellipses.
3. Do not add markdown, `html`, or any additional text before or after the code.

Video frames information
<VIDEO FRAMES INFORMATION>.

Figure 9: Prompt used to reconstruct webpage HTML code from clipped webpage interaction video frames

Video-based Webpage Reconstruction Prompt for Baseline Inference

You are highly skilled in building interactive web pages using React and Tailwind, and you can precisely reconstruct complete, interactive HTML web pages based on videos provided by the user.

UI Requirements

1. Do not omit any details. Background colors, fonts, font sizes, spacing, borders, icons, text, etc., must strictly match the video.
2. Every single line of text in the video must be presented exactly as it is.
3. For image content, please use real images from the `https://picsum.photos/` library, with URLs similar to `https://picsum.photos/id/.../.../...`. Each image must explicitly list its URL; do not use reusable image components. The image URL for each web component must be fixed and not randomly regenerated every time.

Core Task Requirements

Please apply a "microscopic" level of observation to achieve a 1:1 pixel-perfect reproduction of the web page interactions:

1. **Observe overall page changes**: Carefully watch the interactive web page video sent by the user, and do not miss any details on the page, such as pop-up notifications appearing in the top right corner, newly added list items, etc.
2. All interactive operations shown to you must be perfectly replicated in the generated HTML. This means it must be fully functional, and upon completion, the page must exactly match the content in the video.

Code Output Format

1. Only output the complete code within the `<html></html>` tags.
2. Ensure the code is a complete, directly renderable HTML webpage. Do not omit anything using ellipses.
3. Do not add markdown backticks, `html`, or any additional text before or after the code.

Figure 10: Prompt used to reconstruct interactive webpage HTML code directly from videos.

Frame-based Webpage Reconstruction Prompt for Baseline Inference

You are highly skilled in building interactive web pages using React and Tailwind, and you can precisely reconstruct complete, interactive HTML web pages based on frames extracted from videos provided by the user.

UI Requirements

1. Do not omit any details. Background colors, fonts, font sizes, spacing, borders, icons, text, etc., must strictly match the extracted frames.
2. Every single line of text in the extracted frames must be presented exactly as it is.
3. For image content, please use real images from the `https://picsum.photos/` library, with URLs similar to `https://picsum.photos/id/.../.../`. Each image must explicitly list its URL; do not use reusable image components. The image URL for each web component must be fixed and not randomly regenerated every time.

Core Task Requirements

Please apply a "microscopic" level of observation to achieve a 1:1 pixel-perfect reproduction of the web page interactions.
Carefully watch the chronologically ordered frames extracted from the interactive web page video sent by the user, and do not miss any details on the page, such as pop-up notifications appearing in the top right corner, newly added list items, and other subtle visual or content changes. All interactive operations shown to you must be perfectly replicated in the generated HTML. This means the webpage must be fully functional, and upon completion, the page must exactly match the content in the extracted frames.

Code Output Format

1. Only output the complete code within the `<html></html>` tags.
2. Ensure the code is a complete, directly renderable HTML webpage. Do not omit anything using ellipses.
3. Do not add markdown backticks, `html`, or any text before or after the code.

Figure 11: Prompt used to reconstruct interactive webpage HTML code directly from videos frames.

Interaction Selection Prompt for Evaluation

You are a webpage interaction replay assistant. I will give you a set of video frame screenshots arranged in chronological order, capturing a user interaction on a webpage. I will also provide a DOM Tree and current screenshot of a replicated webpage.

DOM Tree Key Fields

Each interactable element contains: id (identifier used in action instructions), tag, attrs, visible_text, address (`"zone-[x,y,w,h]"`), can_interact, input_value, options, etc.

Your Task

1. Compare the frames to identify the interaction that occurred in the original webpage.
2. The interaction is usually exactly one of: **click**, **enter** (typing), **select** (dropdown), or **scroll**.
3. Locate the corresponding element in the DOM Tree by matching visible_text, address, tag, input_value, options, and surrounding context.
4. Output the action instruction needed to replay the interaction on the replicated webpage.

Important Notes on Number of Actions

- In almost all cases, the interaction can and should be reproduced with **one single action**.
- Only output **two actions** in the special case where the video shows a dropdown/select-like interaction, but the replicated DOM Tree does not contain a usable `select` element or matching selectable option.
- In that special case, you may reproduce the interaction with two `click` actions, for example: first click to open the dropdown, then click the target option.
- Do **not** output multiple actions for ordinary clicks, typing, scrolling, or standard select elements.
- Prefer a single `select[element_id][option_text]` whenever a proper select/dropdown option exists in the DOM Tree.

Important Notes on Missing Operations

- If the interaction shown in the video cannot be matched to any actionable element in either the current screenshot or the DOM Tree, output `\boxed{{None}}`.
- Use `None` only when no reasonable click, enter, select, or scroll action can reproduce the observed interaction on the replicated webpage.

Important Notes on Scroll

- If the interaction is a scroll, it is **very likely a scroll to the bottom of the page**—use `scroll[]` with no element id.
- Only use `scroll[element_id]` if the frames clearly show scrolling to a specific element rather than to the page bottom.

Output Format

Wrap the final action instruction in LaTeX `\boxed{{}}`.
Allowed formats:

- `click[element_id]`
- `enter[element_id][input_content]`
- `select[element_id][option_text]`
- `scroll[element_id]` — scroll to bring a specific element into view
- `scroll[]` — scroll all the way to the bottom of the page
- `click[element_id_1]; click[element_id_2]` — only for the special dropdown/select replacement case described above
- `None` — only when the interaction cannot be matched to the screenshot or DOM Tree

Briefly explain your reasoning first, then give the final answer. The final answer must contain exactly one `\boxed{{}}`. If two clicks are needed, put both click actions inside the same `\boxed{{}}`. If no valid operation can be found, output exactly `\boxed{{None}}`.

DOM Tree
<DOM_TREE>

Input Visual Observations

The following are the video frame sequence and replicated webpage screenshot:
<VIDEO_FRAME_SEQUENCE_AND_REPLICATED_WEBPAGE_SCREENSHOT>

Figure 12: Interaction replay prompt used to infer executable action instructions from video frames, DOM trees, and replicated webpage screenshots.

Interaction Verification Prompt for Evaluation

You are evaluating whether a web interaction was correctly replicated by comparing **page visual similarity**.
The two groups of frames below each capture a complete interaction process including before, during, and after the action.

- Group 1 (Expected): Original webpage recording
- Group 2 (Actual): Replicated webpage after attempting the same interaction

The operation being performed is: **<ACTION INFORMATION>**

Step 1: Identify the Operation Type
Determine whether the operation is:

- **Click / Enter / Select**: An action that changes a specific UI element's state
- **Scroll**: An action whose purpose is to reveal page content

Step 2: Visual Score
Output an integer from 0–100, or -1 if incomparable.

For Click, Enter, or Select operations:

- Compare the **final post-action page state** between Group 1 and Group 2.
- Ignore any differences in the initial page state before the action; only the result matters.
- Score how visually similar the two pages look **after** the operation completes.
- Output -1 if the two post-action pages are so fundamentally different in layout or content that no meaningful visual comparison is possible, such as completely different pages being loaded or drastically different structures.

For Scroll operations:

- Scrolling's sole purpose is to make content visible. Do not evaluate whether a scroll motion occurred.
- Compare the **total visible content** across all frames in each group.
- If Group 1 required scrolling to reveal content, but Group 2 already shows all that content without scrolling, this is a **successful match**. Score based on content similarity between what is visible in both groups.
- Focus on whether the same page content is accessible or visible, not on the scrolling gesture itself.

Special Case — Pre-executed State:
If the action was already in the "done" state before it was attempted, such as a checkbox already checked, a tab already active, or a value already set, the operation had no visible effect in Group 2. In this case:

- The functional score will be "failed" because no change occurred.
- The **visual score should still reflect the similarity of the page states normally**. The pages can still look identical even without an observable change.

Scoring criteria:

- 90–100: Post-action pages look visually identical or nearly identical.
- 70–89: Post-action pages are very similar with only minor visual differences.
- 50–69: Post-action pages are similar but with noticeable differences.
- 20–49: Post-action pages differ significantly.
- 0–19: Post-action pages look completely different or an error state appeared.
- -1: The two post-action pages are fundamentally incomparable, such as different pages or broken layouts.

Step 3: Functional Score
Determine whether the operation was actually executed in Group 2, and briefly explain the reason:

- `success`: The operation clearly took effect, or the target state was already achieved before the action.
- `failed`: The operation did not take effect or produced an unexpected error state.

When making this judgment, explicitly mention the key visual evidence, such as:

- whether the expected UI state change appeared;
- whether the target element changed as intended;
- whether the page navigated, updated, expanded, selected, or scrolled as expected;
- whether the action produced no visible effect or an error state.

Output Format
Briefly explain your reasoning first, **ensuring that the justification of the functional score is included**, then provide both scores on separate lines:

- Visual score: `\score{N}`, where N is an integer from 0 to 100, or `\score{-1}` if the pages are incomparable.
- Functional score: `\passed{success}` or `\passed{failed}`.

Figure 13: Prompt used to evaluate whether a replicated webpage correctly preserves the visual and functional effects of an interaction.

Initial State Visual Similarity Prompt for Evaluation

You are evaluating the visual similarity between two webpage screenshots.
- Image 1: The first frame from the original webpage recording (reference)
- Image 2: A screenshot of the replicated/rendered webpage's initial state

Your Task
Compare these two webpage screenshots and score their visual similarity.

Focus on:
- Overall layout and structure
- Color scheme and visual style
- Content placement and hierarchy
- Key UI elements, including headers, navigation, main content area, and hero sections

Scoring criteria:
- 90–100: The pages look visually identical or nearly identical.
- 70–89: Very similar with only minor visual differences, such as slight color variation or minor spacing differences.
- 50–69: Similar but with noticeable differences.
- 20–49: Differ significantly in layout or content.
- 0–19: Look completely different.

Output Format
Briefly explain your reasoning, then provide the score on a separate line:
- Visual similarity score: `\score{N}`, where N is an integer from 0 to 100.

Figure 14: Prompt used to evaluate the initial-state visual similarity between the original webpage recording and the replicated webpage.

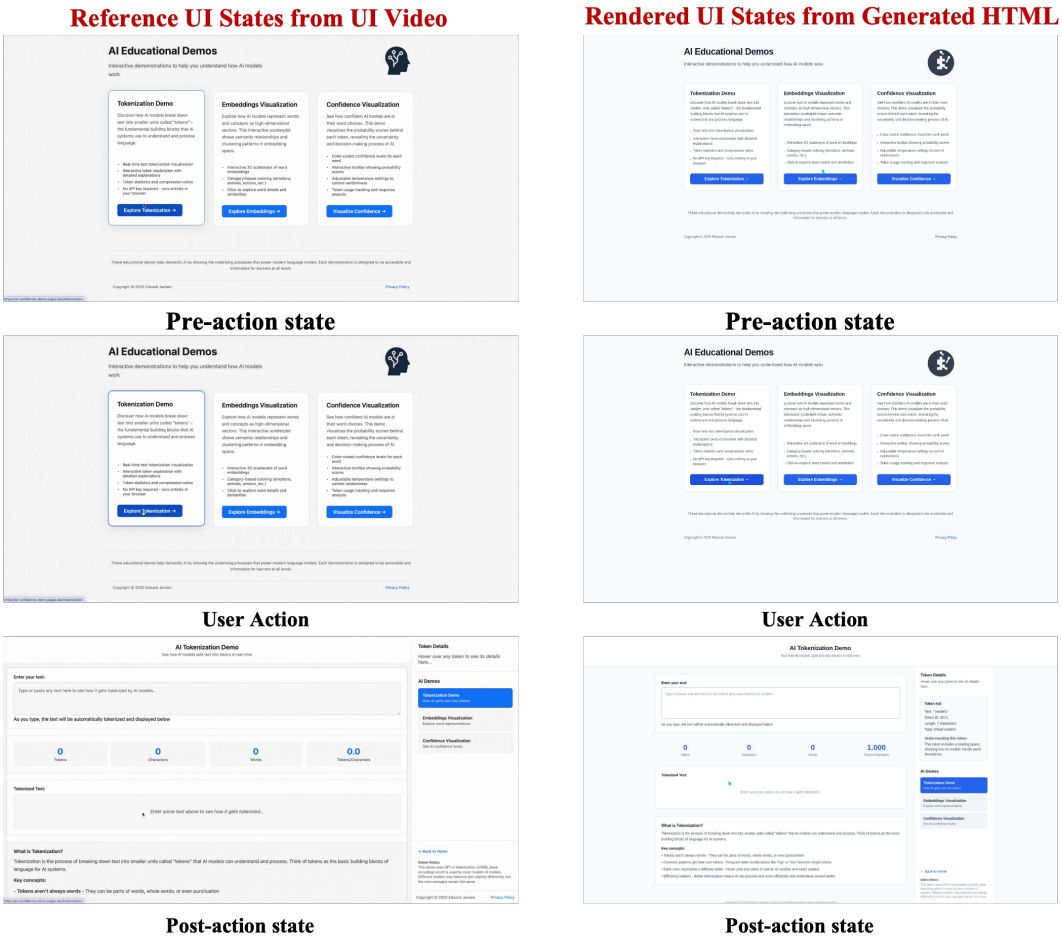


Figure 15: Demo cases of Video2Code on click interactions.

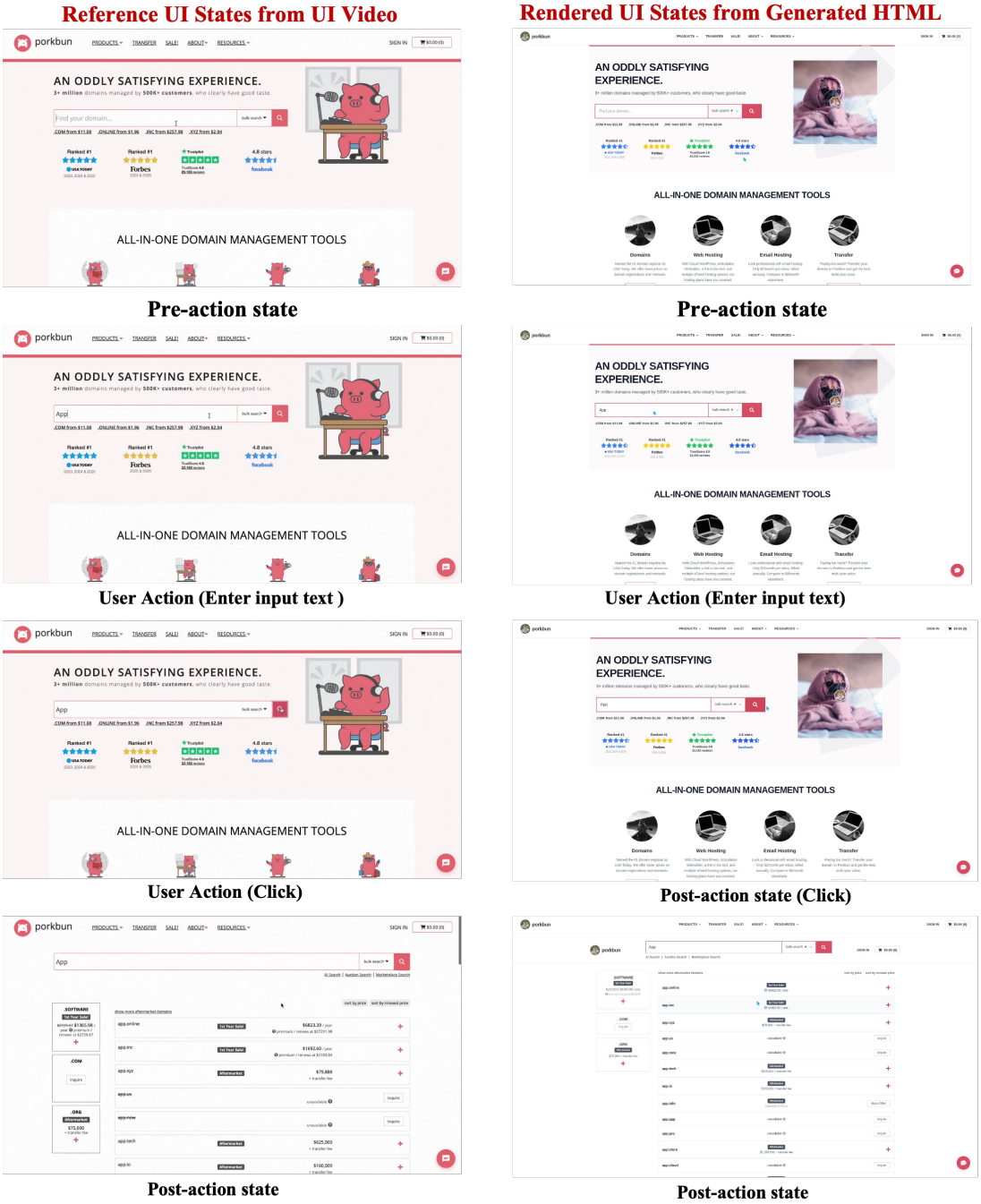


Figure 16: Demo cases of Video2Code on a text-input and click interaction sequence.

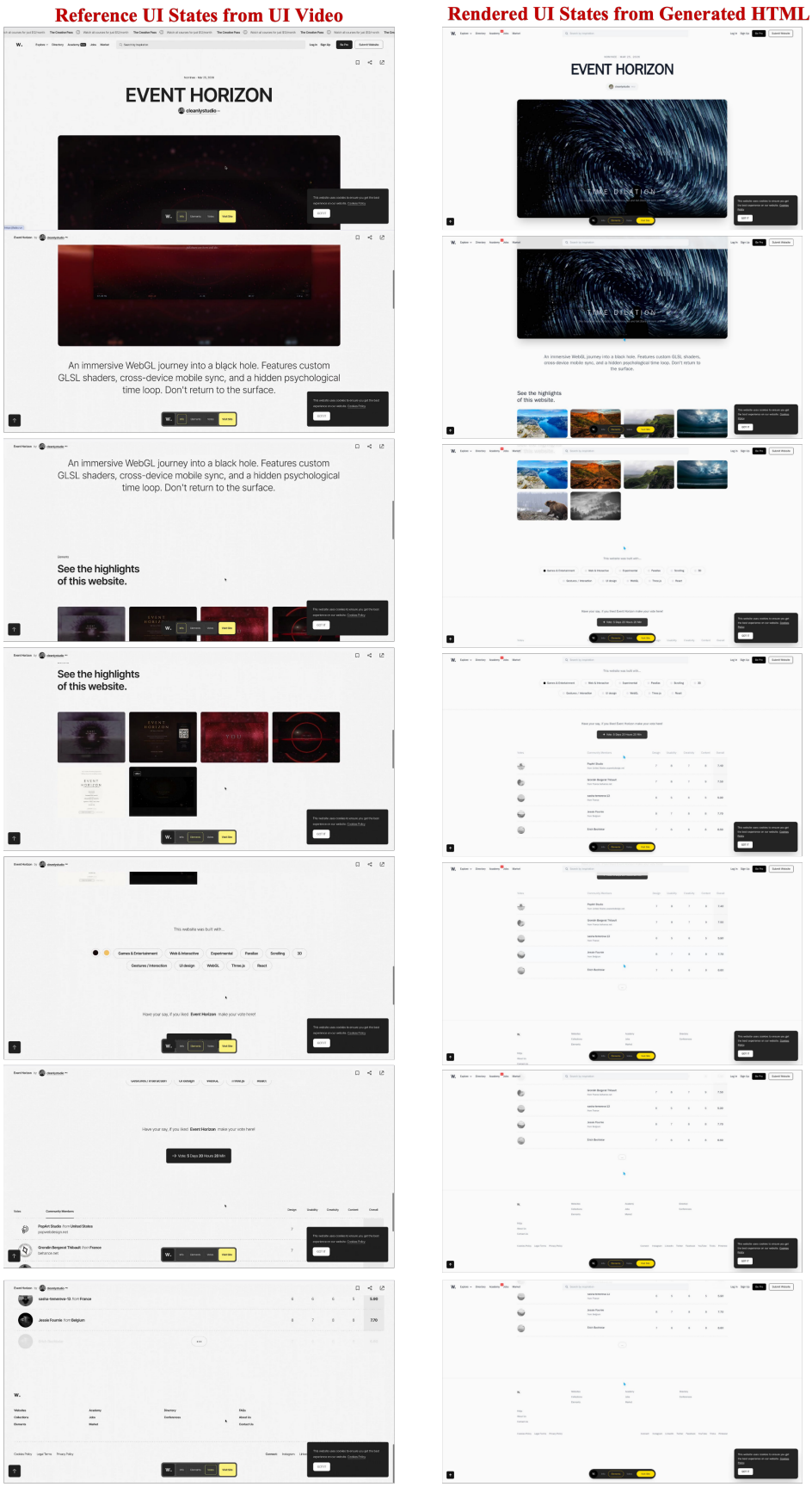


Figure 17: Demo cases of Video2Code on scroll interactions.

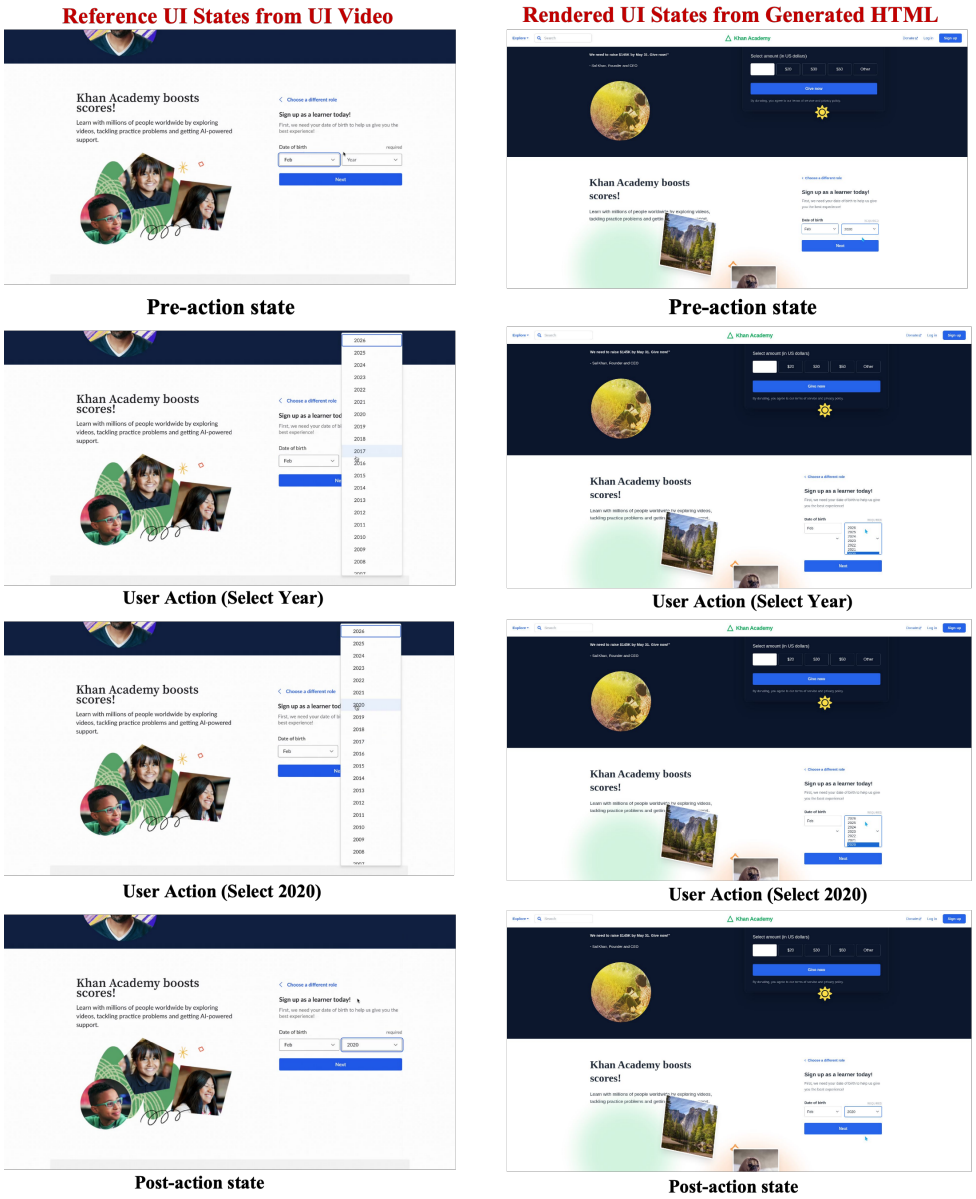


Figure 18: Demo cases of Video2Code on selection interactions.

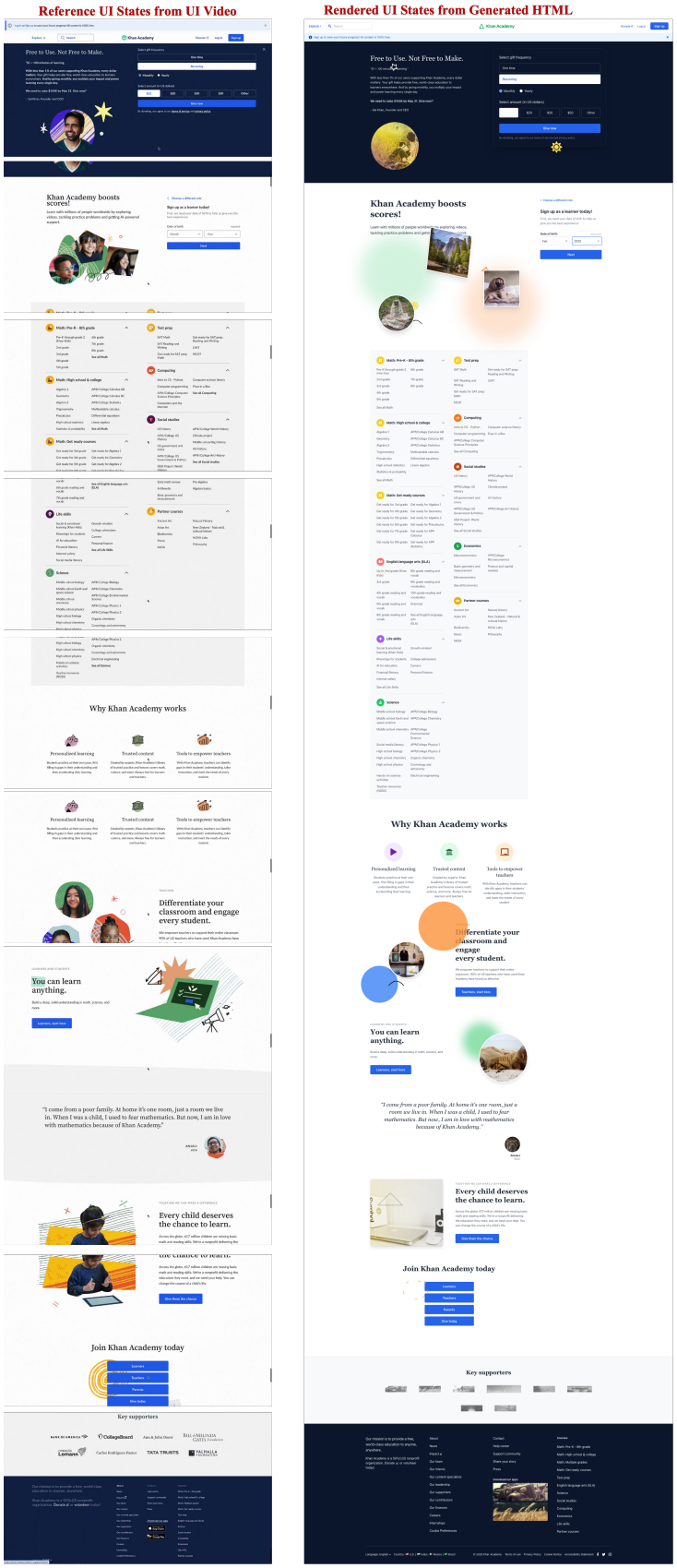


Figure 19: Demo case of Video2Code on a scroll interaction. The figure compares the target frames from the input video with the corresponding full-page rendered states of the generated webpage, showing that Video2Code recovers the scroll-dependent content transition.

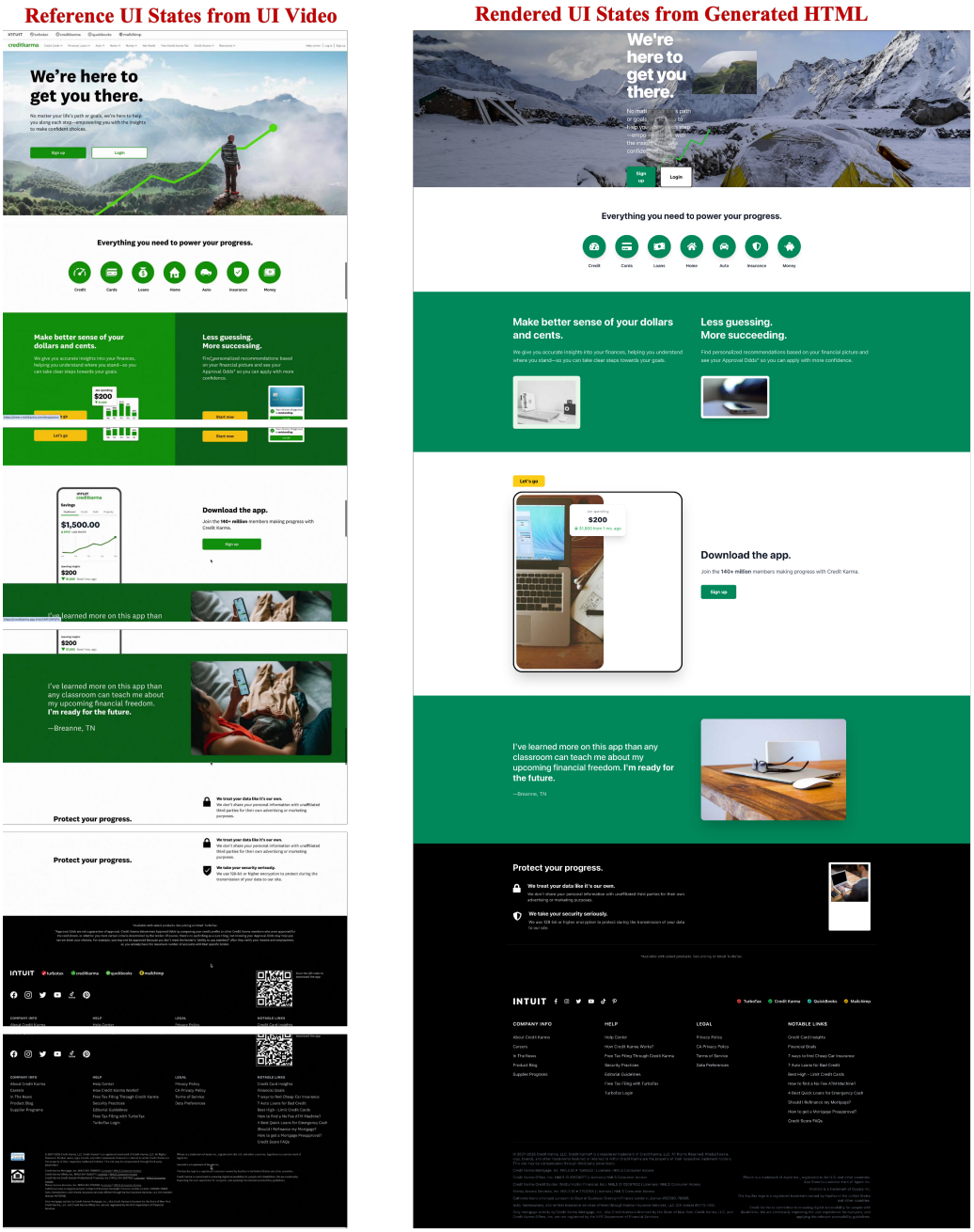
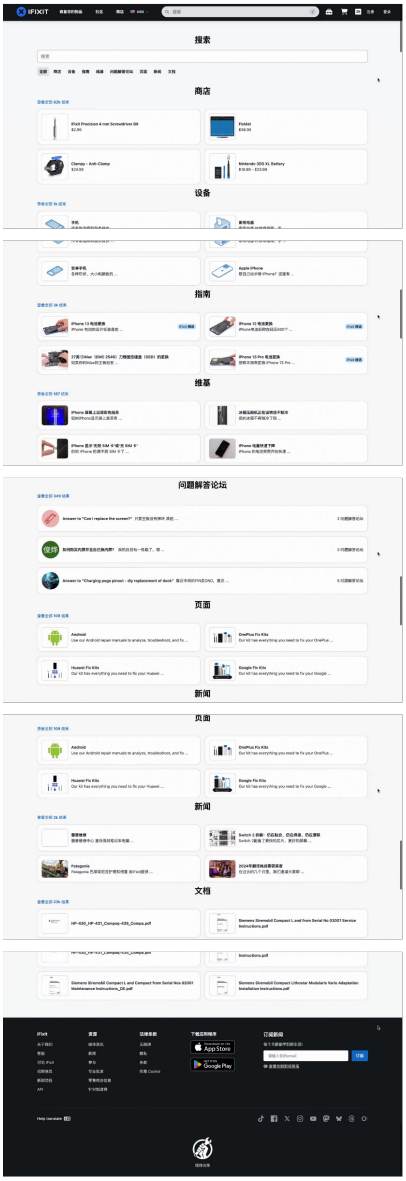


Figure 20: Demo case of Video2Code on a scroll interaction. The generated webpage preserves the long-page structure and reaches the target viewport state after executing the demonstrated scroll operation.

Reference UI States from UI Video



Rendered UI States from Generated HTML

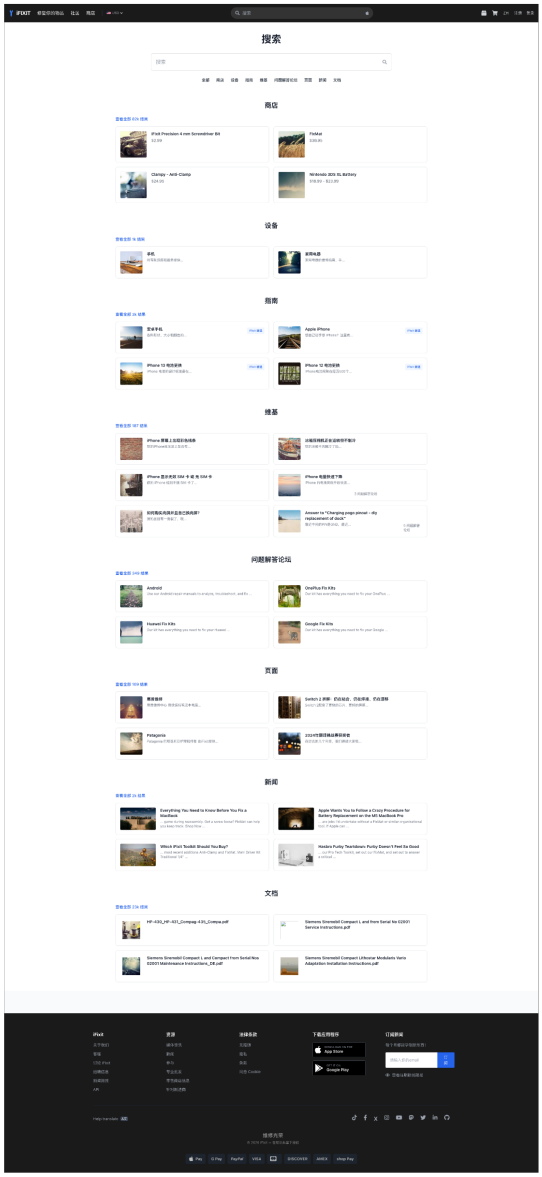


Figure 21: Demo case of Video2Code on a scroll interaction. The target video frames and generated full-screen renderings illustrate how Video2Code reproduces the post-scroll visual state.