DIGIDECK: EVALUATION OF VISUAL WEB AGENTS IN SYNTHETIC INTERACTIVE ENVIRONMENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Evaluating multimodal web agents on the live web is challenging due to limited reproducibility, bot detection, and the potential for irreversible financial, privacy, or security consequences. Sandboxed environments mitigate these risks, but incur high development overhead and often fail to capture the diversity and complexity of real-world web interactions. To address this issue, we introduce DIGIDECK, an automated framework for generating synthetic, interactive web environments for safe and reproducible evaluation of visual web agents. DIGIDECK uses coding LLMs as web world models to predict action-conditioned state transitions in the form of browser-renderable environments, enabling interactive synthetic web trajectories without relying on live services. We demonstrate DIGIDECK's utility by evaluating how agent performance varies across two dimensions: domain shifts and intent requirements. Using Online-Mind2Web as an anchor, we construct a benchmark of 1,080 tasks that systematically varies both intent (navigational, informational, and transactional) and domain distributions. Across seven agents, models show a consistent performance hierarchy across intents: performance decreases by 20.6 percentage points on average from navigational to informational tasks and by a further 17.8 points from informational to transactional tasks. On the distribution-shift task set, aggregate performance decreases by 5.2 percentage points, with substantial variation across models and a larger average informational-to-transactional gap.

1 INTRODUCTION

Multimodal web agents are increasingly capable of interacting with complex, visually rich interfaces, raising the prospect of agents that can perform a broad range of real-world tasks on the web (Gupta et al., 2026; Koh et al., 2024; Deng et al., 2023; Company, 2026; Nag, 2026; Gonzalez-Pumariega et al., 2025). Despite reported progress on established benchmarks, it remains unclear how well these capabilities generalize across shifts in task and website distributions. Evaluating this adaptability is difficult because current benchmarks face a recurring tension between diversity and controllability. Live-web benchmarks expose agents to diverse, naturally evolving interfaces, but suffer from poor reproducibility due to frequent updates and bot-detection systems, while also being unsuitable for tasks involving irreversible consequential actions that may incur real financial, privacy, or security risks (Gou et al.; Xue et al.; Yoran et al., 2024; Qiao et al., 2023). Sandboxed benchmarks provide reproducible and safe environments, but typically incur substantial development overhead and consequently exhibit limited coverage of websites, tasks, and workflows (Xu et al., 2026; Drouin et al., 2024; Koh et al., 2024; Zhou et al., 2024). As a result, existing evaluations provide limited evidence of how web agents behave as task requirements and web environments shift beyond familiar benchmark distributions.

To overcome the shortcomings of existing evaluation methods, we introduce **DIGIDECK**, a framework for automatically synthesizing executable web environments for multimodal web-agent evaluation. DIGIDECK treats LLMs as web world models (Gu et al.): conditioned on the current web state and an agent action, an LLM predicts the resulting state, which a coding model renders as an interactive browser interface. Unlike prior work that uses coding models to generate static websites (Wu et al., 2026; Zheng et al., 2026), DIGIDECK synthesizes *action-conditioned trajectories*, allowing the environment to evolve in response to agent behavior. Because visual agents interact

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

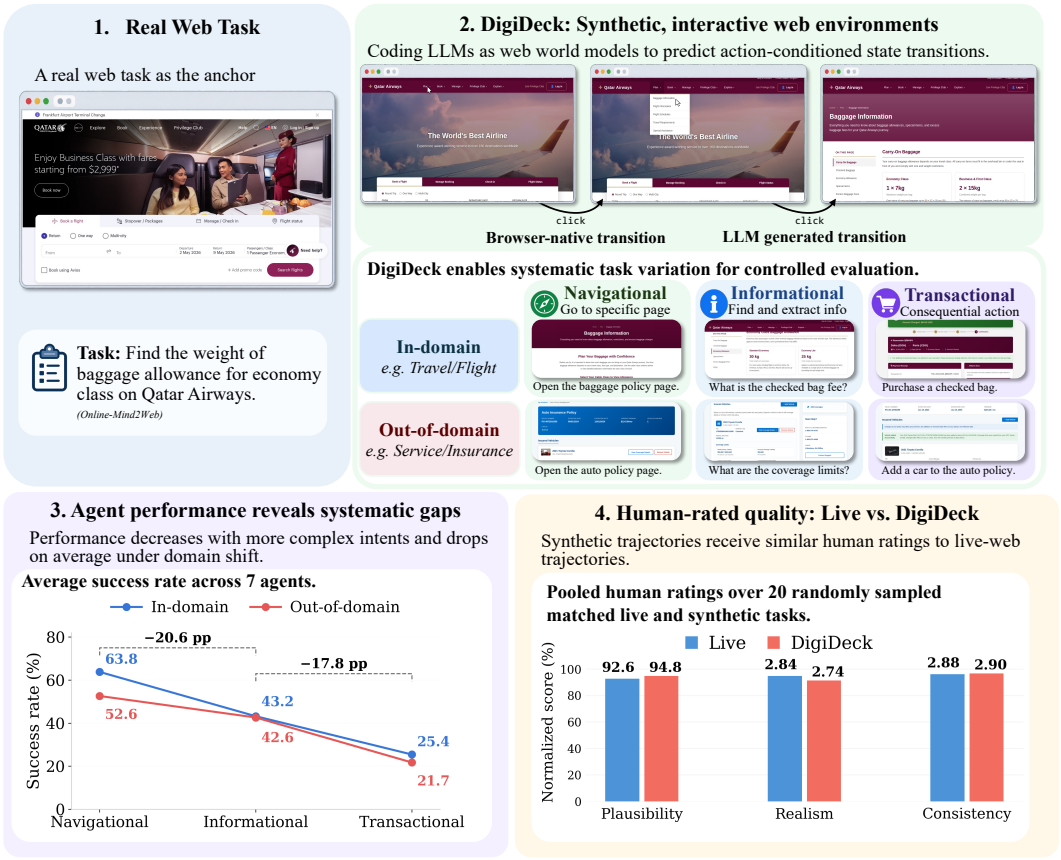

Figure 1: **Overview of DIGIDECK.** Starting from a real web task, DigiDeck constructs browser-renderable environments whose states evolve through native browser execution or action-conditioned LLM generation. The framework enables controlled variation of task intent and domain, revealing substantial performance degradation from navigational to informational to transactional tasks across seven agents. Human evaluation further shows similar ratings between live and synthetic trajectories on transition plausibility, realism, and consistency.

with rendered interfaces, DIGIDECK can synthesize plausible observable states that support controlled evaluation without reconstructing the full backend logic of the underlying service.

Our human evaluation shows that DIGIDECK trajectories receive ratings comparable to real web trajectories in realism, broken-element incidence, and next-state plausibility. As a result, DIGIDECK enables the study of the following research question:

Research Question. How does multimodal web-agent performance change as task requirements and web-environment distributions shift beyond established benchmark settings?

To answer this question, we anchor our analysis in Online-Mind2Web (Xue et al.). We first annotate its 300 tasks by domain, identifying 30 domains represented in the benchmark. To study distributional robustness, we additionally identify 34 domains that are absent in Online-Mind2Web and construct 60 tasks spanning these domains. We then create an intent triplet for each of the resulting 360 tasks, yielding a benchmark of 1,080 tasks. Within each triplet, the core task context is maintained, while the task is minimally modified to instantiate each of the three Broder intents (Broder, 2002): navigational, informational, and transactional. This controlled construction allows us to examine the effect of task intent separately from the underlying domain. Across seven agents, we observe a consistent hierarchy across intents: navigational tasks are easiest, followed by informational and then transactional tasks. Performance decreases by 20.6 percentage points on average from navigational to informational tasks and by a further 17.8 points from informational to transactional tasks. In unseen domains, the average informational-to-transactional gap increases from 17.8 percentage points on benchmark-anchored tasks to 21.0 points on the distribution-shift task set, although the direction and magnitude of this change vary across models.

In summary, our contributions are as follows:

- **DIGIDECK, an automated framework for executable synthetic web environments.** DIGIDECK uses LLMs to predict action-conditioned web-state transitions and coding LLMs to render them as interactive browser environments, enabling safe evaluation of consequential workflows. A human evaluation indicates that synthetic trajectories show comparable quality to live-web trajectories.[1]
- **A controlled benchmark of domain and intent requirement shifts.** We construct a benchmark of 1,080 tasks, comprising intent triplets derived from 300 Online-Mind2Web tasks and 60 additional tasks from previously unseen domains. Within each triplet, we approximately hold the task context fixed while varying the intent across navigational, informational, and transactional requirements.
- **A systematic evaluation of seven multimodal web agents across intent and domain shifts.** We find that performance drops by 20.6 points from navigational to informational tasks and by a further 17.8 points from informational to transactional tasks. On unseen domains, performance variations are model specific, with an average decrease of 5.2 points.

2 RELATED WORK

Recent multimodal web agents increasingly operate directly on rendered interfaces rather than HTML or accessibility trees (Company, 2026; Xu et al., 2025; Chen et al., 2025; Gu et al., 2025). Existing evaluations broadly fall into two categories. *Live-web benchmarks* (Pan et al.; Xue et al.; Yoran et al., 2024; He et al., 2024) benefit from the diversity of the live web, but are difficult to reproduce and cannot safely support many consequential interactions. In contrast, *sandboxed benchmarks* (Zhou et al., 2024; Koh et al., 2024; Garg et al., 2025; Miyai et al., 2025) enable controlled evaluation of such interactions, but incur substantial development overhead and are therefore typically limited to a relatively small range of websites and workflows.

More closely related to our work, LLMs have been used to predict future interface states for web and mobile interfaces (Gu et al.; Chae et al., 2025; 2026; Luo et al., 2025), while coding models have been used to synthesize executable websites or state-driven environments (Zheng et al., 2026; Wang et al., 2026; Wu et al., 2026; Yuan et al., 2026). DIGIDECK differs by treating environment generation as an *action-conditioned simulation*: rather than requiring a complete website implementation or restricting interaction to a predefined transition graph, it generates browser-renderable states conditioned on the current state and agent action. This enables controlled evaluation across diverse interactions without relying on live services. We provide a more comprehensive comparison with related work in Appendix A.

3 DIGIDECK

DIGIDECK is a framework for evaluating multimodal web agents in dynamically synthesized, browser-renderable environments. Figure 2 illustrates the pipeline: given a natural-language task and a target website, DIGIDECK constructs an initial website scaffold containing relevant transitions, which is used as context for initial-state generation. The initial state is then provided to the policy model, a visual web agent, which selects the next action for the given task. Given the scaffold, current state, and agent action, DIGIDECK then predicts and renders the corresponding successor state. This enables plausible multi-step interaction without requiring a complete implementation of the original website or access to the live web.

Website scaffold. Given a task $q = (u, w)$, where u is a natural language instruction and w a target website, DIGIDECK first constructs a hierarchical website scaffold of depth d

$$\mathcal{G} = G_{\text{scaffold}}(q, d), \tag{1}$$

which describes the pages, interface components, entities, and navigation relationships needed to complete the task. Preliminary qualitative experiments showed that conditioning next-state generation directly on the task can cause the code-LLM to shortcut toward task completion. Instead, the scaffold ensures that the website supports the task without directly facilitating its solution. Conditioning next-state generation on the scaffold also helps maintain consistency across the trajectory.

[1] https://anonymous.4open.science/r/digideck-24BE

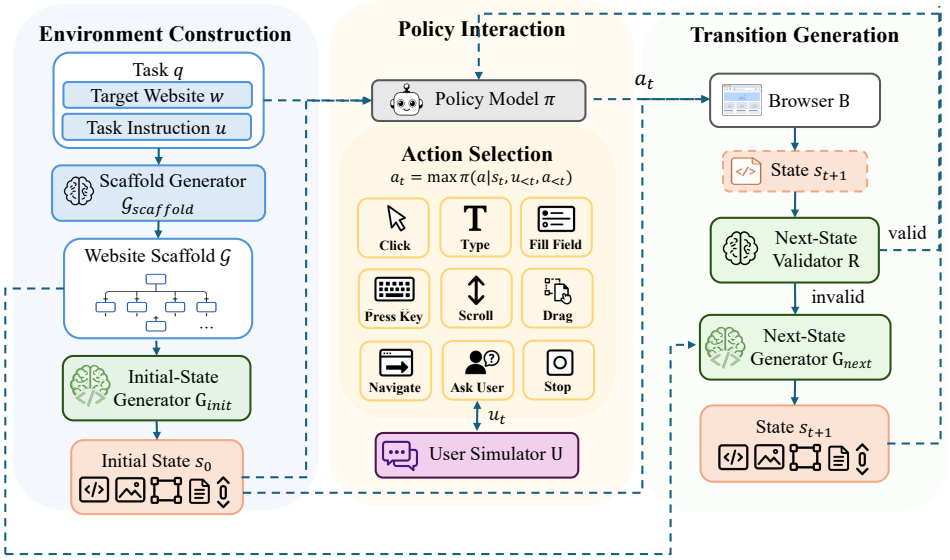

Figure 2: **DIGIDECK pipeline.** Given a task, DIGIDECK constructs a website scaffold and initial browser-renderable state. Agent actions are first executed directly in the browser; when the existing page cannot realize the expected action, a web world model synthesizes the successor state.

Initial state. Conditioned on the scaffold, an initial-state generator produces

$$s_0 = G_{\text{init}}(\mathcal{G}). \tag{2}$$

Each state consists of HTML/CSS/JavaScript code, a state description, and a scroll position. The resulting code is rendered in a browser B to obtain the visual screenshot, and grounding bounding boxes are shown to the agent.

Hybrid state transitions. Modeling the consequences of arbitrary agent actions would require enumerating and implementing all possible future interactions through functional frontend and backend logic, which is computationally expensive and wasteful because many actions may never occur. Instead, DIGIDECK combines native browser execution with an action-conditioned web world model. For each action a_t, DIGIDECK first executes the action on the currently rendered page using ordinary browser automation,

$$\tilde{s}_{t+1} = B(s_t, a_t), \tag{3}$$

where B denotes the browser transition. DIGIDECK then validates whether the resulting state plausibly realizes the requested interaction. The transition validator R returns 1 if the transition is valid given the action, and 0 otherwise. Thus, the next state is selected as

$$s_{t+1} = \begin{cases} \tilde{s}_{t+1}, & \text{if } R(s_t, \tilde{s}_{t+1}, a_t) = 1, \\ G_{\text{next}}(s_t, a_t, \mathcal{G}), & \text{otherwise}, \end{cases} \tag{4}$$

where G_{next} is the action-conditioned successor-state generator. This browser-first strategy preserves native browser behavior whenever the generated page already supports the requested interaction, such as scrolling, typing into a field, or opening a drop-down menu. When the existing page cannot produce the expected outcome, G_{next} generates a plausible successor state for the action conditioned on the current state and website scaffold by routing generation among four transition types: producing a new page, modifying part of the current page, restoring a previous page, or making no change. The resulting state is rendered in the browser, and any viewport errors are provided as feedback to the generator for correction.

Policy interaction. At step t, the evaluated policy π receives the original task instruction u and current rendered observation o_t, together with the action history. The policy returns a browser action $a_t \in A$, where A is the action space, including actions such as clicking, typing, scrolling, pressing a key, navigating between pages or tabs, requesting information from the user, or terminating the task. Refer to Table 6 in the Appendix for a full list of possible actions. DIGIDECK exposes only the rendered browser interface and policy-specific grounding information; the agent does not observe

the website scaffold or privileged environment state.

$$a_t \sim \pi(\cdot \mid u, o_t, a_{<t}) \tag{5}$$

User simulation. Many real web workflows require information intentionally absent from the task instruction, such as contact information, preferences, or confirmation of an action. Following prior interactive-agent evaluations that incorporate simulated users (Yao et al., 2025), DIGIDECK supports an `ask_user` action. When invoked, a user-simulation model generates a response conditioned on the user-interaction history h_t to promote consistency,

$$r_t = U(h_t), \tag{6}$$

after which interaction resumes from the unchanged browser state.

Termination and evaluation. There are three termination conditions: policy-initiated termination, exhaustion of the step budget, or an unrecoverable execution error, such as the repeated emission of invalid actions. The resulting sequence of observations, actions, and rendered states forms the trajectory ξ. A separate trajectory-level evaluator determines whether the task was completed successfully.

$$y = \mathcal{V}(q, \xi), \qquad y \in \{0, 1\}. \tag{7}$$

4 BENCHMARK CONSTRUCTION

To evaluate web agents across both task requirements and domain distribution shifts, we construct a benchmark that varies two dimensions: *user intent* and *task domain*. We anchor our dataset in 300 source tasks from Online-Mind2Web (Xue et al.) and add 60 source tasks spanning domains absent from Online-Mind2Web. For each source task, we construct a matched triplet for navigational, informational, and transactional intent while preserving the website, task subject, and relevant constraints as closely as possible. The resulting benchmark contains $360 \times 3 = 1{,}080$ task instances, balanced across the three intent conditions and spanning 64 domains: 30 benchmark-anchored domains and 34 distribution-shift domains.

Intent-Controlled Task Triplets. Broder's taxonomy categorizes search queries into three intents: *navigational*, *informational*, and *transactional* (Broder, 2002). We adapt this taxonomy to web tasks and apply it to the 300 tasks in Online-Mind2Web (Xue et al.), obtaining 21 navigational, 230 informational, and 49 transactional source tasks. For each source task, we retain the original instruction and construct counterfactual instructions for the other two intents while preserving the website, target entity, and core task context whenever possible, resulting in 900 tasks. For example, an informational task asking for the price of a product can be paired with a navigational task asking the agent to locate that product and a transactional task asking the agent to purchase it. We generate candidate transformations by prompting GPT-5 with the source instruction and target intent, and manually inspect each candidate to ensure that (1) it matches the intended category, (2) the core task context remains aligned across the triplet, and (3) the resulting task is concrete and executable. Failed candidates were regenerated. Additional examples are provided in Table 7 in the Appendix.

Domain Distribution Shift. We next construct a complementary set of tasks from domains absent from Online-Mind2Web in order to evaluate whether intent-dependent performance patterns generalize beyond the benchmark's original domain distribution. To characterize the domains represented in Online-Mind2Web, we use the subdomain taxonomy introduced by Mind2Web (Deng et al., 2023) and Mind2Web-Live (Pan et al.). Of the 300 Online-Mind2Web source tasks, 225 are derived from Mind2Web or Mind2Web-Live (Pan et al.); for these tasks, we recover the corresponding Mind2Web subdomain annotations. We manually categorize the remaining 75 tasks using the same taxonomy when appropriate. This process identifies 30 domains represented in Online-Mind2Web. We then identify 34 additional web-task domains that are absent from these 30 benchmark-anchored domains. From these domains, we construct 60 source tasks by prompting GPT-5 to propose candidates using the original tasks as in-context examples. The resulting tasks are manually inspected to ensure that they (1) belong to an unseen domain, (2) are concrete and executable, and (3) are comparable in form and complexity to the original tasks. Finally, we apply the same matched intent-transformation procedure described above to each of the 60 distribution-shift source tasks, yielding 180 tasks.

5 HUMAN EVALUATION

We evaluate both the quality of DIGIDECK's synthetic trajectories and the reliability of the evaluator against human judgments. Synthetic trajectories are rated similarly to live-web trajectories in

transition plausibility, realism, consistency, and broken-element incidence, while human agreement with the automatic evaluator is comparable to prior work (Xue et al.), from which our evaluator is adapted.

Synthetic Trajectory Quality. We ask annotators to evaluate 20 live-web trajectories and their corresponding DIGIDECK trajectories without being told the trajectory source. Trajectories were randomly sampled from Gemini 3.1 Pro runs. For each task, we additionally evaluate the remaining trajectories in its intent triplet, yielding 80 trajectories and over 800 web states, with an average of 3.4 annotations per state. Annotators evaluate trajectories along four dimensions: two state-level and two trajectory-level dimensions. The state-level dimensions are *next-state plausibility* (plaus.), i.e., whether the web state plausibly follows the previous state given the web action, and *broken elements*, i.e., whether the state contains any broken elements and, if so, whether they are minor, major, or blocking. Annotators then judge the overall realism (real.) of the trajectory, i.e., how realistic the webpage appears, and its consistency (cons.), i.e., whether the transitions and visual style remain consistent over time. They also reported whether the task was successful or not. Reported metrics are computed by pooling individual annotator judgments within each setting, allowing each annotation to contribute to the final rating. For broken elements, each state-level annotation contributes independently to the counts for none, minor, major, or blocking severity. For all other metrics, we average the scores. We report Gwet's AC1 for annotator agreement on realism and consistency and exact pairwise agreement for next-state plausibility and broken-element severity.

Table 1 shows that both real and synthetic trajectories achieve high levels of next-state transition plausibility. Real trajectories achieve 92.6% transition plausibility, compared with 94.8% for synthetic trajectories. Human-rated agent success, measured as the proportion of annotators judging each trajectory successful, was positively correlated across the 20 matched real–synthetic task pairs (Pearson's $r = 0.62$). These results indicate that the action-conditioned transitions produced by DIGIDECK are comparable in plausibility to corresponding real-web interactions. We also observe that synthetic trajectories contain a comparable proportion of broken elements and score similarly to live-web trajectories in terms of plausibility, realism, and consistency. The main difference is that 9.5% of states contain at least one image that could not be generated due to text-to-image model safety filters. We report these image-generation failures separately from simulator errors because they do not arise from the simulator's transition dynamics or webpage functionality. Importantly, annotators judged 95.6% of these image failures to be only minor issues without affecting webpage functionality. We additionally examine whether DIGIDECK's generation quality varies across the navigational, informational, and transactional tasks used in our benchmark. Table 1 reports human-evaluation results separately for each synthetic intent category. Transition plausibility remains high across all three categories (89.9–97.3%), suggesting that differences in agent performance across intents are unlikely to be explained by systematic differences in transition plausibility. Broken-element incidence is also relatively stable, and the variation does not mirror the downstream performance ordering across intents.

Setting	Plaus.↑	Real.↑	Cons.↑	Minor(%)↓	Major(%)↓	Block.(%)↓
Trajectory source						
Real	92.6	2.84	2.88	4.2	2.0	1.1
Synthetic	94.8	2.74	2.90	2.4	1.9	1.2
Synthetic by intent						
Navigational	97.3	2.76	2.90	2.0	3.0	0.0
Informational	89.9	2.75	2.91	3.3	4.1	2.1
Transactional	95.8	2.71	2.90	2.1	0.6	1.2
Total	**94.8**	**2.74**	**2.90**	**2.4**	**1.9**	**1.2**
Annotator agreement						
Agreement	$91.6\%_{\text{pair}}$	0.639_{AC1}	0.827_{AC1}		$89.9\%_{\text{pair}}$ (severity)	

Table 1: Human evaluation of real and synthetic trajectories.

Human Agreement with Automatic Evaluation. We also evaluate the automatic evaluation method used for our tasks. We collect two annotations for 144 randomly sampled trajectories, split equally across informational, navigational, and transactional tasks. We ask human annotators to judge whether each task was successfully completed before seeing the evaluator's judgment (*Initial*). In cases of disagreement, we then show the evaluator's explanation and ask annotators whether they agree with it (*After Expl.*). Table 2 shows that

Setting	Evaluator Agreement			Annotator Agreement			
	Initial		After Expl.	Initial		After Expl.	
	Agree (%)	F_1 (%)	Agree (%)	Agree (%)	κ	Agree (%)	κ
Overall	**80.2**	**86.5**	**86.1**	**84.0**	**0.497**	**84.7**	**0.361**
Informational	71.9	79.1	83.3	77.1	0.433	79.2	0.250
Navigational	86.5	92.1	87.5	89.6	0.555	87.5	0.429
Transactional	82.3	86.8	87.5	85.4	0.500	87.5	0.429

Table 2: Human annotation of automatic evaluator quality.

Model	Benchmark-Anchored				Distribution-Shift			
	Navigational	Informational	Transactional	Average	Navigational	Informational	Transactional	Average
Gemini 3.1 Pro	84.00 [77.00, 91.00]	59.00 [50.00, 69.00]	51.00 [41.00, 61.00]	64.67 [58.67, 70.67]	65.00 [53.33, 76.67]	66.67 [55.00, 78.33]	45.00 [33.33, 56.67]	58.89 [50.00, 67.22]
GPT-5	88.00 [81.00, 94.00]	59.00 [49.00, 69.00]	39.00 [30.00, 49.00]	62.00 [56.67, 67.33]	66.67 [55.00, 78.33]	58.33 [45.00, 71.67]	35.00 [23.33, 46.67]	53.33 [44.44, 62.22]
GLM5V-turbo	65.33 [59.67, 70.67]	43.00 [37.33, 48.67]	27.33 [22.33, 32.33]	45.22 [41.67, 48.78]	66.67 [55.00, 78.33]	56.67 [43.33, 70.00]	18.33 [8.33, 28.33]	47.22 [39.44, 55.00]
Gemini 3.1 Flash Lite	61.67 [56.33, 67.00]	42.33 [36.67, 48.00]	27.67 [22.67, 33.00]	43.89 [40.11, 47.56]	61.67 [49.96, 73.33]	43.33 [31.67, 55.00]	23.33 [13.33, 35.00]	42.78 [34.44, 51.11]
GPT-5-mini	69.33 [64.00, 74.33]	47.00 [41.33, 52.67]	16.00 [12.00, 20.33]	44.11 [40.78, 47.44]	43.33 [31.67, 56.67]	30.00 [18.33, 41.67]	5.00 [0.00, 11.67]	26.11 [19.44, 32.78]
Qwen3-VL 30B	33.67 [28.33, 39.00]	22.00 [17.33, 26.67]	6.33 [3.67, 9.33]	20.67 [17.67, 23.89]	28.33 [16.67, 40.00]	16.67 [8.33, 26.67]	11.67 [5.00, 20.00]	18.89 [12.22, 26.11]
Qwen3-VL 235B	44.33 [38.67, 50.00]	30.00 [24.67, 35.33]	10.67 [7.33, 14.33]	28.33 [24.89, 31.78]	36.67 [25.00, 48.33]	26.67 [15.00, 38.33]	13.33 [5.00, 23.33]	25.56 [18.33, 33.33]

Table 3: Performance (%) across intent categories under benchmark-anchored and distribution-shift evaluation. Brackets show 95% bootstrap confidence intervals. Average is the mean of the three intent rates.

annotators initially agree with the evaluator 80.2% of the time, similar to the agreement reported in the earlier work from which our evaluator is adapted (Xue et al.). After showing the evaluator's rationale, agreement increases to 86.1%, indicating that humans tend to agree with the explanations.

6 Experiments

We briefly summarize the experimental setup and implementation details here; Appendix B provides the complete configuration, including prompts, hyperparameters, and model-specific settings.

Experimental Setup. We evaluate seven multimodal web agents spanning four model families: Gemini 3.1 Pro, GPT-5, GLM5V-turbo, Gemini 3.1 Flash Lite, GPT-5-mini, Qwen3-VL 235B, and Qwen3-VL 30B. For each source task, DigiDeck generates a single website scaffold and initial state that are shared across all intents and reused across all model runs. The three intent variants are jointly provided as context during scaffold generation to ensure that the resulting environment supports all three tasks. Agents receive the rendered browser screenshot together with grounding information. For the Qwen3-VL models, we instead follow the prompt template recommended by the model authors and omit additional grounding information, as we found that modifying the recommended interface degraded performance. Across all other agents, the prompting setup is held fixed. All agents are allocated a maximum of 60 steps to complete the task.

Evaluation. The evaluator, implemented using `GPT-5-mini`, measures task success. To quantify the performance differences associated with intent requirements, we additionally report the absolute performance changes from navigational to informational tasks, $\Delta_{\mathrm{N}\to\mathrm{I}}$, and from informational to transactional tasks, $\Delta_{\mathrm{I}\to\mathrm{T}}$. To measure sensitivity to domain distribution shift, we report Δ_{OOD}, defined as the difference between distribution-shift and benchmark-anchored average success.

Implementation Details. DigiDeck uses `nvidia/GLM-5.2-NVFP4` (Zeng et al., 2026) for website-scaffold construction, successor-state generation, transition validation, and simulated-user interactions. The model is served locally on 8× NVIDIA B200 GPUs. Generated HTML states are rendered in isolated headless Chromium contexts and browser actions are executed using Playwright. We use FLUX.2 Klein 4B (Curvo et al., 2026) to generate image assets. All seven agents use the same environment generator, initial environments, 60-action budget, transition-validation procedure, simulated-user configuration, and trajectory evaluator. We use a fixed random seed of 42. Additional implementation parameters, including scaffold depth, retry budgets, policy context windows, grounding configuration, and state-history settings, are summarized in Table 5.

7 Results

Average model performance across intents. Table 3 shows the success rate of each of the seven agents across intent and domain shifts. We follow each model's recommended interaction interface, which introduces some differences in prompting

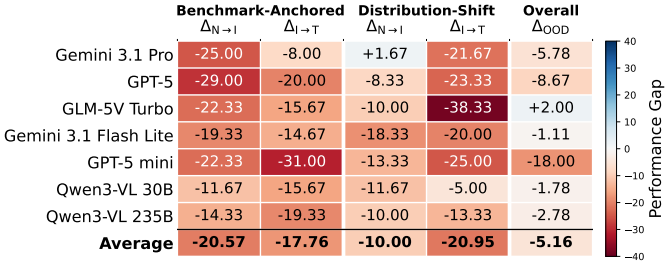

Figure 3: Performance changes across intent and domain shifts.

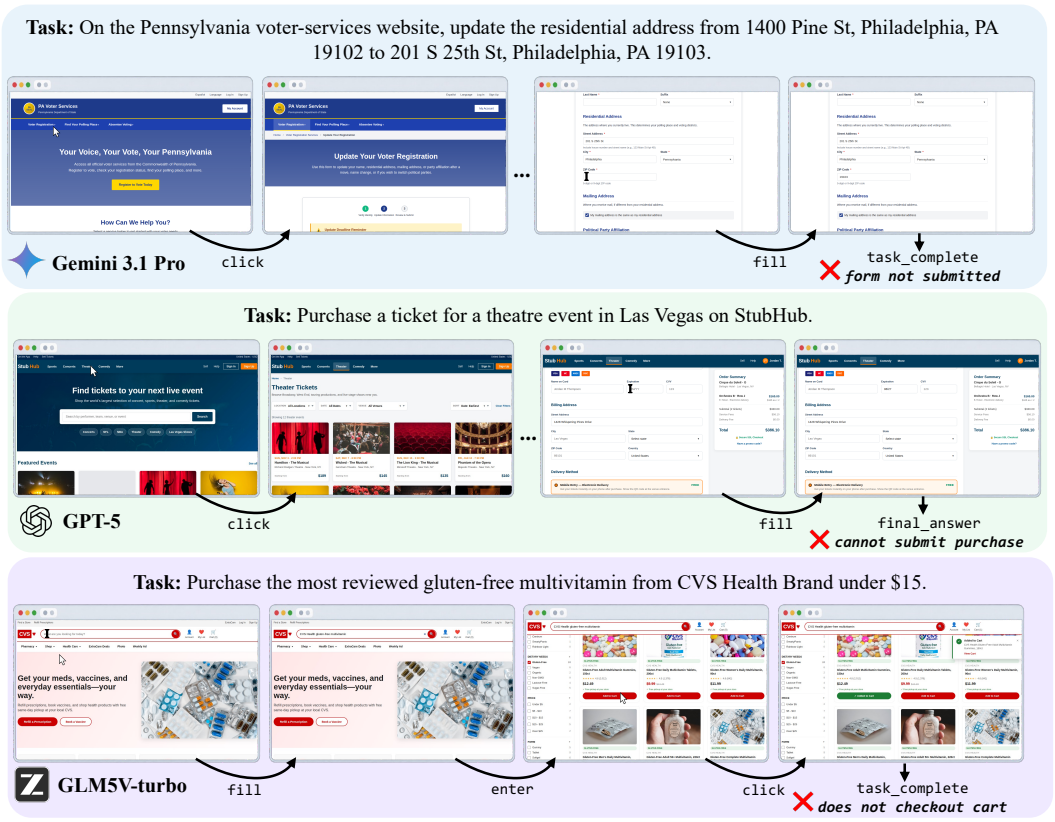

Figure 4: Qualitative examples of models failing at the transactional step.

and grounding, particularly for Qwen3-VL. Accordingly, we emphasize within-model changes across evaluation conditions rather than absolute cross-model rankings. Figure 3 illustrates that under the benchmark-anchored evaluation, performance declines monotonically across all seven agents as tasks progress from navigational to informational to transactional intent. Average success falls from 63.76% on navigational tasks to 43.19% on informational tasks, a 32.26% relative decrease, and further to 25.43% on transactional tasks, an additional 41.12% relative decrease. This ordering holds for every evaluated model in the benchmark-anchored setting, indicating that the degradation is not driven by a particular model family or capability tier. Notably, stronger navigational and informational performance does not necessarily translate into comparable transactional reliability: GPT-5 achieves the highest navigational success rate and ties for the highest informational success rate (88.00% and 59.00%, respectively), yet its transactional success falls to 39.00%. In contrast, Gemini 3.1 Pro achieves a substantially higher transactional success rate of 51.00%, yielding the best benchmark-anchored overall average of 64.67%. The informational-to-transactional gap is substantial across the model set, averaging 17.76 percentage points and reaching 31.00 points for GPT-5-mini. A similar aggregate pattern persists under distribution shift, where average performance decreases from 52.62% on navigational tasks to 42.62% on informational tasks and 21.67% on transactional tasks, although the per-model decline is no longer strictly monotonic. Overall, these results suggest that tasks requiring persistent state-changing actions pose additional challenges beyond locating pages or retrieving information, and that navigational and informational performance alone may therefore provide an incomplete picture of agent reliability for consequential web workflows.

What type of errors drive low performance in transactional tasks? As shown in Figure 4, transactional failures frequently involve reaching an intermediate state without completing the requested state-changing operation. Examples include adding an item to a cart without purchasing it, filling an application without submitting it, and opening a contact workflow without sending the message. Transactional failures may partly reflect deliberate safeguards against irreversible actions, although this behavior is not consistent across models or tasks. A second possible explanation is that the difficulty of collecting end-to-end transactional trajectories at scale without incurring financial or privacy costs leaves models with less training signal for the final execution step.

Does trajectory length explain the transactional performance gap? Figure 5 shows the average performance of the seven agents on benchmark-anchored tasks as a function of trajectory length. We observe substantial variability in performance across trajectory lengths. Despite this variability, transactional tasks tend to underperform navigational and informational tasks across both shorter and longer trajectories, suggesting that the transactional performance gap is not solely attributable to trajectory length. However, because the tasks represented within each length bin may differ, this analysis should be interpreted as descriptive rather than as a controlled comparison.

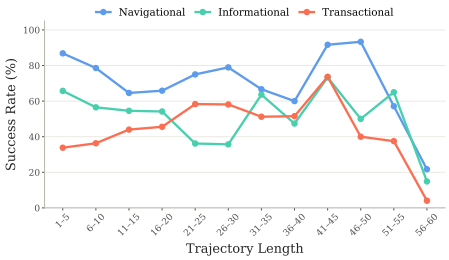

Figure 5: Success rate by observed trajectory length across intents.

Does website complexity account for the performance gap? We analyze the complexity of the generated task environments using deterministic proxies computed from the HTML states visited during each trajectory adapted from contemporary work (Yuan et al., 2026). The analysis reported here covers 300 trajectories corresponding to 100 complete matched triplets, with one navigational, one informational, and one transactional trajectory for every source task using Gemini 3.1 Pro trajectories. Each trajectory is analyzed independently. For every rendered state, an HTML parser counts navigation options, interaction controls, actionable elements, visual elements, complex visual elements, and visible words. Details are provided in Appendix D. Figure 6 reports all six metrics, showing little variation across intents, suggesting that the observed performance gap is not readily explained by coarse measures of interface complexity alone.

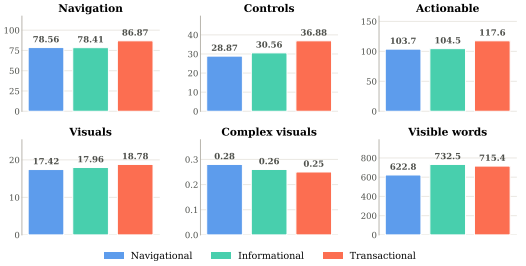

Figure 6: Website complexity across web intents.

8 Conclusion, Limitations and Future Work

DigiDeck is a framework that combines browser-native execution with coding LLMs as web world models to evaluate visual web agents in controlled scenarios that are difficult to study on the live web. By generating action-conditioned state transitions, DigiDeck avoids the need to fully implement backend logic while supporting a flexible range of interactions. We demonstrate its utility by constructing 1,080 tasks that systematically vary both user intent and domain. Across seven agents, performance decreases substantially from navigational to informational to transactional tasks. Generalization to unseen domains is more model-dependent, but on average we observe a wider informational-to-transactional performance gap under distribution shift.

DigiDeck is a simulator rather than a complete reconstruction of the live web, and generated environments may therefore differ from their real-world counterparts in interface details or transition dynamics. Our human evaluation finds similar transition plausibility, realism, consistency, and broken-element incidence between synthetic and live-web trajectories, while performance on paired real and synthetic trajectories shows positive correspondence. Nevertheless, future work should more systematically characterize the sim-to-real gap between synthetic and live-web environments. Due to computational constraints, the current implementation also relies on a single web-world model; future work could study how the choice of model affects simulator quality and downstream evaluation. Although we validate both generated trajectories and evaluator judgments against human annotations, alternative environment generators or evaluators may produce different absolute results. Finally, DigiDeck's ability to support interactions that may not be implemented in a fixed website, such as accessing an email interface, comes at a higher computational cost than replaying a static benchmark.

Beyond evaluation, DigiDeck could support automated identification of visual web-agent failure modes and provide interactive environments for training agents, including reinforcement-learning settings. More broadly, we view action-conditioned environment generation as a promising approach for studying and improving visual web agents in settings where live-web interaction is unsafe, expensive, or difficult to reproduce.

AI USE STATEMENT

In this work, we used generative AI tools to generate synthetic datasets, implement methods, support qualitative and thematic data analysis, and assist with the interpretation of results. We did not use generative AI tools to develop theoretical models or conceptual frameworks, formulate mathematical claims, propose or refine hypotheses, assist with translation, or clean and reformat datasets. Providing critical ingredients for proving mathematical claims and assisting in the writing of proofs were not applicable to this work. Additionally, we used generative AI tools to create or modify scientific figures and images, create or edit software code, and edit the manuscript to improve readability. We reviewed and verified AI-assisted outputs, including manually inspecting generated synthetic data both internally and externally through volunteers participating in the human study, validating generated or modified code through testing, checking figures and visual artifacts for correctness, and reviewing all AI-assisted analyses and manuscript edits for accuracy and consistency with the underlying experiments and results. We take responsibility for the final content of this work, including text, claims, code, data, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

DIGIDECK is partly motivated by concerns regarding deploying and evaluating web agents on live services, in which consequential actions could incur financial, privacy, or security costs. DIGIDECK proposes a method for evaluating such scenarios without endangering users' information since the simulated users only provide fictional or generated data to allow task completion. Our human evaluation was conducted with volunteers from our extended research group. Participation was voluntary, and no sensitive personal information was collected as part of the study. We recognize that systems capable of generating realistic interactive websites may have potential dual-use applications, including the creation of deceptive interfaces. Our work is intended for controlled research on web-agent evaluation and robustness, and the generated environments operate in isolated settings without connection to the backend systems of the corresponding real-world services.

REPRODUCIBILITY STATEMENT

We provide implementation and evaluation details throughout the main paper and supplementary material to support reproduction of our results. Section 6 details the experimental setup, which is discussed in greater detail in Appendix B, including the complete set of prompts, configurations, and other hyperparameters used. To further facilitate replication, we release all code at https://anonymous.4open.science/r/digideck-24BE/.

REFERENCES

Andrei Broder. A taxonomy of web search. In *ACM Sigir forum*, volume 36, pp. 3–10. ACM New York, NY, USA, 2002.

Hyungjoo Chae, Namyoung Kim, Kai Ong, Minju Gwak, Gwanwoo Song, Jihoon Kim, Sunghwan Kim, Dongha Lee, and Jinyoung Yeo. Web agents with world models: Learning and leveraging environment dynamics in web navigation. In *International Conference on Learning Representations*, volume 2025, pp. 63707–63738, 2025.

Hyungjoo Chae, Jungsoo Park, and Alan Ritter. Safe and scalable web agent learning via recreated websites. *arXiv preprint arXiv:2603.10505*, 2026.

Wentong Chen, Junbo Cui, Jinyi Hu, Yujia Qin, Junjie Fang, Yue Zhao, Chongyi Wang, Jun Liu, Guirong Chen, Yupeng Huo, et al. Guicourse: From general vision language model to versatile gui agent. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 21936–21959, 2025.

H Company. Holo3.1: Fast local computer use agents, 2026. URL `https://huggingface.co/Hcompany/Holo3.1-35B-A3B`.

Pedro M. P. Curvo, Maksim Zhdanov, Floor Eijkelboom, and Jan-Willem van de Meent. Follow the mean: Reference-guided flow matching, 2026.

Yutong Dai, Krithika Ramakrishnan, Jing Gu, Matthew Fernandez, Yanqi Luo, Viraj Prabhu, Zhenyu Hu, Silvio Savarese, Caiming Xiong, Zeyuan Chen, and Ran Xu. Scuba: Salesforce computer use benchmark. In *International Conference on Learning Representations*, 2026.

Nicola Dainese, Matteo Merler, Minttu Alakuijala, and Pekka Marttinen. Generating code world models with large language models guided by monte carlo tree search. *Advances in Neural Information Processing Systems*, 37:60429–60474, 2024.

Matt Deitke, Eli VanderBilt, Alvaro Herrasti, Luca Weihs, Kiana Ehsani, Jordi Salvador, Winson Han, Eric Kolve, Aniruddha Kembhavi, and Roozbeh Mottaghi. Procthor: Large-scale embodied ai using procedural generation. *Advances in Neural Information Processing Systems*, 35:5982–5994, 2022.

Matt Deitke, Christopher Clark, Sangho Lee, Rohun Tripathi, Yue Yang, Jae Sung Park, Mohammadreza Salehi, Niklas Muennighoff, Kyle Lo, Luca Soldaini, et al. Molmo and pixmo: Open weights and open data for state-of-the-art vision-language models. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 91–104, 2025.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2web: towards a generalist agent for the web. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, pp. 28091–28114, 2023.

Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator. In *Conference on robot learning*, pp. 1–16. PMLR, 2017.

Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H Laradji, Manuel Del Verme, Tom Marty, Léo Boisvert, Megh Thakkar, Quentin Cappart, David Vazquez, et al. Workarena: How capable are web agents at solving common knowledge work tasks? *arXiv preprint arXiv:2403.07718*, 2024.

Divyansh Garg, Shaun VanWeelden, Diego Caples, Andis Draguns, Nikil Ravi, Pranav Putta, Naman Garg, Tomas Abraham, Michael Lara, Federico Lopez, James Liu, et al. Real: Benchmarking autonomous agents on deterministic simulations of real websites. *arXiv preprint arXiv:2504.11543*, 2025.

Gonzalo Gonzalez-Pumariega, Vincent Tu, Chih-Lun Lee, Jiachen Yang, Ang Li, and Xin Eric Wang. Scaling agents for computer use. 2025.

Boyu Gou, Zanming Huang, Yuting Ning, Yu Gu, Michael Lin, Weijian Qi, Andrei Kopanev, Botao Yu, Bernal Jimenez Gutierrez, Yiheng Shu, et al. Mind2web 2: Evaluating agentic search with agent-as-a-judge. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Yu Gu, Kai Zhang, Yuting Ning, Boyuan Zheng, Boyu Gou, Tianci Xue, Cheng Chang, Sanjari Srivastava, Yanan Xie, Peng Qi, et al. Is your llm secretly a world model of the internet? model-based planning for web agents. *Transactions on Machine Learning Research*.

Zhangxuan Gu, Zhengwen Zeng, Zhenyu Xu, Xingran Zhou, Shuheng Shen, Yunfei Liu, Beitong Zhou, Changhua Meng, Tianyu Xia, Weizhi Chen, et al. Ui-venus technical report: Building high-performance ui agents with rft. *arXiv preprint arXiv:2508.10833*, 2025.

Tanmay Gupta, Piper Wolters, Zixian Ma, Peter Sushko, Rock Yuren Pang, Diego Llanes, Yue Yang, Taira Anderson, Boyuan Zheng, Zhongzheng Ren, et al. Molmoweb: Open visual web agent and open data for the open web. *arXiv preprint arXiv:2604.08516*, 2026.

Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. Webvoyager: Building an end-to-end web agent with large multimodal models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6864–6890, 2024.

Lawrence Keunho Jang, Jing Yu Koh, Daniel Fried, and Ruslan Salakhutdinov. Odysseys: Benchmarking web agents on realistic long horizon tasks. *arXiv preprint arXiv:2604.24964*, 2026.

Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Russ Salakhutdinov, and Daniel Fried. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 881–905, 2024.

Wolfgang Lehrach, Daniel Hennes, Miguel Lazaro-Gredilla, Xinghua Lou, Carter Wendelken, Zun Li, Antoine Dedieu, Jordi Grau-Moya, Marc Lanctot, Atil Iscen, et al. Code world models for general game playing. *arXiv preprint arXiv:2510.04542*, 2025.

Mingxin Li, Yanzhao Zhang, Dingkun Long, Keqin Chen, Sibo Song, Shuai Bai, Zhibo Yang, Pengjun Xie, An Yang, Dayiheng Liu, et al. Qwen3-vl-embedding and qwen3-vl-reranker: A unified framework for state-of-the-art multimodal retrieval and ranking. *arXiv preprint arXiv:2601.04720*, 2026.

Dezhao Luo, Bohan Tang, Kang Li, Georgios Papoudakis, Jifei Song, Shaogang Gong, Jianye Hao, Jun Wang, and Kun Shao. Vimo: A generative visual gui world model for app agents. *arXiv preprint arXiv:2504.13936*, 2025.

Atsuyuki Miyai, Zaiying Zhao, Kazuki Egashira, Atsuki Sato, Tatsumi Sunada, Shota Onohara, Hiromasa Yamanishi, Mashiro Toyooka, Kunato Nishina, Ryoma Maeda, Kiyoharu Aizawa, and Toshihiko Yamasaki. Webchorearena: Evaluating web browsing agents on realistic tedious web tasks. *arXiv preprint arXiv:2506.01952*, 2025.

Ruben Nag. Manus ai: From autonomous agent breakthrough to cross-border acquisition and regulatory scrutiny. *Open Access Cases*, 3(1), 2026.

Yichen Pan, Dehan Kong, Sida Zhou, Cheng Cui, Yifei Leng, Bing Jiang, Hangyu Liu, Yanyi Shang, Shuyan Zhou, Tongshuang Wu, et al. Webcanvas: Benchmarking web agents in online environments. In *Agentic Markets Workshop at ICML 2024*.

Bo Qiao, Liqun Li, Xu Zhang, Shilin He, Yu Kang, Chaoyun Zhang, Fangkai Yang, Hang Dong, Jue Zhang, Lu Wang, et al. Taskweaver: A code-first agent framework. *arXiv preprint arXiv:2311.17541*, 2023.

Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and service robotics: Results of the 11th international conference*, pp. 621–635. Springer, 2017.

Hao Tang, Darren Key, and Kevin Ellis. Worldcoder, a model-based llm agent: Building world models by writing code and interacting with the environment. *Advances in Neural Information Processing Systems*, 37:70148–70212, 2024.

Karen Ullrich, Jingtong Su, Claudia Shi, Arjun Subramonian, Amir Bar, Ivan Evtimov, Nikolaos Tsilivis, Randall Balestriero, Julia Kempe, and Mark Ibrahim. Openapps: Simulating environment variations to measure ui-agent reliability. In *International Conference on Learning Representations*, 2026.

Bowen Wang, Dunjie Lu, Junli Wang, Tianyi Bai, Shixuan Liu, Zhipeng Zhang, Haiquan Wang, Hao Hu, Tianbao Xie, Shuai Bai, et al. Cua-gym: Scaling verifiable training environments and tasks for computer-use agents. *arXiv preprint arXiv:2605.25624*, 2026.

Jason Wei, Zhiqing Sun, Spencer Papay, Scott McKinney, Jeffrey Han, Isa Fulford, Hyung Won Chung, Alex Tachard Passos, William Fedus, and Amelia Glaese. Browsecomp: A simple yet challenging benchmark for browsing agents. *arXiv preprint arXiv:2504.12516*, 2025.

Yifan Wu, Yiran Peng, Yiyu Chen, Jianhao Ruan, Zijie Zhuang, Cheng Yang, Jiayi Zhang, Man Chen, Yenchi Tseng, Zhaoyang Yu, et al. Autowebworld: Synthesizing infinite verifiable web environments via finite state machines. *arXiv preprint arXiv:2602.14296*, 2026.

Hongchi Xia, Xuan Li, Zhaoshuo Li, Qianli Ma, Jiashu Xu, Ming-Yu Liu, Yin Cui, Tsung-Yi Lin, Wei-Chiu Ma, Shenlong Wang, et al. Sage: Scalable agentic 3d scene generation for embodied ai. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 22358–22368, 2026.

Frank Fangzheng Xu, Yufan Song, Boxuan Li, Yuxuan Tang, Kritanjali Jain, Mengxue Bao, Zora Wang, Xuhui Zhou, Zhitong Guo, Murong Cao, et al. Theagentcompany: benchmarking llm agents on consequential real world tasks. *Advances in Neural Information Processing Systems*, 38, 2026.

Yiheng Xu, Zekun Wang, Junli Wang, Dunjie Lu, Tianbao Xie, Amrita Saha, Doyen Sahoo, Tao Yu, and Caiming Xiong. Aguvis: Unified pure vision agents for autonomous gui interaction. In *International Conference on Machine Learning*, pp. 69772–69805. PMLR, 2025.

Tianci Xue, Weijian Qi, Tianneng Shi, Chan Hee Song, Boyu Gou, Dawn Song, Huan Sun, and Yu Su. An illusion of progress? assessing the current state of web agents. In *Second Conference on Language Modeling*.

Yue Yang, Fan-Yun Sun, Luca Weihs, Eli VanderBilt, Alvaro Herrasti, Winson Han, Jiajun Wu, Nick Haber, Ranjay Krishna, Lingjie Liu, et al. Holodeck: Language guided generation of 3d embodied ai environments. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 16227–16237, 2024.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik R Narasimhan. {τ}-bench: A benchmark for \underline{T}ool-\underline{A}gent-\underline{U}ser interaction in real-world domains. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=roNSXZpUDN.

Ori Yoran, Samuel Joseph Amouyal, Chaitanya Malaviya, Ben Bogin, Ofir Press, and Jonathan Berant. Assistantbench: Can web agents solve realistic and time-consuming tasks? In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 8938–8968, 2024.

Peng Yuan, Yuyang Yin, Yuxuan Cai, and Zheng Wei. Webforge: Breaking the realism-reproducibility-scalability trilemma in browser agent benchmark. *arXiv preprint arXiv:2604.10988*, 2026.

Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chenghua Huang, Chengxing Xie, et al. Glm-5: from vibe coding to agentic engineering. *arXiv preprint arXiv:2602.15763*, 2026.

Xianren Zhang, Shreyas Prasad, Di Wang, Qiuhai Zeng, Suhang Wang, Wenbo Yan, and Mat Hans. A functionality-grounded benchmark for evaluating web agents in e-commerce domains. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1512–1528, 2026.

Yuhao Zheng, Li'an Zhong, Yi Wang, Rui Dai, Kaikui Liu, Xiangxiang Chu, Linyuan Lv, Philip Torr, and Kevin Qinghong Lin. Code2world: A gui world model via renderable code generation. *arXiv preprint arXiv:2602.09856*, 2026.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, volume 2024, pp. 15585–15606, 2024.

Mingchen Zhuge, Changsheng Zhao, Haozhe Liu, Zijian Zhou, Shuming Liu, Wenyi Wang, Ernie Chang, Gael Le Lan, Junjie Fei, Wenxuan Zhang, et al. Neural computers. In *ICLR 2026 the 2nd Workshop on World Models: Understanding, Modelling and Scaling*.

A Related Work

Visual Web Agents and their Evaluation. Recent advances in multimodal language models (Li et al., 2026; Deitke et al., 2025) have accelerated a shift in autonomous web-agent research toward visual grounding. Unlike agents that rely on HTML or accessibility trees, vision-based agents interact directly with the human-facing web interface, avoiding verbose and potentially irrelevant markup (Company, 2026; Xu et al., 2025; Chen et al., 2025; Gu et al., 2025). Existing evaluations

broadly span live-web and sandboxed environments. Live-web benchmarks include Mind2Web-Live (Pan et al.), Online-Mind2Web (Xue et al.), AssistantBench (Yoran et al., 2024), WebVoyager (He et al., 2024), Amazon-Bench (Zhang et al., 2026), Odysseys (Jang et al., 2026), and BrowserComp (Wei et al., 2025). These benchmarks expose agents to realistic and evolving web content. However, live evaluation is inherently vulnerable to website changes, bot detection, and non-reproducible interface states, and it becomes particularly problematic for consequential tasks that irreversibly modify real-world state. Sandboxed environments, such as WebArena (Zhou et al., 2024), VisualWebArena (Koh et al., 2024), REAL (Garg et al., 2025), SCUBA (Dai et al., 2026), WebChoreArena (Miyai et al., 2025), and OpenApps (Ullrich et al., 2026) mitigate these issues via realistic website replicas. While these environments enable controlled evaluation, they typically require substantial manual engineering and consequently lack variety. In Table 4 we categorize tasks by user intent: *Informational* tasks require retrieving requested information; *Navigational* tasks require reaching a specific page; *Transactional* tasks require modifying website state; and *Transactional Irreversible* tasks involve state-changing actions that cannot be safely or easily undone. This analysis shows that existing evaluations trade off environmental diversity, reproducibility, and support for consequential interaction. DIGIDECK targets this gap by automatically synthesizing browser-renderable environments and action-conditioned state transitions, allowing diverse tasks to be evaluated without operating on live services.

Benchmark	Tasks	Websites	Environment	Informational	Navigational	Transactional	Trans. Irreversible
Mind2Web Live	104	46	Live	73	6	25	7
Online Mind2Web	300	147	Live	230	21	49	7
AssistantBench	181	≈51	Live	181	0	0	0
WebVoyager	643	15	Live	628	0	15	7
Odysseys	200	57	Live	140	0	60	1
BrowserComp	1266	1	Live	1266	0	0	0
OpenApps	343	6	Sandbox	0	99	244	67
REAL	112	11	Sandbox	39	1	72	61
WebArena	812	5	Sandbox	405	17	390	80
SCUBA	300	1	Sandbox	25	0	275	6
VisualWebArena	910	3	Sandbox	259	233	418	164
DIGIDECK	1080	181	Synthetic	360	360	360	246

Table 4: Benchmark task types and environments.

World Models and Synthetic Web Environments. Synthetic simulations have a long history of applications in training and evaluating complex systems such as autonomous-vehicle testing (Dosovitskiy et al., 2017; Shah et al., 2017), and embodied agents (Xia et al., 2026; Yang et al., 2024; Deitke et al., 2022). To address the limitations of static evaluations and the risks of live-web testing, recent work has increasingly explored using generative models to simulate environments and their dynamics. One line of work represents environment transitions through code. Early work (Dainese et al., 2024) used Python programs to construct digital toy worlds, enabling precise planning for reinforcement learning agents. This idea has since been extended to more complex rule-based systems (Lehrach et al., 2025; Tang et al., 2024), where LLMs translate natural-language instructions and rules into verifiable simulation engines. In web settings, prior work (Gu et al.; Chae et al., 2025) has shown that LLMs can internally simulate and score possible future web states before committing to an action. Related efforts extend this direction to graphical user interfaces, training models to predict mobile-interface next states from actions using visualization code derived from real trajectories (Chae et al., 2026; Zheng et al., 2026; Luo et al., 2025). Most closely related to our work, Wang et al. (2026) and Wu et al. (2026) synthesize fully functional web environments by translating finite-state machines into front-end code with a deterministic state-driven runtime. However, this design constrains agents to closed-world environments governed by predefined state transitions, and its BFS-based exploration can yield relatively homogeneous trajectories with limited semantic diversity. DIGIDECK addresses these limitations by supporting more diverse, flexible, and realistic task environments. A closely related contemporary work (Yuan et al., 2026) leverages coding LLMs to generate complete websites. DIGIDECK differs in that it treats environment generation as an action-conditioned simulation, allowing evaluation beyond a fixed set of pre-specified transitions. More ambitious paradigms, such as Neural Computers (Zhuge et al.), propose unifying computation, memory, and I/O within a learned runtime state, instantiated through diffusion video models that roll out interface frames. Although this paradigm is promising, its current unpredictability makes code-rendered environments better suited for controlled, reproducible web-agent evaluation.

B DIGIDECK IMPLEMENTATION DETAILS

This appendix provides the prompt templates used throughout the DIGIDECK framework to instantiate the experiments reported in the paper.

B.1 COMMON SYSTEM PROMPT

Unless noted otherwise, DIGIDECK uses the following system message for scaffold generation, state generation, transition validation, and user simulation.

```
You are a helpful assistant with extensive knowledge of how the web
    works.
```

B.2 WEBSITE SCAFFOLD GENERATION

```
# Objective
You are an expert Information Architect and Web Designer. Your task
    is to
design a realistic, hierarchical website structure graph (scaffold)
    for
<WEBSITE_URL>.

# Target Tasks
The website structure MUST contain the necessary pages and components
     to
allow a user to complete the following tasks:
<TASK_INSTRUCTIONS>

# Architectural Constraints
* No Spoilers: The literal task instructions above MUST NOT appear
    anywhere
  in your output.
* Depth Requirement: The scaffold must span exactly <DEPTH> layers of
  navigational depth.
* Realistic Navigation (No Shortcuts): The structure must represent a
  real-world website layout. Do not create artificial shortcuts,
    direct
  links, or magic pages designed specifically to solve the target
    tasks.
* Required Interaction: To succeed at the tasks, a simulated user
    must be
  forced to realistically navigate through the full depth of the
    scaffold
  to find the solution.
* Component Details: Each page or major component in the scaffold
    must
  include a brief description of its specific purpose and
    functionality.
* Interaction Semantics: For every interactive component, describe
    both its
  user-visible trigger and its expected outcome in plain language.
    For
  example: Search button: submits the entered query and displays the
  matching products page. These descriptions will become stable
  interaction metadata in generated pages.

# Output Format
```

```
Respond ONLY with a valid JSON object. Do not include markdown
    formatting
blocks, conversational text, or explanations.

Structure your JSON exactly like this:
{
  website_url: <WEBSITE_URL>,
  total_depth: <DEPTH>,
  structure: [
    {
      name: Home Page,
      type: page,
      layer: 1,
      description: The main landing page containing global navigation
    .,
      children: [
        {
          name: Category Dropdown,
          type: component,
          layer: 2,
          description: Navigation component to select product
    categories.,
          children: []
        }
      ]
    }
  ]
}
```

For a matched intent triplet, <TASK_INSTRUCTIONS> contains the transactional task followed by the navigational and informational variants.

B.3 Initial-State Generation

```
# Objective
You are an expert Frontend Web Developer. Your task is to generate
    the
complete, renderable initial HTML state for the website <WEBSITE_URL>
     based
on the provided architectural scaffold.

# Scaffold
<WEBSITE_SCAFFOLD>

# Constraints & Generation Rules

## Core Behavior & State
* Top-Level View: The initial state must be a single page
    representing the
  root view (homepage) of the website, including all global
    navigation and
  primary components defined in the scaffold.
* Valid HTML: The output must be a complete, valid HTML5 document (
    starting
  with <!DOCTYPE html>) that can be rendered in a standard web
    browser.
  NEVER output a blank page.
* Authentication Default: In this initial state, assume the user is
    NOT
  logged in. Display standard Log In or Sign Up UI elements where
```

```
  appropriate.

## Visuals & Assets
* Images: Use publicly accessible URLs of real photographic or
    illustrated
  images. Do NOT use text-only placeholders, labeled rectangles, or ?
    text=
  parameters.
* Alt Text: Provide specific, highly descriptive alt text identifying
     the
  image's actual subject. Avoid generic terms such as banner or
  product image.
* Fallbacks: If a suitable public image cannot be found, provide a
  descriptive alt attribute with an intentionally broken/unavailable
    source
  so the configured image-generation fallback can create an
    appropriate
  visual.

## Interactivity & Links
* No External Navigation: Do not use real outgoing links. Use
  href=about:blank for any links or anchor tags that would navigate
    away
  from the current page.
* Event Handlers: Ensure any element that should trigger an action
    has an
  appropriate event handler or attribute to maintain basic frontend
  interactivity.
* Interaction Descriptors: Every interactive element (a, button, form
  control, and any element with a click handler, interactive role, or
  non-negative tabindex) MUST have a data-digideck-action attribute.
    Its
  value must be a concise, specific, imperative description of the
    expected
  result of interacting with that exact element, not merely its label
    .
  Include the relevant destination, state change, or submitted input
  context. Keep this attribute accurate when element behavior or
    state
  changes.

# Output Format
You must follow this exact two-block structure. Do not use tool calls
    . Do not
output any conversational text outside of these blocks.

```description
[A detailed natural language description explaining the layout,
    components
included from the scaffold, and the reasoning behind the initial
    design]
```

```html
[Complete, renderable HTML5 code]
```
```

If viewport validation rejects the generated page, the following suffix is appended and the request is retried.

```
# Viewport repair
Fix the previous HTML while preserving its content and intended
    design.
Return the complete corrected response in the required output format.

## Viewport feedback
<VIEWPORT_VALIDATION_FEEDBACK>

## Previous HTML
```html
<PREVIOUS_HTML>
```
```

B.4 GENERIC POLICY PROMPT

For policies using the common DIGIDECK tool interface, the following user prompt is sent with the current screenshot history. Private simulator metadata, including `data-digideck-action` descriptions is never revealed to the policy.

```
You are a helpful agent navigating web environments and completing
    all asked
tasks.
Task instruction: <TASK_INSTRUCTION>

Information provided by the user:
<USER_INFORMATION_OR_NONE>

Existing notes:
<NOTES_OR_NONE>

Recently completed browser actions, oldest to newest:
<ACTION_HISTORY_OR_NONE>

Current browser tabs (zero-based indexes):
<TAB_LIST_OR_NONE>

Select exactly one browser action tool call that best advances the
    task.
<ASK_USER_INSTRUCTION_IF_ENABLED>
If the task has already been completed, use the task_complete action.
     If you
need to provide a final response to the user, use the final_answer
    action and
include your response in the message field. Either action ends the
    task.
Use go_back or go_forward for simulated per-tab browser history.
    Strongly
prefer to interact with elements on the page rather than using go_to,
     unless
absolutely necessary or the task explicitly mentions a different URL.
Requirements that should be filters cannot be satisfied merely by
    entering
all requirements into search.
Use the screenshot and the screenshot-relative coordinates in the
    bounding
boxes; center_x and center_y are safe default interaction points.

Current bounding boxes:
<BOUNDING_BOXES_AS_JSON>
```

When user simulation is enabled, <ASK_USER_INSTRUCTION_IF_ENABLED> is replaced by:

```
When information or a preference from the user is necessary to
    proceed and
is not already available, use ask_user and put the question in the
    message
field. Do not guess required user information. Also do not ask the
    user to
perform any actions themselves.
```

Preliminary results showed that unless this section is included in the prompt, models had a high propensity to hallucinate user information despite being provided with the capability to ask a user for confirmation, and encouraged to do so in the action description.

If a proposed action fails validation, the following retry suffix is appended:

```
Your previous action was invalid. Regenerate exactly one corrected
browser_action tool call.
Validation feedback: <VALIDATION_ERROR_AND_PREVIOUS_TOOL_CALL>
```

The prompt is accompanied by an OpenAI-compatible `browser_action` tool. Its action enum is:

```
click, fill, type, press, hover, wait, go_back, go_forward, go_to,
new_tab, tab_focus, tab_close, ask_user, note, scroll_at, drag,
final_answer, task_complete
```

The `ask_user` action is omitted when user simulation is disabled. The tool schema advertises only fields applicable to these actions: `text`, `key`, `message`, `url`, `index`, source and target coordinates, scroll deltas, and timeout. Relative coordinates are constrained to $[0, 100]$. For local vLLM models without native tool parsing, the same action enum is placed in the prompt with this suffix:

```
Return the next action as one JSON object and no prose. The object
    must
contain action and only the applicable fields from: text, key,
    message, url,
index, x, y, target_x, target_y, delta_x, delta_y, timeout_ms. Valid
    action
values are: <ACTION_ENUM>.
```

B.5 Implementation Details

Table 5 summarizes the implementation and fixed environment settings used for the reported experiments.

B.6 Unified Browser Action Space

Table 6 gives the union of executable actions represented by the internal `BrowserAction` interface. The generic policy tool exposes the applicable subset directly; Qwen3-VL actions are translated into this representation by model-specific adapters. Coordinates are viewport-relative values in $[0, 100]$ in the reported experiments and are converted to CSS pixels immediately before Playwright execution. The internal `initialize` action is omitted because policies cannot select it.

Category	Setting	Implementation or value
Runtime	Software stack	Python, synchronous Playwright, and isolated headless Chromium contexts
Runtime	Model-client interfaces	OpenAI, Anthropic, OpenRouter, LiteLLM, and vLLM
Rendering	Viewport	1280×720 CSS pixels
Rendering	Action coordinates	Viewport-relative values normalized to $[0, 100]$
Episode	Maximum length	60 policy actions
Episode	Random seed	42
Generation	Scaffold depth	3 layers
Generation	Retry budget	3 attempts
Generation	Scaffold and successor-state model	`nvidia/GLM-5.2-NVFP4` through LiteLLM
Generation	Broken-image replacement	FLUX.2 Klein 4B
Policy context	Screenshot history	1 current screenshot
Policy context	Action history	10 actions; 1 for Qwen3-VL
Grounding	Maximum boxes	100 leaf interactive elements
State context	Description/HTML history	10 state descriptions and 0 historical HTML documents
Validation	Transition validator	Enabled; reuses the state-generation client when no separate model is configured
Interaction	User simulation	Enabled; uses the state-generation client
Evaluation	Terminal trajectory judge	Enabled; `GPT-5-mini`
Reproducibility	Initial artifacts	One shared scaffold and initial page per source task identifier across intents and policy models

Table 5: DIGIDECK implementation details and common experimental configuration.

Table 6: Unified DIGIDECK browser action space. Optional arguments are shown in parentheses.

Action	Arguments	Runtime effect
`click`	x, y, (`timeout_ms`)	Click the left mouse button at the target point.
`right_click`	x, y, (`timeout_ms`)	Click the right mouse button at the target point.
`middle_click`	x, y, (`timeout_ms`)	Click the middle mouse button at the target point.
`double_click`	x, y, (`timeout_ms`)	Double-click the left mouse button at the target point.
`fill`	x, y, `text`, (`key`)	Focus a field, select its existing content, insert replacement text, optionally press a key, and blur the field.
`type`	x, y, `text`, (`key`)	Focus a field, append text without clearing its value, and optionally press a key.
`type_focused`	`text`	Type text into the element that already has keyboard focus.
`press`	x, y, `key`	Focus the target element and press one Playwright keyboard key.
`press_focused`	`key` or `keys`, (`timeout_ms`)	Press one key or hold and release a sequence of keys on the currently focused element.
`hover`	x, y, (`timeout_ms`)	Move the pointer to the target point without clicking.
`wait`	(`timeout_ms`)	Pause execution; the default duration is 5,000 ms.
`scroll_at`	x, y, `delta_y`, (`delta_x`)	Move the pointer to a location and apply a mouse-wheel scroll. Positive vertical deltas scroll down.
`drag`	x, y, `target_x`, `target_y`	Hold the left mouse button and drag from the source to the destination in ten pointer movements.
`go_back`	None	Move backward one entry in the active tab's simulated HTML history.
`go_forward`	None	Move forward one entry in the active tab's simulated HTML history.
`go_to`	`url`	Request a generated navigation to an absolute URL.
`new_tab`	None	Open and focus a new tab containing the blank simulated start page.
`tab_focus`	`index`	Focus the open tab at the specified zero-based index.
`tab_close`	`index`	Close the indexed tab and focus a remaining tab; the only open tab cannot be closed.
`ask_user`	`message`	Ask the user simulator for missing information and append its answer to policy-visible user information.
`note`	`message`	Save policy working information without changing the rendered page.
`final_answer`	`message`	Terminate the episode and store a textual response for the user.
`task_complete`	(`status`)	Terminate without a textual response; `status` may mark success or policy-declared failure.

The generic `browser_action` tool directly advertises `click`, `fill`, `type`, `press`, `hover`, `wait`, history and tab actions, `ask_user`, `note`, `scroll_at`, `drag`, and the two terminal actions. The right-, middle-, and double-click variants and focused keyboard variants are produced by the Qwen3-VL adapters when those native interfaces request them.

B.7 QWEN3-VL COMPUTER-USE PROMPT

Qwen3-VL uses its native cookbook message structure instead of the generic policy prompt. Its system message is:

```
You are a helpful assistant.

# Tools

You may call one or more functions to assist with the user query.

You are provided with function signatures within <tools></tools> XML
    tags:
<tools>
<SERIALIZED_FUNCTION_DEFINITIONS>
</tools>

For each function call, return a json object with function name and
    arguments
within <tool_call></tool_call> XML tags:
<tool_call>
{name: <function-name>, arguments: <args-json-object>}
</tool_call>
```

The serialized `computer_use` function is the Qwen3-VL cookbook schema. It describes a 1000×1000 grounding space and exposes the following actions:

```
key, type, mouse_move, left_click, left_click_drag, right_click,
middle_click, double_click, triple_click, scroll, hscroll, wait,
terminate, answer
```

Its arguments are `action`, `keys`, `text`, `coordinate`, `pixels`, `time`, and `status`. When user simulation is enabled, DIGIDECK adds an `ask_user_info` function with one required string field, `message`, and this description:

```
Ask the user for missing information or a preference required to
    complete
the task. Do not ask the user to perform browser actions.
```

The first user message contains the current screenshot and the task instruction. If available, it appends:

```
Information provided by the user in response to earlier ask_user
    calls:
<USER_INFORMATION_AS_JSON>
```

Subsequent requests retain the configured screenshot/response pairs and append the new screenshot. After an invalid tool call, DIGIDECK appends:

```
The previous tool call was invalid and was not executed. Correct it
    and
return a valid tool call. <VALIDATION_FEEDBACK>
```

B.8 NEXT-STATE GENERATION

The next-state prompt is assembled from the following blocks. Optional blocks are included only when their corresponding data is available.

```
# Objective
Predict the complete next HTML state of <WEBSITE_URL> after the
    simulated
browser action.

# Website Scaffold
<WEBSITE_SCAFFOLD>

# Simulated Action
<ACTION_METADATA_AS_JSON>

# Coordinate Semantics
The simulated action and bounding-box x, y, width, and height values
    use
RELATIVE viewport coordinates normalized from 0 to 100. The viewport
    is
<WIDTH> CSS pixels wide by <HEIGHT> CSS pixels high.
Convert them to viewport CSS pixels as follows:
* pixel_x = relative_x / 100 * viewport_width
* pixel_y = relative_y / 100 * viewport_height
* pixel_width = relative_width / 100 * viewport_width
* pixel_height = relative_height / 100 * viewport_height
The origin (0, 0) is the viewport's top-left and (100, 100) is its
bottom-right. Do not interpret these values as CSS pixels.

# Bounding boxes overlapping with the action coordinates
All captured foreground boxes that contain the action point are
    included.
Nested boxes may describe the same interaction target at different
    DOM
levels; use their combined text, metadata, and interaction
    descriptions to
infer the intended action.
<OVERLAPPING_BOUNDING_BOXES_AS_JSON>

<OPTIONAL_HISTORICAL_STATE_BLOCKS>
<OPTIONAL_VIEWPORT_FEEDBACK>
<OPTIONAL_PREVIOUS_GENERATION_FAILURE>

# Previous HTML
```html
<PREVIOUS_HTML>
```

Constraints & Generation Rules

Core Behavior & State
* Consistency: For identical actions from the same state, generate
    the
  identical next page. Preserve relevant content/styling from the
    previous
  page.
* No Blockers: Do not introduce unrecoverable blockers such as real
    CAPTCHAs
  or email verification. If auth, payment, or consent is required,
    simulate
  the corresponding interactive page.
```

```
* No External Links: Do not use real outgoing links. Use href=about:
    blank
  for navigation away from the current domain.
* Invalid/No-Op Actions: If no element was interacted with, the
    action is
  invalid, or its correct result leaves both HTML and scroll position
  unchanged, use Option D. Never reproduce the Previous HTML in an
    html
  block merely to express that nothing changed. Explain the reason in
     the
  description block.

## Visuals & Assets
* Images: Reuse existing publicly accessible image URLs/alt text. If
    new
  images are needed, use real photographic/illustrated public URLs.
* Forbidden Assets: Do NOT use emojis as UI elements. Do NOT use text
    -only
  placeholders, labeled rectangles, or ?text= parameters.
* Alt Text: Use highly descriptive alt text. Avoid generic terms. If
    a real
  image cannot be found, provide descriptive alt text with an
    intentionally
  broken source so fallbacks can generate it.
* Realism: Include redundant but reasonable UI components that would
  realistically exist in this environment.

## Interactivity & Authentication
* Assume Functional Intent: Even if an element lacks a click handler
    (or has
  a dummy handler) in the Previous HTML, assume it functions
    correctly. If
  it logically should trigger a state change, reflect that in the
    next state.
* Interaction Descriptors: Every interactive element MUST have an
    accurate
  data-digideck-action attribute describing its expected result. Use
    the
  overlapping bounding box's interaction_description as authoritative
  evidence. Preserve descriptors for unchanged elements and update
    them when
  behavior changes.
* Session State: Track and preserve auth states. If an action
    requires login,
  simulate the login prompt. If already logged in, show personalized
    content.
* Search/Filtering/Sorting: If the action is a search or filter,
    ensure the
  next state reflects the expected results and updated controls. If
    no
  results are found, show a realistic no-results message.
* Clicks Should Trigger State Changes: If an interactive element is
    clicked,
  ensure the next state reflects the expected change. Do not leave
    the page
  unchanged unless no actionable element was actually activated.
* Existing JavaScript behavior is only evidence, not authoritative
    truth.
  Infer intent primarily from the action, interaction_description,
    visible
  UI, scaffold, and prior state. Preserve an observed JavaScript
    result only
  when it is consistent with that intent.
```

````
# Output Format
You MUST NEVER output a blank page. ALWAYS respond with the next
    state
representation. Do not use tool calls. Do not use exclamation marks.

Step 1: Always begin with:
```description
<DETAILED_STATE_AND_TRANSITION_REASONING>
```

Step 2: Choose exactly one available option.

Option A - Complete HTML:
```html
<COMPLETE_RENDERABLE_HTML_DOCUMENT>
```

Option B - Targeted changes:
```changes
<<<<<<< SEARCH
<EXACT_UNIQUE_SNIPPET_FROM_PREVIOUS_HTML>
=======
<REPLACEMENT_HTML>
>>>>>>> REPLACE
```
Every SEARCH block must be an exact, contiguous, verbatim match from
    the
Previous HTML. Keep SEARCH snippets minimal but unique.

Option C - Restore an exact cached historical state:
```restore_state
{state_id: state_002}
```
Use only a listed immutable state that exactly matches the intended
    result.
Do not reproduce or modify its HTML. A restore response cannot
    contain html,
changes, no_change, or scroll blocks.

Option D - No change:
```no_change
no change
```
Use only when the correct result has exactly the same HTML and scroll
position. A no-change response cannot contain html, changes,
    restore_state,
or scroll blocks.

Step 3: Optionally append one scroll block after an HTML or changes
    block.
Use either exact non-negative window coordinates:
```scroll
{window_x: 0, window_y: 640}
```
or a stable target selector:
```scroll
{target_selector: #search-results, block: start,
 inline: nearest}
```
Use exactly one form and omit the block when scrolling should not
    change.
````

For a `go_to` action, the following instruction is inserted after the action:

```
Navigation Instruction: The go_to action requests a simulated
    navigation.
Generate the destination screen for exactly this URL: <REQUESTED_URL
    >. Do
not retain the previous page merely because no element/coordinates
    were
provided.
```

When cached historical states are available, each state identifier and description is listed, with HTML included only according to the configured history limit. On a generation or rendering failure, the error is inserted under `# Previous generation failure`. When full-page recovery is required, the following override is appended:

```
# Mandatory Full-Page Recovery Override
The previous generated output could not be safely applied or did not
    render
as a complete styled website. Return Option A when the state must
    change: one
complete, standalone, fully styled and renderable HTML document in an
     html
code block. It must contain <html>, <head>, and <body> elements and
    preserve
the website's established design and relevant state. Option C remains
available when the intended result exactly matches a listed cached
historical state; return its ID rather than rebuilding it. Return
    Option D
only when the correct result is exactly the Previous HTML at the same
     scroll
position.

Do not return a changes block. Do not return an HTML fragment, plain
    text,
Markdown, an explanation in place of HTML, or an unstyled document.
    Do not
reproduce the Previous HTML in an html block to report a no-op. Do
    not return
an unlisted or modified cached state. For a real change that does not
     exactly
match a listed cached state, reconstruct the full page from the
    Previous
HTML, scaffold, action, and failure feedback above.
```

B.9 Transition-Validation Prompt

```
# Objective
You are an expert web testing AI. Your task is to determine whether
    the
observed HTML transition is a plausible, sufficient, and correct
    result of
the specified browser action.

## Evaluation Criteria
* Focus on the Transition: Judge the actual change between the
    previous and
```

```
  current HTML, not whether the action itself was a good strategic
    choice.
* Strict Sufficiency: The HTML change MUST completely accomplish the
    action's
  expected outcome. Plausible but partial progress is insufficient
    and must
  be rejected.
* Assume Simulated Environment: The website might use dummy event
    handlers.
  Evaluate based on expected real-world behavior. If an element
    should
  trigger a state change, the new HTML must reflect that change.
* Clicks Should Trigger State Changes: Ensure that a click produces
    the
  expected modal, navigation, content update, sorting, or other
    effect.
* When in Doubt, Reject: If you are unsure whether the transition
    fully
  satisfies the action, reject it.
* Clicks on e-store items: Ensure that the next state reflects the
    expected
  product-page, cart, or other commerce transition.

## Output Format
Respond strictly in JSON matching the provided schema. Provide your
    reason
first and then the final corresponds_to_action Boolean.

---
# Browser Action
<ACTION_METADATA_AS_JSON>

# Coordinate Semantics
<ABSOLUTE_OR_RELATIVE_COORDINATE_DESCRIPTION>

<OPTIONAL_ACTION_BOUNDING_BOX>
<OPTIONAL_DRAG_SOURCE_AND_DESTINATION_BOXES>

---
# State Data
## HTML Diff
```diff
<UNIFIED_HTML_DIFF>
```

Previous HTML
```html
<PREVIOUS_HTML>
```

Current HTML
```html
<CURRENT_HTML>
```
```

The request is constrained by this JSON output schema:

```
{
  type: object,
  properties: {
    reason: {type: string},
```

```
    corresponds_to_action: {type: boolean}
  },
  required: [reason, corresponds_to_action],
  additionalProperties: false
}
```

B.10 SIMULATED-USER PROMPT

```
You are simulating the user who requested the task below. Answer the
    agent's
question naturally and concisely with the information necessary to
    continue.
Stay consistent with earlier answers. Do not explain that you are a
simulator. If the requested detail is not specified, choose a
    reasonable
preference as the user.

Answer all questions, and never say that you cannot answer. Never
    assume you
are logged in to any website or service. If the agent asks you, tell
    them to
infer it from the browser state.

Never offer to do something yourself. If the agent asks you, always
    tell the
agent to do it and provide the necessary information to do it,
    including
usernames, passwords, and any other required credentials or personal
    and
payment information.

If the agent asks you to perform an action, always respond with a
    message
that instructs the agent to perform the action and provide any
    necessary
information for the agent to do so. Never perform any actions
    yourself.

Task instruction:
<TASK_INSTRUCTION>

Earlier user information:
<QUESTION_ANSWER_HISTORY_AS_JSON>

Agent question:
<AGENT_QUESTION>

Return only the user's answer.
```

B.11 TRAJECTORY-EVALUATION PROMPTS

The evaluator uses three dedicated system prompts. The key-point extraction system prompt is:

```
You are an expert tasked with analyzing a task to identify the key
    points
explicitly stated in its description.
```

```
Extract only critical elements, conditions, or steps stated by the
    user. Do
not infer additional requirements. Comparative words such as best,
highest, cheapest, latest, most recent, lowest, closest,
highest-rated, largest, and newest require use of the relevant sort
function and should be represented as a sorting key point.

Respond with:
Key Points:
1. <key point>
2. <key point>
```

Its user message is:

```
Task: <TASK_INSTRUCTION_AND_ADDITIONAL_INSTRUCTIONS>
```

The screenshot-relevance system prompt is:

```
You are an expert evaluator determining whether a webpage screenshot
    contains
necessary steps or evidence for completing a task.

Describe the visible page and assess whether it shows relevant
    actions,
progress, applied filters, results, or other indispensable evidence.
    Partial
but useful evidence should receive partial credit.

Respond exactly in this format:
Reasoning: <specific observations and explanation>
Score: <integer from 1 to 5>

Scores:
1 = no relevant evidence
2 = minimal or ambiguous evidence
3 = some relevant evidence, but incomplete
4 = important and highly relevant evidence
5 = clear evidence crucial to task completion
```

Each candidate screenshot is attached to this user message:

```
Task: <TASK>

Key Points for Task Completion:
<EXTRACTED_KEY_POINTS>

Evaluate the attached webpage snapshot.
```

The final-judgment system prompt is:

```
You are an expert evaluating whether a web navigation agent completed
     the
```

```
user's task. Check the action history and selected trajectory
    screenshots
against every key point.

Evaluation criteria:
1. Required filtered results must be visibly correct. A missing
    selection,
   confirmation, or visible effect is a failure.
2. Superlatives and comparisons such as best, highest, cheapest,
    latest,
   lowest, closest, highest-rated, largest, or newest must be handled
     through
   sorting.
3. Requirements that should be filters cannot be satisfied merely by
   entering all requirements into search.
4. Required money, year, bed, bathroom, or other numeric ranges must
    match
   exactly; a broader or narrower range is a failure.
5. Tasks requiring submission or displayed results succeed only after
     that
   step.
6. Empty results can still be successful when the required action was
   performed correctly.
7. If the page already shows every available item, filtering is
    unnecessary
   if the agent correctly selects an item meeting the requirement.
8. If the policy model uses information that was not provided in the
    task
   description, or provided by the user in the user information, it
    is a
   failure.

Respond in exactly two lines:
Thoughts: <concise reasoning checking every key point>
Status: success or failure
```

The final user message is:

```
User Task: <TASK>

Key Points:
<EXTRACTED_KEY_POINTS>

Action History:
1. <SERIALIZED_ACTION>
2. <SERIALIZED_ACTION>
...

<OPTIONAL_USER_INFORMATION_AS_JSON>

Potentially important trajectory snapshots and their reasons:
1. <IMAGE_JUDGE_REASONING>
2. <IMAGE_JUDGE_REASONING>
...
```

The selected screenshots are attached to this request in the same order as their reasons.

#	Website	Original	Transactional	Informational	Navigational
1	qatarairways.com	Find the weight of baggage allowance for economy class on Qatar Airways.	Log in to Qatar Airways and add an extra baggage to my upcoming flight reservation.	Find the weight of baggage allowance for economy class on Qatar Airways.	Open the Qatar Airways baggage allowance page for economy class.
2	us.megabus.com	Find out what to do when I lose an item on a bus on us.megabus.	Submit a lost item report on us.megabus.	Find out what to do when I lose an item on a bus on us.megabus.	Open the us.megabus website and navigate to the 'Lost & Found' section.
3	new.mta.info	Find the list of neighborhood maps for Brooklyn on new.mta.info.	Download a neighborhood map for Brooklyn on new.mta.info.	Find the list of neighborhood maps for Brooklyn on new.mta.info and report the one with the most stations.	Find the list of neighborhood maps for Brooklyn on new.mta.info.
4	imdb.com	Show crazy credits for the movie " Prometheus" on IMDb.	Log in to IMDb and submit a review for the movie "Prometheus," including your thoughts on the crazy credits.	Find the crazy credits for the movie "Prometheus" on IMDb and summarize the unique or interesting elements of the credits.	Show crazy credits for the movie " Prometheus" on IMDb.
5	justice.gov	Browse the list of Civil Division forms.	File a complaint using the Civil Division form.	Find the Civil Division form for filing a complaint and summarize the required information to complete the form.	Browse the list of Civil Division forms.
6	rottentomatoes.com	View the list of the Most Popular TV on rotten tomatoes.	Log in to rotten tomatoes and submit a review for the Most Popular TV show with the highest rating.	Find the list of the Most Popular TV on rotten tomatoes and return the one with the highest rating.	View the list of the Most Popular TV on rotten tomatoes.

Table 7: Six examples of the transactional, informational, and navigational transformations applied to the same original instructions. Each row preserves the website and underlying topic while changing the requested outcome according to the target intent.

Intent	Navigation	Controls	Actionable	Visuals	Complex visuals	Visible words
Navigational	78.56 ± 44.516	28.87 ± 15.417	103.71 ± 45.448	17.42 ± 11.611	0.28 ± 1.123	622.78 ± 284.321
Informational	78.41 ± 44.603	30.56 ± 17.606	104.52 ± 46.186	17.96 ± 11.525	0.26 ± 0.966	732.54 ± 403.156
Transactional	86.87 ± 48.870	36.88 ± 17.807	117.58 ± 53.077	18.78 ± 13.020	0.25 ± 0.973	715.44 ± 314.301

Table 8: Raw complexity measurements by intent. Each entry is the mean trajectory maximum μ plus or minus the standard deviation σ over 100 trajectories.

B.12 Image-Replacement Input

Image replacement does not add a separate natural-language wrapper. When an image fails to load, the element's descriptive alt-text is passed directly to the configured text-to-image model as its prompt. The model name, provider, prompt, seed, and generated result are included in the image cache key or metadata.

C Web Intent Triplets

Table 7 illustrates qualitative examples of intent transformation. The core context is maintained across all three intents, with navigational tasks focusing on finding a particular page, informational on extracting some relevant content, and transactional on completing an action that requires a state-change that could have real-life consequences that are difficult to reverse.

D Task Complexity Analysis

We analyze the complexity of the generated task environments using deterministic proxies computed from the HTML states visited during each trajectory adapted from contemporary work (Yuan et al., 2026). The reported analysis covers 100 matched triplets of tasks across the three Broder intents, yielding a total of 300 trajectories.

Each trajectory is analyzed independently. For every rendered state, an HTML parser counts navigation options, interaction controls, actionable elements, visual elements, complex visual elements, and visible words.

Table 8 reports the mean and standard deviation for all six raw complexity metrics, also illustrated in the main paper in Figure 6. For each trajectory, we aggregate state-level complexity by taking the

maximum value of each metric across all visited states, since trajectories may traverse pages with substantially different levels of complexity. The six raw complexity metrics used in our analysis are computed as follows:

- **Navigation options**: the number of linked anchors or areas, `option` elements, and elements with a link, menu-item, option, tab, or tree-item ARIA role.
- **Interaction controls**: the number of non-hidden inputs, buttons, selects, text areas, editable elements, and elements with an interaction ARIA role, such as checkbox, combobox, radio, slider, switch, or textbox.
- **Actionable elements**: navigation options plus interaction controls on the same page. We compute this page-level sum before selecting the trajectory maximum; it is not the sum of the separate trajectory maxima.
- **Visual elements**: the number of `img`, `svg`, `video`, `canvas`, and `iframe` elements.
- **Complex visual elements**: the subset of visual elements represented by canvases or iframes, or described in accessible metadata as charts, diagrams, graphs, heatmaps, maps, plots, floor plans, or visualizations.
- **Visible words**: the number of word tokens in visible HTML text after excluding script, style, template, and `noscript` content.