ContractSkill: Repairable Contract-Based Skills for Multimodal Web Agents

Zijian Lu
School of Communication and Information Engineering, Nanjing University of Posts and Telecommunications
Nanjing, China
18818732360@163.com

Yiping Zuo*
College of Computer, Nanjing University of Posts and Telecommunications
Nanjing, China
zuoyiping@njupt.edu.cn

Yupeng Nie
School of Communication and Information Engineering, Nanjing University of Posts and Telecommunications
Nanjing, China
18094228550@163.com

Xin He
College of Computer, Nanjing University of Posts and Telecommunications
Nanjing, China
xhe@njupt.edu.cn

Weibei Fan
College of Computer, Nanjing University of Posts and Telecommunications
Nanjing, China
wbfan@njupt.edu.cn

Chen Dai
College of Computer, Nanjing University of Posts and Telecommunications
Nanjing, China
daichen@njupt.edu.cn

Abstract

Despite rapid progress in multimodal GUI agents, reusable skill acquisition remains difficult because on-demand generated skills often leave action semantics, state assumptions, and success criteria implicit. This makes them brittle to execution errors, hard to verify, and difficult to repair. We present ContractSkill, a framework that converts a draft skill into a contracted executable artifact with explicit preconditions, step specifications, postconditions, recovery rules, and termination checks. This representation enables deterministic verification, step-level fault localization, and minimal patch-based repair, turning skill refinement into localized editing rather than full regeneration. Experiments on VisualWebArena and MiniWoB with GLM-4.6V and Qwen3.5-Plus show that ContractSkill improves self-generated skills from 9.4% and 10.9% to 28.1% and 37.5% on VisualWebArena, and from 66.5% and 60.5% to 77.5% and 81.0% on MiniWoB. Repaired artifacts also transfer across models, improving the target model's self-generated-skill baseline by up to 47.8 points and 12.8 points on the two benchmarks, respectively. These results suggest that agent skills are better treated as explicit procedural artifacts that can be verified, repaired, and shared across models.

CCS Concepts

• **Computing methodologies** → **Intelligent agents**; • **Human-centered computing** → *Graphical user interfaces*; • **Software and its engineering** → *Automatic programming*.

Keywords

multimodal web agents, skill, deterministic verifier, program repair, cross-model transfer

1 Introduction

Multimodal web agents now operate in increasingly realistic browser environments, where a model must jointly interpret screenshots, structured page information, and natural-language goals while choosing actions across long interaction horizons. VisualWebArena has made this setting concrete and measurable for visually grounded web interaction, and it highlights both the promise and the brittleness of current systems [8]. The problem is inherently multimedia-centric because success depends on coordinating visual grounding, interface semantics, and sequential decision making in a shared action loop.

To address long-horizon execution, many recent agents rely on self-generated procedural guidance. In mobile and web settings, systems often produce a textual skill, manual, or reflection trace before or during action execution [2, 20, 29]. These artifacts help, but they are still awkward knowledge objects. They are usually generated on demand, tailored to a single run, and expressed as loosely structured text. As a result, they behave less like durable skills and more like one-off procedural drafts.

Recent benchmark evidence reinforces this concern. SkillsBench reports that self-generated skills provide negligible or negative benefit on average across diverse tasks, which suggests that current models do not yet reliably author high-quality procedural knowledge for later reuse [10].

This creates a practical failure mode. Self-generated skills usually do not specify what must hold before a step, what concrete page evidence should hold after the step, or what recovery policy should be applied when the page state deviates from expectation. Once execution fails, the system often falls back to rewriting the entire skill description or regenerating a new trajectory. This strategy may repair the immediate run, but it does not give the agent a clean, externally maintainable knowledge object.

Existing methods only partially solve this problem. Reflection-based or self-repair methods primarily update the *current episode*. They revise action choices, reasoning traces, or transient plans based on observed feedback [17, 20]. Skill-bank style work, in contrast, focuses on larger repositories of reusable knowledge, often in open-ended environments [3, 19]. Our target is different from both. We focus on a *single explicit skill artifact* for multimodal web

*Corresponding author.

interaction and ask how to make it executable, verifiable, locally repairable, and consumable by another model.

We therefore propose ContractSkill, a contract-based framework for repairing skill artifacts in multimodal web agents. Starting from a model-generated draft skill, ContractSkill compiles the draft into an executable artifact with preconditions, step objects, postconditions, recovery policies, and termination checks. A deterministic verifier then evaluates each step against page-state evidence and returns a structured error code together with the failing step index. Instead of rewriting the whole skill, ContractSkill applies one or a few minimal patch operators that change only the relevant selector, precondition, postcondition, recovery rule, or action argument. The repaired artifact can be reused by the source model and, importantly, consumed by a target model that did not generate the skill in the first place.

We summarize the primary contributions of this paper below.

- We introduce a contract-based representation that compiles a self-generated textual skill into a repairable skill artifact with explicit execution semantics.
- We design a verifier-guided repair framework that combines deterministic page-state checking, step-level fault localization, and minimal patch operators.
- We formulate a cross-model skill transfer setting in which a repaired artifact produced by one model can be executed by another, showing that repaired AI-generated skills can become portable procedural assets rather than model-specific prompts.

2 Related Work

2.1 Multimodal Web and GUI Agents

Web-agent benchmarks span a spectrum from controlled browser tasks to realistic multimodal web workflows. Early environments such as World of Bits and MiniWoB established reproducible settings for studying fundamental web interaction skills, while later platforms such as WebArena moved toward longer-horizon and more realistic web execution [11, 16, 31]. VisualWebArena further extends this line to visually grounded web tasks that require agents to interpret image-text inputs and act on realistic websites [8]. Together, MiniWoB and VisualWebArena provide complementary evaluation settings, with MiniWoB emphasizing controlled and repeatable interaction patterns and VisualWebArena emphasizing realistic multimodal web interaction.

2.2 Skill Artifacts and Procedural Knowledge

Recent work increasingly treats agent skills as explicit and portable procedural knowledge rather than transient prompting context. Agent Skills for Large Language Models surveys this abstraction layer, SkillsBench argues that skill quality and transferability should be evaluated directly, and analysis of single-agent skill libraries studies when external skills reduce coordination cost and when they fail [9, 10, 26].

Closer to our setting, Agent Workflow Memory stores reusable textual workflows, ASI induces and verifies executable programmatic skills online, and PolySkill separates abstract goals from concrete implementations for cross-site reuse [21, 22, 28]. SkillWeaver discovers website-specific APIs through iterative self-improvement, Hierarchical Memory Tree organizes web experience with intent-, stage-, and action-level memory plus explicit pre/post conditions, and RAG-GUI retrieves external GUI guidelines at inference time [18, 25, 30]. These works reinforce the value of structured procedural knowledge, but our focus is narrower. We study a *single externalized multimodal web skill artifact* with deterministic step-level diagnosis, minimal post-hoc repair, and cross-model reuse.

2.3 Self-Repair and Reflection in Agents

Recent work on self-improvement and reflection studies how agents iteratively refine behavior through memory, reflection, and evolution. ParamMem augments language agents with parametric reflective memory to increase the diversity and usefulness of self-reflection signals across tasks [27]. Group-Evolving Agents explores open-ended self-improvement through experience sharing and repeated evolution at the group level [23]. Despite these advances, the target of refinement in this line of work remains primarily the agent's current behavior, trajectory, or internal update process. By contrast, our work repairs the externalized skill artifact itself. This difference matters because the repaired artifact can persist as a reusable object across episodes and can also be consumed by another model.

2.4 Program Repair and Verifier-Guided Optimization

Program repair offers a useful methodological lens for our setting because it emphasizes three principles that are also central here, namely localizing failures, constraining the edit space, and using a verifier as the acceptance oracle. Classic automatic program repair spans generate-and-validate, template-based, semantics-based, and learning-guided methods, as exemplified by GenProg, PAR, SemFix, DirectFix, Angelix, and Prophet [6, 7, 12–15]. More broadly, verifier-guided refinement frameworks such as counterexample-guided abstraction refinement show how candidate solutions can be iteratively falsified and improved through structured feedback [4]. Although our setting does not repair source code directly, ContractSkill follows the same discipline. It treats a skill as a program-like artifact with contracts, uses deterministic checks to identify the failing component, and applies minimal local edits rather than unconstrained rewrites. This perspective brings verifier-guided repair principles into multimodal web agent skills.

3 Problem Definition

3.1 Environment

We consider a multimodal web environment, following recent execution-based agent benchmarks such as VisualWebArena, WorkArena, and OSWorld [5, 8, 24], in which an agent receives a natural-language task instruction x. At step t, the environment provides an observation $o_t = (I_t, D_t)$, where I_t is the current page screenshot and D_t is a structured page summary, such as a DOM abstraction, accessibility summary, or textual page state description. The agent outputs an action $a_t \in \mathcal{A}$, where $\mathcal{A}$ includes operations such as `click`, `type`, `scroll`, and `select`. The environment then transitions to a new observation o_{t+1}.

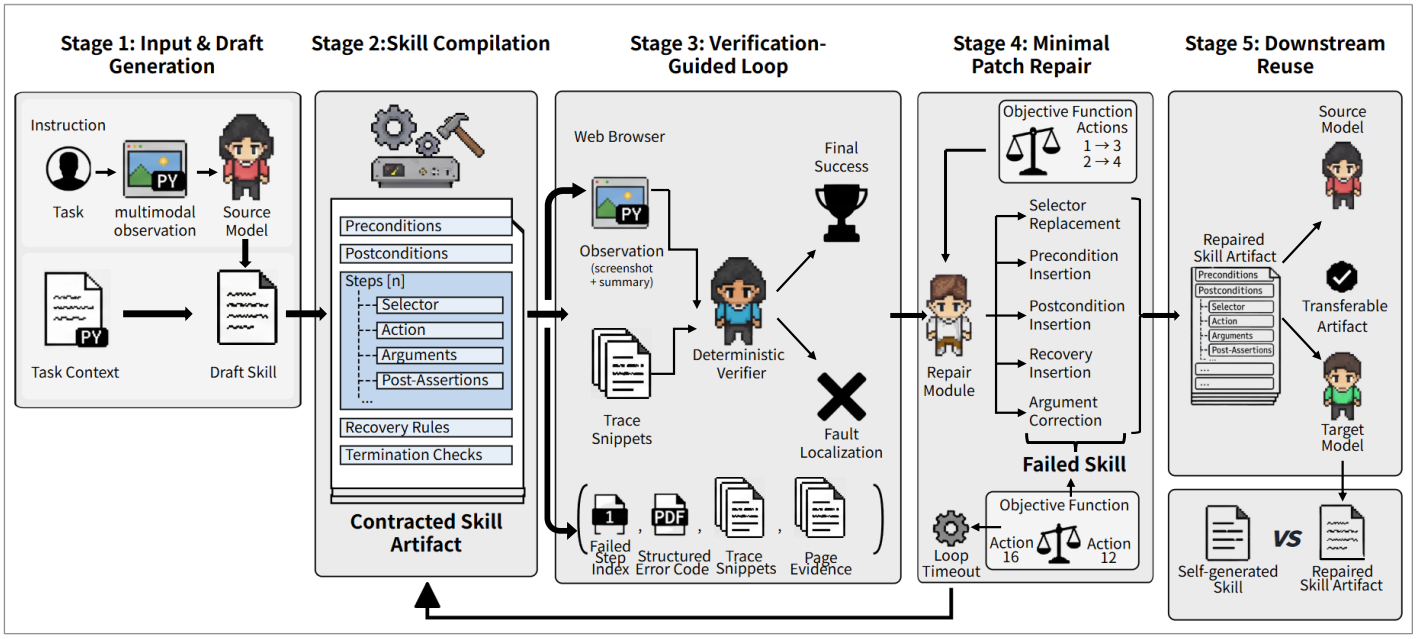

Figure 1: Overview of ContractSkill. The pipeline proceeds through five stages. It starts with input and draft generation, then compiles the draft into a contracted artifact, performs verification-guided execution with deterministic fault localization, applies minimal patch repair under an explicit objective, and finally reuses the repaired artifact with both the source model and a target model.

Task success is judged by a deterministic verifier V. Unlike free-form language judgments, V checks final and intermediate page states using reproducible signals, such as URL patterns, DOM matches, text presence, form values, or page-state transitions. This design is consistent with evaluation practice in reproducible web-agent benchmarks, which ground success in observable page evidence rather than subjective model preferences [5, 8, 31]. This setting lets us reason about failure in programmatic terms rather than relying only on subjective model reflection.

3.2 Draft Skill

Given instruction x and an initial observation, a source model first generates a textual draft skill s_0. This draft is a weak procedural object because it contains useful action intent but is not yet executable in a controlled or verifiable way. In particular, s_0 may omit hidden state assumptions, under-specify selectors, conflate goals with actions, or fail to state what evidence marks step completion. This pattern is common in self-generated manuals, mobile skills, and benchmarked skill artifacts [2, 10, 29]. We therefore treat the draft as a starting point rather than as the final reusable skill.

3.3 Contracted Skill Artifact

The core object in this paper is the contracted skill artifact s. We formalize it as

$$s = (g, P, U, Q, R, T), \qquad U = \{u_j\}_{j=1}^{m}, \tag{1}$$

where g is the goal, P denotes preconditions, U is the ordered step list, Q denotes postconditions, R denotes recovery rules, and T denotes termination checks. Each step is itself a structured object,

$$u_j = (\mathrm{sel}_j, \mathrm{act}_j, \mathrm{arg}_j, \Pi_j), \tag{2}$$

where sel_j is the selector, act_j is the action type, arg_j denotes optional arguments, and Π_j is the set of post-assertions for step j. This representation makes the skill executable at the step level and gives the verifier explicit hooks for diagnosis. It also makes the skill closer in spirit to reusable external knowledge objects, such as interaction manuals and structured prior knowledge stores, than to a one-off prompt fragment [2, 3, 19].

3.4 Objective

Starting from draft skill s_0, we search over a local neighborhood of candidate repairs and seek an improved artifact s^* that balances success, cost, and edit size according to the following objective.

$$s^* = \arg \max_{s \in \mathcal{N}_K(s_0)} \mathrm{Succ}(s) - \lambda \mathrm{Cost}(s) - \gamma \mathrm{Edit}(s, s_0). \tag{3}$$

Here $\mathrm{Succ}(s)$ denotes task success under the verifier, $\mathrm{Cost}(s)$ captures execution cost such as action count or tool calls, and $\mathrm{Edit}(s, s_0)$ penalizes unnecessarily large modifications. This objective reflects the desired behavior of a repair system, which is to improve the skill while preserving as much of the original procedural content as possible. The edit penalty is deliberately aligned with minimal-change preferences that have been useful in program repair.

Table 1: Structured error codes used by the deterministic verifier.

Error code	Meaning
`NOT_FOUND`	The specified selector does not match a valid page element.
`WRONG_STATE`	The current page state violates a step precondition.
`ASSERT_FAIL`	The action completes, but the expected postcondition is not satisfied.
`LOOP_TIMEOUT`	The agent repeats unproductive actions or exceeds the step budget.
`INPUT_INVALID`	The action arguments do not satisfy page-side constraints.

4 ContractSkill

4.1 Skill Artifact Schema

ContractSkill represents each skill as an explicit artifact with the following fields, namely `skill_name`, `goal`, `preconditions`, `steps`, `postconditions`, `recovery`, and `terminate`. Each step has four mandatory elements, namely `selector`, `action`, `args`, and `post_assertion`. Compared with free-form text, this schema exposes the exact operational assumptions behind the skill. The schema is designed to externalize procedures as reusable objects, which is also the motivating direction behind manual construction and page-level knowledge extraction in recent agent systems.

4.2 Deterministic Verifier

We distinguish two verification levels. A *final verifier* determines whether the full task has succeeded, and a *step verifier* checks whether each action was executed under valid state assumptions and produced the expected page evidence. Let $\mathcal{T}$ denote the execution trace together with the page-state trace. We write the verifier as

$$\mathcal{V}(s, \mathcal{T}) = \begin{cases} 1, & \text{if the task succeeds,} \\ (0, i, e, \tau_i), & \text{otherwise,} \end{cases} \tag{4}$$

where i is the failed step index, e is a structured error code, and τ_i is the local trace evidence used for diagnosis. Verification uses deterministic checks such as URL matching, DOM retrieval, textual assertions, form-value checks, and state-change predicates. In practice, selectors are resolved on the current accessibility or DOM snapshot using role, visible text, and stable attributes rather than persistent node ids. Postconditions are instantiated from action-specific templates, such as value matching for `type`, state or navigation change for `click`, and option validation for `select`. This design keeps the repair signal grounded in observable page state rather than in model self-evaluation, matching the execution-oriented evaluation logic used in WebArena, VisualWebArena, and WorkArena [5, 8, 31].

4.3 Fault Localization

When execution fails, ContractSkill does not treat the failure as a monolithic property of the entire skill. Instead, it extracts a localized diagnosis

$$(i, e, \tau_i) = \text{Localize}\big(s, \mathcal{T}\big), \tag{5}$$

Table 2: Minimal patch operators in ContractSkill.

Operator	Typical use
Selector replacement	Replace or expand a brittle selector after `NOT_FOUND`.
Precondition insertion	Add a missing state assumption after `WRONG_STATE`.
Postcondition insertion	Add missing evidence of completion after `ASSERT_FAIL`.
Recovery insertion	Add a fallback action such as close-popup or reload.
Argument correction	Fix input values or action arguments after `INPUT_INVALID`.

where i is the failed step index, e is the structured error code, and τ_i is the relevant trace snippet or page-state evidence. This step-level localization is important because different failures imply different repair actions. A missing selector and an invalid state assumption should not trigger the same patch strategy. The same diagnosis-first discipline is central in semantics-based repair methods, which isolate a small repair site before synthesis or validation [14, 15].

4.4 Minimal Patch Operators

We define five local patch operators,

$$\mathcal{P} = \begin{array}{c}\{\text{SelReplace}, \text{PreInsert}, \text{PostInsert}, \\ \text{RecoveryInsert}, \text{ArgCorrect}\}.\end{array} \tag{6}$$

Using this operator set, the K-step repair neighborhood of the draft artifact is

$$\mathcal{N}_K(s_0) = \{p_k \circ \cdots \circ p_1(s_0) \mid p_r \in \mathcal{P},\ 0 \le k \le K\}. \tag{7}$$

Each operator changes only the artifact fragment tied to the diagnosed failure. This design preserves the rest of the artifact and avoids the instability that often accompanies full-text rewrites. The operator set intentionally favors small, interpretable edits, echoing the minimality bias emphasized in DirectFix and learning-guided repair systems such as Prophet [12, 13].

4.5 Verifier-Guided Repair Loop

Algorithmically, ContractSkill follows an iterative execute-diagnose-patch-validate loop. First, the source model produces a draft skill and compiles it into an artifact. Second, the artifact is executed on a repair set. Third, the verifier reports the failing step and error code. Fourth, the repair module chooses one or more minimal patch operators and constructs candidate patches. Finally, each candidate is validated under the same verifier and accepted only if it improves the current score. Concretely, we write

$$J(s) = \text{Succ}(s) - \lambda \text{Cost}(s) - \gamma \text{Edit}(s, s_0), \tag{8}$$

$$s' \text{ is accepted if } J(s') > J(s). \tag{9}$$

In our setup, each round is limited to a small edit budget, typically one to three patches, so that repair remains local and interpretable. The acceptance logic follows the broader verifier-guided refinement pattern in which counterexamples or failed checks narrow the next search step [4].

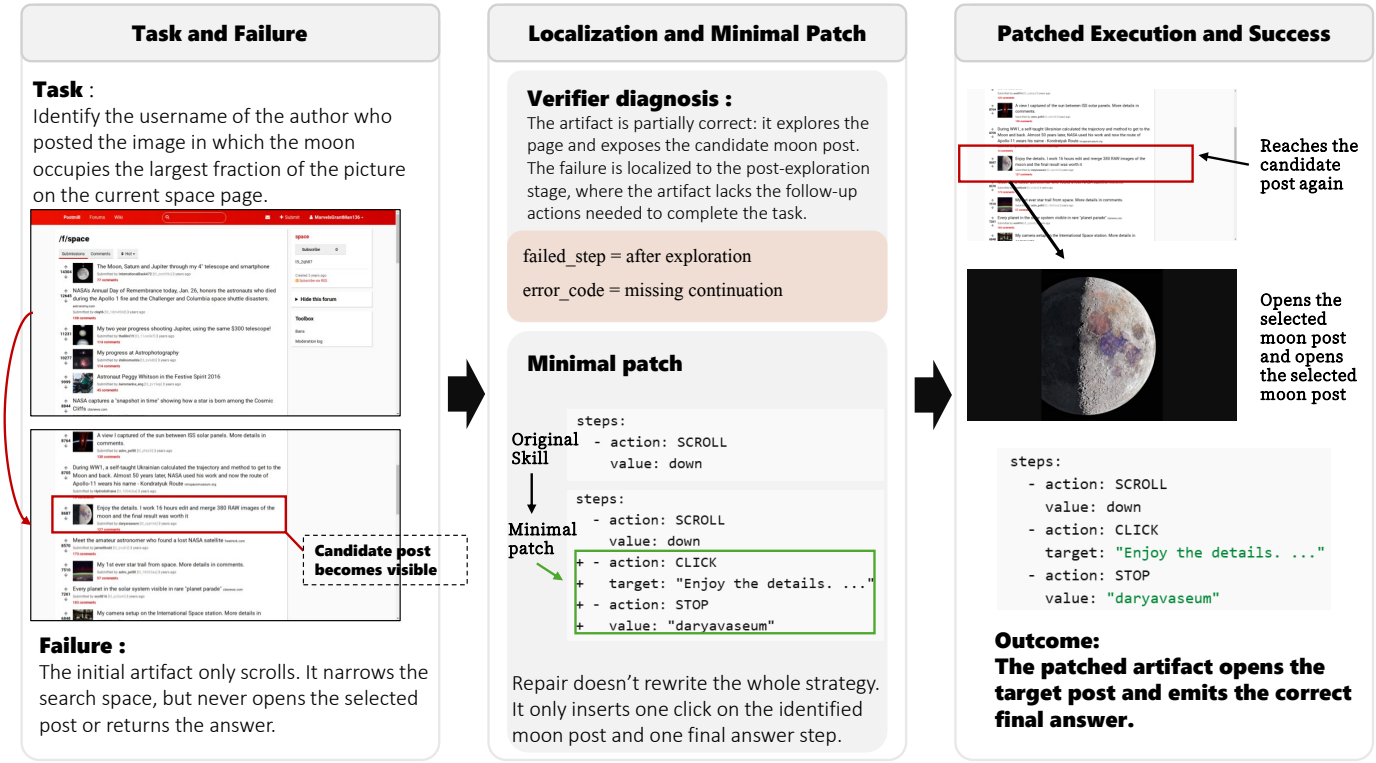

Figure 2: Representative repair case on VisualWebArena. The verifier localizes the failure to the post-exploration stage, and ContractSkill repairs the artifact by inserting only the missing continuation steps rather than rewriting the whole strategy.

5 Experimental Setup

Our experiments are organized around three questions. We ask (1) whether ContractSkill improves end-to-end web execution over direct acting and naive skill use, (2) whether any improvement comes specifically from verifier-guided local repair rather than rewriting alone, and (3) whether repaired skill artifacts can transfer across models. We structure the evaluation accordingly.

5.1 Benchmarks and Evaluation Splits

We evaluate on two benchmarks with complementary roles. VisualWebArena (VWA) [8] serves as the realistic multimodal web benchmark, where long-horizon interaction and environment noise make brittle skills particularly costly. We report VWA results on the cleaned **main_100** subset, i.e., the 100 tasks for which matched summaries are available across all compared runs. For MiniWoB [11], we use the **main_20** handbook split, which contains 20 tasks with 10 instances per task. We evaluate both benchmarks over three runs. MiniWoB serves as a controlled benchmark with simpler task structure, allowing the effect of repair decisions to be isolated more cleanly. This yields 300 task-level evaluations on VWA and 600 evaluation episodes on MiniWoB in total.

For transfer, we replace the usual seen-versus-held-out template split with benchmark-specific transfer layers that better match our claim. **Layer 1** (`layer1_paired_success`) contains tasks that both source models originally solve, enabling paired comparisons under matched difficulty. **Layer 2** (`layer2_source-only success`) contains tasks solved by only one source model, which directly tests whether a repaired skill can carry useful procedural knowledge beyond the overlap of easy cases. We report both directional settings, Q→G and G→Q.

5.2 Models

We instantiate the source and target agents with GLM-4.6V and Qwen3.5-Plus. In the self-use setting, the same model generates the draft skill, repairs it, and executes the repaired artifact. In the transfer setting, the source model produces and repairs the artifact, while the target model consumes it without regenerating the skill. This separation lets us test whether the final artifact encodes reusable procedure rather than model-specific hidden context.

5.3 Baselines

We compare against two core baselines in all main-result tables. **No-Skill** executes the task directly without any external procedural guidance, representing the standard direct-acting setting. **Self-Generated Skill** lets the acting model produce a skill and immediately use it, but does not perform verifier-guided repair after failure. This comparison separates the value of skill writing itself from the value of repair. On MiniWoB, we additionally include more controlled comparisons. **Text-Only Rewrite** rewrites the skill after failure in free-form text, without contracts, structured error codes, or localized edits, and therefore isolates the effect of

rewriting alone. We also report two ablations of ContractSkill. **No Failure Localization** removes targeted identification of the failing skill component, and **Unconstrained Repair** permits broad revisions rather than restricted local fixes. These controlled variants help determine whether the gains of ContractSkill arise from structured verifier-guided repair.

5.4 Metrics

Our primary metric is verifier-based success rate. To characterize efficiency, we also report average steps and average total tokens whenever those fields are present in the cleaned summaries. We interpret efficiency as a success-cost trade-off rather than strict dominance over every baseline, because a repaired artifact may spend extra steps or tokens to complete tasks that a brittle self-generated skill fails to finish. Tokens are reported as average total tokens per task and abbreviated in thousands (k). For transfer, we compare repaired-artifact execution against the target model's own self-generated skill baseline and report both absolute success and the point improvement.

5.5 Implementation Details

All compared methods use the same observation interface, action space, and deterministic verifier. For fairness, we keep the execution budget and the maximum allowed repair effort fixed across methods within each benchmark family. The verifier evaluates structured environment state through deterministic checks, including URL consistency, visible-text matching, form-value validation, and retrieval over the DOM summary. We re-resolve selectors on the latest page snapshot after every action and allow a short wait-and-requery window for delayed rendering, which reduces brittleness to dynamic interfaces without introducing non-deterministic matching. This controlled setup makes the comparison method-centric. Gains cannot be attributed to extra interaction budget, different action primitives, or weaker success criteria.

6 Main Results

We report the main results in the order of argumentative strength. We first test whether ContractSkill works in realistic web interaction, then examine the same claim in a controlled environment where the mechanism is easier to isolate, and finally test whether the repaired artifact transfers across models.

6.1 VisualWebArena

Table 3 reports the main VWA results. The overall pattern is clear. On realistic web tasks, simply having a skill is not enough, but repairing it into a contracted artifact is consistently beneficial. With GLM-4.6V, ContractSkill reaches **28.1%** success, compared with **12.5%** for No-Skill and **9.4%** for Self-Generated Skill. The efficiency profile also improves. Average steps drop from 12.59 for No-Skill to 3.62, and average token usage decreases from 52.7k to 46.45k. With Qwen3.5-Plus, the packaged final verified aggregation raises ContractSkill to **37.5%**, exceeding both No-Skill (**28.1%**) and Self-Generated Skill (**10.9%**), while keeping average token usage slightly below No-Skill (40.1k vs. 42.6k). These results matter because VWA is precisely the regime in which brittle skills tend to fail, with long trajectories, changing page state, and multimodal ambiguity. The fact that ContractSkill improves both models suggests that the gain comes from structured repair of the external artifact rather than from model-specific prompting tricks.

Table 3: Main results on VisualWebArena `main_100`.

GLM-VWA			
Method	Succ. (%)	Steps	Tokens (k)
No-Skill	12.5	12.59	52.7
Self-Generated Skill	9.4	1.95	5.6
ContractSkill	**28.1**	3.62	46.45
Qwen-VWA			
Method	Succ. (%)	Steps	Tokens (k)
No-Skill	28.1	9.41	42.6
Self-Generated Skill	10.9	1.91	7.3
ContractSkill	**37.5**	8.52	40.1

Note. Tokens are average total tokens per task reported in thousands.

Table 4: Main results on MiniWoB `main_20` with 10 instances per task.

GLM-MiniWoB			
Method	Succ. (%)	Steps	Tokens (k)
No-Skill	61.5	4.11	4.44
Self-Generated Skill	66.5	1.84	4.6
ContractSkill	**77.5**	1.93	5.16
Qwen-MiniWoB			
Method	Succ. (%)	Steps	Tokens (k)
No-Skill	56.5	1.61	1.9
Self-Generated Skill	60.5	1.85	4.5
ContractSkill	**81.0**	2.00	7.9

Note. Tokens are average total tokens per task reported in thousands.

6.2 MiniWoB

Table 4 provides the controlled counterpart to the VWA findings. Here the conclusion is even cleaner. ContractSkill is the strongest method for both models, reaching **77.5%** on GLM and **81.0%** on Qwen. Relative to Self-Generated Skill, the improvement is 11.0 points for GLM and 20.5 points for Qwen. The efficiency numbers show that this gain is not achieved by reverting to expensive direct exploration. On GLM, ContractSkill uses 1.93 average steps and 5.16k tokens, remaining far below the 4.11-step cost of No-Skill. On Qwen, ContractSkill increases steps only slightly over Self-Generated Skill (2.00 vs. 1.85) while producing a much larger success gain. Because MiniWoB removes much of the uncontrolled noise present in open web environments, these results strengthen the central claim that the improvement comes from repairable external procedure, not merely from extra interaction budget. We place Text-Only Rewrite in the ablation table below, where it serves as the most direct check against the "it is just rewriting" alternative explanation.

6.3 Cross-Model Transfer

Table 5 tests whether the repaired artifact remains useful after the source model is removed from the loop. The answer is yes. Aggregated over both directions, transfer raises VWA success from **32.6%** to **80.4%** (+47.8 points) and MiniWoB success from **83.5%** to **96.2%** (+12.8 points) relative to the target model's own self-generated skill baseline. The directional results show the same pattern rather than a one-sided effect. On VWA, Q→G improves from 29.2% to 79.2%, and G→Q improves from 36.4% to 81.8%. On MiniWoB, Q→G reaches 100.0% versus 87.5%, and G→Q reaches 92.3% versus 79.2%. Transfer execution remains lightweight, requiring only 1.77–3.21 average steps depending on the direction, and the released summaries record zero live token usage because the artifact is reused instead of regenerated online. This result is important for the paper's broader claim. The repaired object is not only a better prompt for the original model, but a portable procedural artifact. The Layer 2 gains are especially informative because they show transfer beyond the easy overlap region where both models already succeed.

7 Analysis and Ablation

We next ask why the main gains appear. Since the MiniWoB bundles provide the most complete controlled ablations, we use them to isolate the contribution of structured repair.

7.1 Failure Localization and Minimal Repair

Table 6 supports two conclusions. First, the gain is not explained by rewriting alone. Text-Only Rewrite reaches only 62.0% on GLM and 60.5% on Qwen, far below full ContractSkill at 77.5% and 81.0%, respectively. This gap is important because Text-Only Rewrite already gives the model an opportunity to revise the skill after observing failure; what it lacks is the contracted artifact, explicit verifier feedback, and localized repair target.

Figure 2 shows the same mechanism qualitatively on a representative VWA task. In this example, the original artifact already narrows the search space and surfaces the candidate moon post, but it stops at the exploration stage and never performs the final task-completing actions. The verifier localizes this failure as a missing continuation after exploration, and the repair module inserts only the missing click and answer steps. The repaired artifact therefore succeeds without changing the rest of the strategy.

Second, both design choices in ContractSkill matter. Removing failure localization drops performance to 65.0% on GLM and 70.0% on Qwen, indicating that knowing *where* the artifact fails is a major part of the benefit. Relaxing the repair objective into unconstrained editing performs slightly better than removing localization, but still reaches only 68.5% on GLM and 70.5% on Qwen. The efficiency trend points in the same direction. On Qwen, unconstrained repair increases average token usage from 7.9k to 11.3k without recovering the lost success. On GLM, it also increases token usage beyond both Text-Only Rewrite and the localization ablation. Taken together, these results support the intended mechanism of ContractSkill. Failure localization narrows the repair site, and minimal patching prevents costly global rewrites that are harder to validate and easier to destabilize.

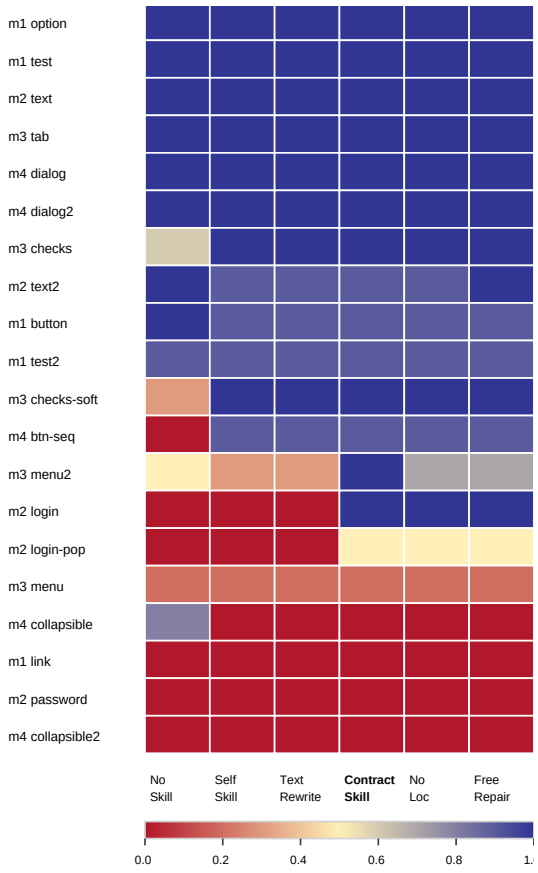

Figure 3: Template-level MiniWoB pass rates across the main baselines and ablations. ContractSkill gains concentrate on a small subset of repairable templates.

7.2 Template-Level MiniWoB Patterns

Figure 3 complements the aggregate MiniWoB ablation with template-level pass rates. Many easy templates are already saturated across all methods, so the overall gain does not come from uniform improvement everywhere. Instead, the advantage of full ContractSkill concentrates on a smaller set of templates that require reliable continuation after partial progress. The clearest gains appear on the two login templates and on `mw_m3_click_menu_2`, where the repaired artifact clearly outperforms both `skill_no_repair` and `text_only_rewrite`.

The same figure also clarifies the remaining boundary cases. Some templates, such as `mw_m1_click_link` and `mw_m2_enter_password`, remain unsolved for every method. A different failure mode appears on `mw_m4_click_collapsible`, where no-skill execution outperforms all skill-based variants. This pattern is consistent with the mechanism suggested by Table 6. Structured repair helps most when there is a reusable procedural skeleton to preserve and patch, but it is not a universal substitute for exploration on every template.

Table 5: Cross-model transfer results compared against the target model's own self-generated skill baseline. Aggregate rows merge Layer 1 and Layer 2 within each benchmark.

Setting	Transfer Succ. (%)	Target Skill (%)	Delta	Transfer Steps	Baseline Steps
VWA (aggregate)	**80.4**	32.6	+47.8	2.83	1.46
VWA Q→G	79.2	29.2	+50.0	3.21	1.46
VWA G→Q	81.8	36.4	+45.5	2.41	1.45
MiniWoB (aggregate)	**96.2**	83.5	+12.8	1.80	1.66
MiniWoB Q→G	100.0	87.5	+12.5	1.84	1.68
MiniWoB G→Q	92.3	79.2	+13.1	1.77	1.63

Table 6: MiniWoB ablation results for fault localization and repair constraints.

	GLM-MiniWoB			Qwen-MiniWoB		
Variant	Success (%)	Avg. Steps	Avg. Tokens (k)	Success (%)	Avg. Steps	Avg. Tokens (k)
Full ContractSkill	**77.5**	1.93	5.16	**81.0**	2.00	7.9
Text-Only Rewrite	62.0	1.71	2.7	60.5	1.70	5.3
No Failure Localization	65.0	1.75	4.9	70.0	1.95	7.4
Unconstrained Repair	68.5	1.92	6.6	70.5	2.00	11.3

Note. Tokens are average total tokens per task reported in thousands.

As a complementary VWA analysis, Figure 4 summarizes the residual failure structure across both source models. Relative to No-Skill, ContractSkill leaves fewer failures concentrated in broad late-stage profiles, especially step-6+ outcomes and loop-timeout-heavy mixes, and exposes more failures as localized selector or assertion errors. This shift does not remove all failures, but it makes the remaining ones more structured and more informative for targeted repair.

Figure 4: Supplementary VWA failure analysis across Qwen3.5-Plus and GLM-4.6V. The top panel shows verifier error codes among failed episodes. The bottom panel shows the first failing step bucket. Across both models, ContractSkill reduces the dominance of late coarse failures that are common in No-Skill execution and exposes a larger share of residual failures as earlier, more localizable breakdowns.

8 Discussion and Limitations

Taken together, the results suggest a specific view of skills in multimodal agents. A useful skill should not remain an ephemeral prompt that is rewritten from scratch in each episode. Instead, it should be externalized as a procedural object that can be checked, repaired, and reused. This perspective helps explain why ContractSkill performs well across both VWA and MiniWoB. The method does not simply ask the model to "try again," but converts failure into a structured repair problem over an explicit artifact.

At the same time, our current scope is intentionally narrow. We study repair for *single* multimodal web skill artifacts rather than a full lifelong skill library. The artifact schema is tailored to browser interaction and does not yet cover the broader action space of desktop environments such as OSWorld or Agent S [1, 24]. Our transfer experiments test cross-model reuse within the same benchmark family, which is a meaningful but still limited notion of portability. In addition, deterministic verification is strongest when success can be expressed through programmatic checks over URL, DOM state, or form values; visually ambiguous or highly dynamic web pages remain harder to verify reliably. Finally, the released bundles are not uniformly instrumented for deeper analyses such as stability curves, patch-size distributions, or error-code breakdowns. For this reason, we center the paper on evidence that is consistently available across all settings, namely success, interaction cost, and cross-model reuse. We view richer longitudinal skill-bank analysis

as a natural next step rather than a prerequisite for the present claim.

9 Conclusion

This paper examines a central weakness of multimodal web agents. AI-generated skills are easy to produce, but often too fragile to trust, reuse, or transfer. We present ContractSkill, a framework that turns a draft textual skill into a contracted executable artifact and improves it through deterministic verification, step-level fault localization, and minimal patch repair. Across both VisualWebArena and MiniWoB, ContractSkill consistently outperforms self-generated skills, while the repaired artifacts remain reusable across GLM-4.6V and Qwen3.5-Plus. Taken together, these results suggest a broader shift in how skills should be treated in web agents. Skills should not be disposable prompts that vanish after one run. They should instead be external procedural objects that can be audited, repaired, accumulated, and shared across models. We hope this perspective helps move multimodal web agents from one-shot prompting toward maintainable skill libraries.

References

[1] Saaket Agashe, Jiuzhou Han, Shuyu Gan, Jiachen Yang, Ang Li, and Xin Eric Wang. 2024. Agent S: An Open Agentic Framework that Uses Computers Like a Human. arXiv preprint arXiv:2410.08164. doi:10.48550/arXiv.2410.08164

[2] Minghao Chen, Yihang Li, Yanting Yang, Shiyu Yu, Binbin Lin, and Xiaofei He. 2024. AutoManual: Constructing Instruction Manuals by LLM Agents via Interactive Environmental Learning. In *Advances in Neural Information Processing Systems*, Vol. 37. Curran Associates, Inc., Red Hook, NY, USA.

[3] Weizhi Chen, Ziwei Wang, Leyang Yang, Sheng Zhou, Xiaoxuan Tang, Jiajun Bu, Yong Li, and Wei Jiang. 2025. PG-Agent: An Agent Powered by Page Graph. In *Proceedings of the 33rd ACM International Conference on Multimedia*. Association for Computing Machinery, New York, NY, USA, 6878–6887. doi:10.1145/3746027.3755189

[4] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. 2000. Counterexample-Guided Abstraction Refinement. In *Computer Aided Verification (Lecture Notes in Computer Science, Vol. 1855)*. Springer, Chicago, IL, USA, 154–169. doi:10.1007/10722167_15

[5] Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H. Laradji, Manuel Del Verme, Tom Marty, Leo Boisvert, Megh Thakkar, Quentin Cappart, David Vazquez, Nicolas Chapados, and Alexandre Lacoste. 2024. WorkArena: How Capable Are Web Agents at Solving Common Knowledge Work Tasks? arXiv preprint arXiv:2403.07718. doi:10.48550/arXiv.2403.07718

[6] Claire Le Goues, ThanhVu Nguyen, Stephanie Forrest, and Westley Weimer. 2012. GenProg: A Generic Method for Automatic Software Repair. *IEEE Transactions on Software Engineering* 38, 1 (2012), 54–72. doi:10.1109/TSE.2011.104

[7] Dongsun Kim, Jaechang Nam, Jaewoo Song, and Sunghun Kim. 2013. Automatic Patch Generation Learned from Human-Written Patches. In *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, San Francisco, CA, USA, 802–811. doi:10.1109/ICSE.2013.6606626

[8] Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Russ Salakhutdinov, and Daniel Fried. 2024. 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)*. Association for Computational Linguistics, Bangkok, Thailand, 881–905. doi:10.18653/v1/2024.acl-long.50

[9] Xiaoxiao Li. 2026. When Single-Agent with Skills Replace Multi-Agent Systems and When They Fail. doi:10.48550/arXiv.2601.04748 arXiv:2601.04748 [cs.AI]

[10] Xiangyi Li, Wenbo Chen, Yimin Liu, Shenghan Zheng, Xiaokun Chen, Yifeng He, Yubo Li, Bingran You, Haotian Shen, Jiankai Sun, Shuyi Wang, Qunhong Zeng, Di Wang, Xuandong Zhao, Yuanli Wang, Roey Ben Chaim, Zonglin Di, Yipeng Gao, Junwei He, Yizhuo He, Liqiang Jing, Luyang Kong, Xin Lan, Jiachen Li, Songlin Li, Yijiang Li, Yueqian Lin, Xinyi Liu, Xuanqing Liu, Haoran Lyu, Ze Ma, Bowei Wang, Runhui Wang, Tianyu Wang, Wengao Ye, Yue Zhang, Hanwen Xing, Yiqi Xue, Steven Dillmann, and Han chung Lee. 2026. SkillsBench: Benchmarking How Well Agent Skills Work Across Diverse Tasks. arXiv preprint arXiv:2602.12670. doi:10.48550/arXiv.2602.12670

[11] Evan Zheran Liu, Kelvin Guu, Panupong Pasupat, Tianlin Shi, and Percy Liang. 2018. Reinforcement Learning on Web Interfaces Using Workflow-Guided Exploration. In *International Conference on Learning Representations*. doi:10.48550/arXiv.1802.08802

[12] Fan Long and Martin Rinard. 2016. Automatic Patch Generation by Learning Correct Code. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. Association for Computing Machinery, St. Petersburg, FL, USA, 298–312. doi:10.1145/2837614.2837617

[13] Sergey Mechtaev, Jooyong Yi, and Abhik Roychoudhury. 2015. DirectFix: Looking for Simple Program Repairs. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*. IEEE, Florence, Italy, 448–458. doi:10.1109/ICSE.2015.63

[14] Sergey Mechtaev, Jooyong Yi, and Abhik Roychoudhury. 2016. Angelix: Scalable Multiline Program Patch Synthesis via Symbolic Analysis. In *Proceedings of the 38th International Conference on Software Engineering*. Association for Computing Machinery, Austin, TX, USA, 691–701. doi:10.1145/2884781.2884807

[15] Hoang Duong Thien Nguyen, Dawei Qi, Abhik Roychoudhury, and Satish Chandra. 2013. SemFix: Program Repair via Semantic Analysis. In *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, San Francisco, CA, USA, 772–781. doi:10.1109/ICSE.2013.6606623

[16] Tianlin Shi, Andrej Karpathy, Linxi Fan, Jonathan Hernandez, and Percy Liang. 2017. World of Bits: An Open-Domain Platform for Web-Based Agents. In *Proceedings of the 34th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 70)*. PMLR, Sydney, Australia, 3135–3144. https://proceedings.mlr.press/v70/shi17a.html

[17] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language Agents with Verbal Reinforcement Learning. In *Advances in Neural Information Processing Systems*, Vol. 36. Curran Associates, Inc., Red Hook, NY, USA.

[18] Yongliang Tan, Zhe Wang, Hongxin Zhang, Huao Cui, Ganqu Cui, and Wanli Ouyang. 2026. Enhancing Web Agents with a Hierarchical Memory Tree. doi:10.48550/arXiv.2601.06214 arXiv:2601.06214 [cs.AI]

[19] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2024. Voyager: An Open-Ended Embodied Agent with Large Language Models. Transactions on Machine Learning Research. https://openreview.net/forum?id=ehfRiF0R3a Accepted by TMLR; ICLR 2025 Journal Track.

[20] Junyang Wang, Haiyang Xu, Haitao Jia, Xi Zhang, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Jitao Sang. 2024. Mobile-Agent-v2: Mobile Device Operation Assistant with Effective Navigation via Multi-Agent Collaboration. In *Advances in Neural Information Processing Systems*, Vol. 37. Curran Associates, Inc., Red Hook, NY, USA.

[21] Zora Zhiruo Wang, Apurva Gandhi, Graham Neubig, and Daniel Fried. 2025. Inducing Programmatic Skills for Agentic Tasks. In *Proceedings of the Conference on Language Modeling*. https://openreview.net/forum?id=lsAY6fWsog

[22] Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. 2024. Agent Workflow Memory. doi:10.48550/arXiv.2409.07429 arXiv:2409.07429 [cs.CL]

[23] Zhaotian Weng, Antonis Antoniades, Deepak Nathani, Zhen Zhang, Xiao Pu, and Xin Eric Wang. 2026. Group-Evolving Agents: Open-Ended Self-Improvement via Experience Sharing. doi:10.48550/arXiv.2602.04837 arXiv:2602.04837 [cs.AI]

[24] Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. 2024. OSWorld: Benchmarking Multimodal Agents for Open-Ended Tasks in Real Computer Environments. CoRR, abs/2404.07972. doi:10.48550/arXiv.2404.07972

[25] Ran Xu, Kaixin Ma, Wenhao Yu, Hongming Zhang, Joyce C. Ho, Carl Yang, and Dong Yu. 2025. Retrieval-augmented GUI Agents with Generative Guidelines. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Suzhou, China, 17866–17875. doi:10.18653/v1/2025.emnlp-main.902

[26] Renjun Xu and Yang Yan. 2026. Agent Skills for Large Language Models: Architecture, Acquisition, Security, and the Path Forward. doi:10.48550/arXiv.2602.12430 arXiv:2602.12430 [cs.MA]

[27] Tianjun Yao, Yongqiang Chen, Yujia Zheng, Pan Li, Zhiqiang Shen, and Kun Zhang. 2026. ParamMem: Augmenting Language Agents with Parametric Reflective Memory. doi:10.48550/arXiv.2602.23320 arXiv:2602.23320 [cs.LG]

[28] Simon Yu, Gang Li, Weiyan Shi, and Peng Qi. 2025. PolySkill: Learning Generalizable Skills Through Polymorphic Abstraction. doi:10.48550/arXiv.2510.15863 arXiv:2510.15863 [cs.CL]

[29] Chi Zhang, Zhao Yang, Jiaxuan Liu, Yanda Li, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. 2025. AppAgent: Multimodal Agents as Smartphone Users. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, New York, NY, USA, 70:1–70:20. doi:10.1145/3706598.3713600

[30] Longtao Zheng, Boyuan Zheng, Arthur Cheang, Hanbo Li, Josh Gardner, and Graham Neubig. 2025. SkillWeaver: Building Website-Specific Skills for Web Agents via Iterative Self-Improvement. doi:10.48550/arXiv.2504.07079 arXiv:2504.07079 [cs.AI]

[31] Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. 2024. WebArena: A Realistic Web Environment for Building Autonomous Agents. In *The Twelfth International Conference on Learning Representations*. OpenReview.net, Vienna, Austria. https://openreview.net/forum?id=oKn9c6ytLx Poster.