SCAFFOLD: Self-Improving Web Agents via Recursive Parametric Skill Abstraction

Bowei He[1, 2], Xiaokun Zhang[3], Meng Ding[4], Xue Liu[1, 2]
[1] MBZUAI, [2] McGill University, [3] CityUHK, [4] UMass Boston
Bowei.He@mbzuai.ac.ae

Abstract

Web agents need to navigate visually rich, long-horizon interfaces that change across sites, yet most previous agents still learn each task in isolation and discard the procedural knowledge they accumulate. Recent skill-augmented frameworks take an important first step, but they treat the skill library as a flat or two-tier prompt-side cache and offer no principled mechanism for compressing redundancy or composing skills recursively. We introduce SCAFFOLD, a self-improving framework for visual web agents that (i) induces parametric, executable skills from successful trajectories under a multi-instance abstraction constraint, (ii) maintains a recursively composed hierarchy in which higher-level skills invoke lower-level ones, (iii) compacts the library via a minimum-description-length (MDL) criterion and behavioral equivalence checking, and (iv) periodically distills skill-augmented trajectories back into model weights to internalize the abstractions. Across WebArena, VisualWebArena, and a held-out split of Online-Mind2Web, SCAFFOLD improves success rate by 11.1–17.2 absolute points over the strongest skill-augmented baseline and shows monotonic gains across five self-improvement iterations without library collapse. We release the code and documents in the Github repository.

1 Introduction

Autonomous web agents, LLM-based systems that complete user instructions by interacting with browsers, have advanced rapidly with the arrival of strong multimodal foundation models (Zheng et al., 2024a; He et al., 2024; Qin et al., 2025; Wang et al., 2025a). Yet on every realistic benchmark, from WebArena (Zhou et al., 2024) and VisualWebArena (Koh et al., 2024) to OSWorld (Xie et al., 2024) and Online-Mind2Web (Xue et al., 2025), the performance still lags human performance by a wide margin. A central cause is that current agents treat each task as an isolated episode: procedural knowledge acquired while booking a flight is discarded before the next task begins, even if that next task shares most of the same sub-procedure (Wang et al., 2025b; Zheng et al., 2025).

This observation has motivated a wave of *self-improving* web agents that retain experience in some form. Three sub-paradigms have emerged. **Trajectory replay** methods store raw experience for retrieval-augmented prompting (Zheng et al., 2024b; Zhao et al., 2024; Chhikara et al., 2025). **Workflow induction** methods abstract action sequences into natural-language routines that the agent can re-read at inference time (Wang et al., 2025b; Fang et al., 2026). **Skill induction** methods go further and synthesize callable APIs or parametric skills, exemplified by SKILLWEAVER (Zheng et al., 2025), APPAGENTX (Jiang et al., 2025), and the very recent SKILLRL (Xia et al., 2026). Skill induction is the strongest of them because the reuse unit is an executable program rather than a textual hint, and skills can in principle be *composed*. This property, as long argued by the hierarchical reinforcement learning community (Sutton et al., 1999), is essential for tackling long-horizon problems.

Despite this trajectory, three limitations of current skill-augmented agents remain unresolved.[1] First, induced skill libraries are essentially flat: SKILLWEAVER stores APIs in a single pool, and SKILLRL uses only a two-tier (general / task-specific) split. Since neither supports skills calling other skills, the maximum abstraction depth is bounded by one. Second, libraries grow monotonically and accumulate near-duplicate skills with no principled compression mechanism, a known failure mode in lifelong learning (Wang et al., 2024a). Third, the knowledge encoded in skills exists only in prompts and is never internalized into the model,

[1] We focus on visual web agents throughout; complementary work on text-only LLM agents includes (Xia et al., 2026; Wang et al., 2026; Wu et al., 2025a).

Findings of the Association for Computational Linguistics: EMNLP 2026, pages 16634–16650
October 24-29, 2026 ©2026 Association for Computational Linguistics

leaving the base policy after iterations as weak as it was at the beginning.

Contributions. We introduce SCAFFOLD, a framework that addresses each of these limitations:

- A *multi-instance parameter induction* procedure that proposes a skill only when multiple semantically similar trajectories support the same parametric abstraction, sharply reducing spurious abstracted skills (§3.3).
- A *recursive composition* mechanism: at iteration k, the inducer is allowed to call any skill from iterations $1, \ldots, k$, producing a hierarchy whose depth grows monotonically. We track depth and reuse explicitly in our design (§3.4).
- An *MDL-driven library compaction* step that periodically merges behaviorally equivalent skills, refactors recurring sub-patterns into new mid-level skills, and prunes unused ones, with empirical validation on a held-out task set (§3.5).
- A *distillation loop* that uses skill-augmented trajectories to supervise-fine-tune the base policy, converting the in-context library into improved weights (§3.6).

On WebArena, VisualWebArena, and a held-out Online-Mind2Web subset, SCAFFOLD improves over SKILLWEAVER by 13.6–17.7 absolute success-rate points. Critically, it continues to improve through five self-improvement iterations (whereas baselines saturate at two), exhibits a desirable library-depth and reuse-rate distribution, and zero-shot transfers to held-out sites with a 17.2-point margin over the strongest competitor.

2 Related Work

Web and GUI agents. Modern web agents either prompt strong proprietary models (Zheng et al., 2024a; He et al., 2024; Yang et al., 2025) or fine-tune open vision-language models on large GUI trajectory corpora (Cheng et al., 2024; Hong et al., 2024; Wu et al., 2025b; Xu et al., 2025; Qin et al., 2025; Wang et al., 2025a; Lai et al., 2024). The latter line achieves impressive grounding but relies on static datasets that fail to capture the procedural diversity of real websites (Zheng et al., 2025). Reinforcement-learning-based agents close part of this gap by training on the environment directly (Qi et al., 2025; Patel et al., 2024; Gandhi and Neubig, 2026; Shao et al., 2024; Guo et al., 2025), but typically lack any mechanism for accumulating reusable behavioral abstractions, which is our focus.

Self-improvement through experience. The idea of letting a model improve itself by training on its own filtered generations dates back to STAR (Zelikman et al., 2022) and was extended to richer settings by Quiet-STaR (Zelikman et al., 2024) and self-play fine-tuning (Chen et al., 2024). For agents specifically, (Patel et al., 2024) showed that filtered self-generated trajectories can lift WebArena performance, and Reflexion (Shinn et al., 2023) introduced verbal in-context self-correction without weight updates. SCAFFOLD inherits the spirit of these works but operates over a *structured* skill library rather than raw text or trajectory data.

Skill libraries and workflows. VOYAGER (Wang et al., 2024a) pioneered LLM-driven skill libraries in the programmatic Minecraft environment; TROVE (Wang et al., 2024b) extended this idea to tool induction for programmatic tasks. For web/GUI agents, Agent Workflow Memory (AWM) (Wang et al., 2025b) induces natural-language workflows from past trajectories and retrieves them at test time, achieving large gains on Mind2Web and WebArena. SKILLWEAVER (Zheng et al., 2025) advances this paradigm by synthesizing skills as executable APIs and using iterative practice for refinement. APPAGENTX (Jiang et al., 2025) explores analogous ideas on mobile UIs. The most direct prior work is SKILLRL (Xia et al., 2026), which trains an LLM agent jointly with a two-tier hierarchical skill library (general skills + task-specific skills) using GRPO (Shao et al., 2024). SCAFFOLD differs in operating on visual web agents with truly recursive hierarchies, an MDL-based compaction step, and an explicit skill transfer evaluation.

Concurrent skill frameworks. POLYSKILL (Yu et al., 2026) separates a skill's abstract goal from its per-site implementations via abstract interfaces, whereas our skills re-bind semantic references at run time; SKILLEVO (Xu et al., 2026) pairs GRPO with an evolving skill path graph whose reuse unit is an experience path rather than a parametric program and which carries no compression objective.

Memory and lifelong learning for agents. A parallel literature treats agent experience as memory rather than skill, including MEM0 (Chhikara et al., 2025), EXPEL (Zhao et al., 2024), EVOLVER (Wu et al., 2025a), SYNAPSE (Zheng et al., 2024b), and trajectory-informed memory generation (Fang et al., 2026). These methods are largely orthogonal to skill induction and could be combined

with SCAFFOLD; we use MEM0 and SYNAPSE as memory-style baselines in §4. Finally, classical hierarchical RL (Sutton et al., 1999) and library learning in program synthesis (Ellis et al., 2021) provide theoretical grounding for the recursive composition and MDL-based compression operations central to our method.

3 Methodology

3.1 Problem Setup and Notation

We consider an agent operating in a partially observable web environment $\mathcal{E}$. At step t, the agent receives observation $o_t = (s_t, d_t, u_t)$ consisting of a screenshot s_t, accessibility-tree / document object model (DOM) snippet d_t, and current URL u_t. The agent emits an action $a_t \in \mathcal{A}_{\text{prim}} \cup \mathcal{A}_{\text{skill}}$, where $\mathcal{A}_{\text{prim}}$ are primitive actions (click, type, scroll, wait) and $\mathcal{A}_{\text{skill}}$ is the set of currently available skill invocations. A task $\tau = (I, \text{init}, V)$ specifies a natural-language instruction I, an initial environment state, and a verifier V that returns binary success on the final state. We denote a trajectory by $\zeta = (o_0, a_0, o_1, \ldots, o_T)$. A **skill** is a tuple $\sigma = (\text{name}, \text{description}, \boldsymbol{\theta}, \text{pre}, \text{body}, \text{post}, d_\sigma)$. $\boldsymbol{\theta}$ is a typed parameter list admitting primitive values (strings, numbers), semantic element references (natural-language descriptors grounded to DOM elements at run time by a VLM grounder), and pointers to other skills. pre and post are LLM-checkable preconditions and postconditions phrased as predicates over o_t, body is an executable program over $\mathcal{A}_{\text{prim}} \cup \mathcal{A}_{\text{skill}}$, and $d_\sigma \in \mathbb{N}$ is the *abstraction depth*. The library $\mathcal{L}_k$ at iteration k is a set of such skills together with an embedding-based retrieval index keyed on description. We call σ *semi-parametric*: control flow inside body calls no policy model, while semantic element references are grounded on the live observation by a small VLM (§4.9).

3.2 Overall Pipeline

Each self-improvement iteration $k \to k+1$ consists of five stages, executed in order: (1) *trajectory collection* with the current policy π_k and library $\mathcal{L}_k$; (2) *multi-instance skill induction*; (3) *recursive composition* of new skills over $\mathcal{L}_k$; (4) *MDL-driven library compaction*; and (5) *distillation* of the skill-augmented behavior into π_{k+1}. Algorithm 1 summarizes the loop. Below we describe each stage in detail; the design choices are motivated by the failure modes of prior skill-induction methods that we observe empirically (cf. §4.3). The overall framework is also illustrated in Figure 1.

Algorithm 1 SCAFFOLD Self-Improvement Loop

Require: Base policy π_0, task pool $\mathcal{T}$, verifier V, iterations K, compaction interval M
$\mathcal{L}_0 \leftarrow \emptyset$
for $k = 0, \ldots, K-1$ **do**
  $\mathcal{Z}_k \leftarrow \text{ROLLOUT}(\pi_k, \mathcal{L}_k, \mathcal{T})$ ▷ collect trajectories
  $\mathcal{Z}_k^+ \leftarrow \{\zeta \in \mathcal{Z}_k : V(\zeta) = 1\}$
  $\mathcal{C}_k \leftarrow \text{CLUSTERBYINSTR}(\mathcal{Z}_k^+)$ ▷ group similar trajs
  $\Sigma^{\text{new}} \leftarrow \emptyset$
  for cluster $C \in \mathcal{C}_k$ with $|C| \geq n_{\min}$ **do**
    $\sigma \leftarrow \text{INDUCE}(C, \mathcal{L}_k)$ ▷ multi-instance abstraction; may invoke any $\sigma' \in \mathcal{L}_k$
    if $\text{VALIDATEHOLDOUT}(\sigma, \mathcal{T}_{\text{val}})$ **then**
      $\Sigma^{\text{new}} \leftarrow \Sigma^{\text{new}} \cup \{\sigma\}$
    end if
  end for
  $\mathcal{L}_{k+1} \leftarrow \mathcal{L}_k \cup \Sigma^{\text{new}}$
  if $(k+1) \bmod M = 0$ **then**
    $\mathcal{L}_{k+1} \leftarrow \text{COMPACT}_{\text{MDL}}(\mathcal{L}_{k+1}, \mathcal{Z}_{0:k}^+)$
  end if
  $\pi_{k+1} \leftarrow \text{DISTILL}(\pi_k, \mathcal{L}_{k+1}, \mathcal{Z}_k^+)$
end for
return $\pi_K, \mathcal{L}_K$

3.3 Multi-Instance Skill Induction

A persistent failure mode in workflow-induction methods is the synthesis of *spurious abstractions*: a skill is proposed from a single trajectory but encodes incidental details (specific element IDs, hard-coded text) that fail to generalize. We address this by requiring every candidate skill to be supported by at least $n_{\min}$ trajectories with the same parameter structure.

Successful trajectories $\mathcal{Z}_k^+$ are first clustered by the embedding of their instruction I. Within each cluster $C = \{\zeta_1, \ldots, \zeta_n\}$ with $n \geq n_{\min}$, the inducer (a frozen LLM) is prompted with all n trajectories and asked to: (a) identify which positions vary across trajectories and propose a typed parameter list $\boldsymbol{\theta}$, (b) emit an executable body using $\boldsymbol{\theta}$, primitives in $\mathcal{A}_{\text{prim}}$, and any skill in $\mathcal{L}_k$, (c) state a precondition / postcondition pair as Python expressions over o_t. The induced skill must *re-execute correctly* on at least one held-out instance from C; otherwise it is discarded. This validation step replaces the post-hoc filtering used in SKILLWEAVER and AWM, and we show in §4.3 that it nearly halves the rate of induced-then-deprecated skills across iterations.

3.4 Recursive Composition and Depth Tracking

The library is hierarchical by construction: when the inducer emits a body, it may invoke any skill

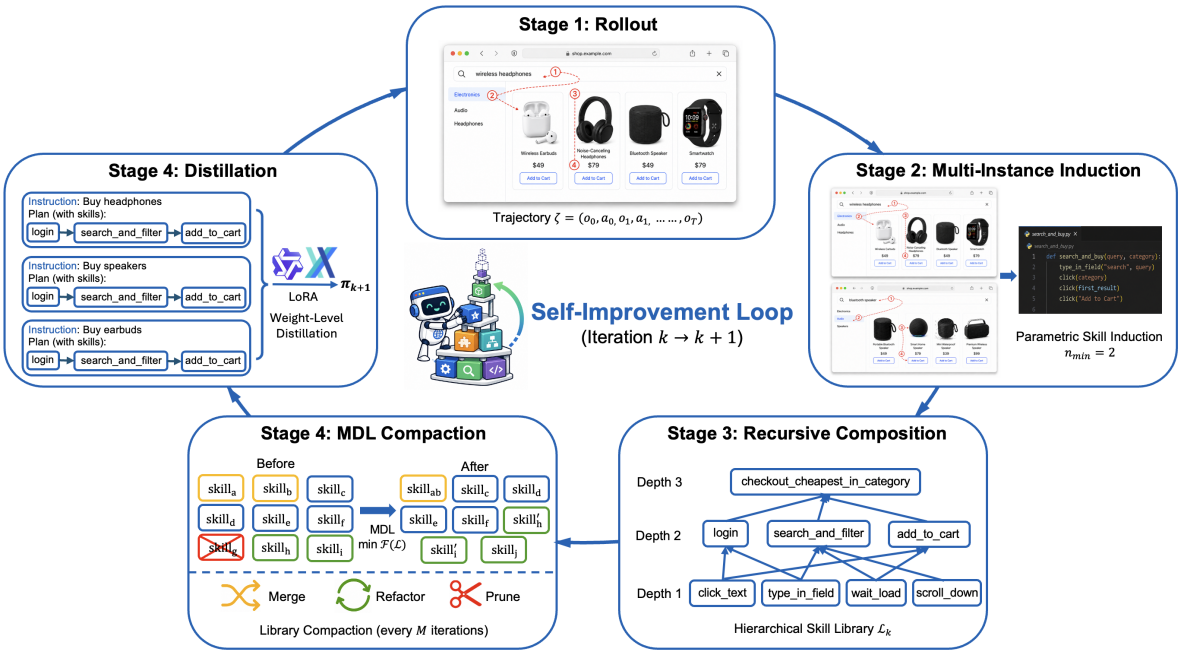

Figure 1: Overview of SCAFFOLD framework. The five stages are executed sequentially in each iteration: rollout, multi-instance induction, recursive composition, MDL-driven compaction, and distillation. The library growing in both depth and width is periodically compressed.

$\sigma' \in \mathcal{L}_k$ as a sub-routine. The depth of a new skill σ is defined as

$$d_\sigma = 1 + \max_{\sigma' \in \text{calls}(\sigma)} d_{\sigma'}, \tag{1}$$

with $d_\sigma = 1$ when $\text{calls}(\sigma) \subseteq \mathcal{A}_{\text{prim}}$. We do not enforce a maximum depth; instead we measure how depth evolves across iterations (§4.6). To prevent runaway abstraction, we forbid cycles via a simple topological check at induction time, and we cap the per-skill body length at 30 lines.

The choice to allow *any* prior skill, not just skills from iteration $k-1$, enables the inducer to refactor a long chain of primitives into a single mid-level skill at any iteration. Concretely, an induced `book_flight` skill at iteration 3 may call `search_dates` (iteration 1), `fill_passenger_info` (iteration 2), and a freshly synthesized `confirm_payment` (also iteration 3). This stands in contrast to SKILLRL's fixed two-tier hierarchy, which forces the agent to flatten the natural call structure of real web procedures.

3.5 MDL-Driven Library Compaction

Without governance, the library grows monotonically and accumulates redundant skills, which both inflates the retrieval namespace and dilutes the gradient signal during distillation. Every M iterations (we use M=2) we apply a *compaction* step whose objective is the classical minimum-description-length functional (Rissanen, 1978):

$$\mathcal{F}(\mathcal{L}) = \underbrace{\sum_{\sigma \in \mathcal{L}} |\sigma|}_{\text{library cost}} + \underbrace{\sum_{\zeta \in \mathcal{Z}^+} \min_{\text{parse}(\zeta|\mathcal{L})} |\text{parse}(\zeta \mid \mathcal{L})|}_{\text{data cost}}. \tag{2}$$

Here $|\sigma|$ is the token length of the skill body, and $|\text{parse}(\zeta \mid \mathcal{L})|$ is the length of the shortest program over $\mathcal{L} \cup \mathcal{A}_{\text{prim}}$ that reproduces ζ. Intuitively, $\mathcal{F}$ rewards libraries that are short (few, concise skills) but that can compactly explain past successful behavior; it penalizes both *over-specific* skills (used by few trajectories) and *redundant* skills (whose role is already covered).

We approximate the minimization of Eq. 2 via three greedy operators applied iteratively until $\mathcal{F}$ no longer decreases: **(i) Merge**: two skills σ_a, σ_b that are *behaviorally equivalent* on a sampled hold-out set (i.e., produce identical post-states with probability $\geq \rho$) are merged into the shorter of the two, with the longer being replaced by an alias. **(ii) Refactor**: if the same primitive-action subsequence of length $\geq \ell_{\min}$ appears in $\geq r_{\min}$ existing skill bodies, the inducer is asked to propose a new mid-level skill that captures it, and the existing skills are rewritten to call it. **(iii) Prune**: skills used by zero trajectories in $\mathcal{Z}^+_{0:k}$ over the most recent M iterations are removed (unless invoked transitively by another skill). Each candidate modifica-

tion is accepted only if (a) it reduces $\mathcal{F}$ and (b) it does not decrease success rate on a held-out validation set $\mathcal{T}_{\text{val}}$. This last constraint is the key safety net that distinguishes principled compaction from naive deduplication.

Functional redundancy and lifecycle. Equivalence is tested on a stratified context set including perturbed states, so two skills reaching one goal by different routes are collapsed only when indistinguishable everywhere: 14 goal-overlapping groups survive at k=5. Unmet preconditions block invocation at run time, violated postconditions trigger one retry, and skills whose rolling success rate drops below 60% are quarantined (Appendix G).

3.6 Distillation Back into Weights

After each iteration we have $\mathcal{Z}_k^+$, a set of successful trajectories produced by π_k *with* the in-context library $\mathcal{L}_{k+1}$. We convert each such trajectory into a *plan-augmented* training example $(I, \text{plan}_\sigma, \zeta)$, where plan_σ is the sequence of skill invocations the agent used. The base policy π_{k+1} is then obtained by supervised fine-tuning of π_k on these triples with a standard token-level cross-entropy loss, plus an auxiliary loss that predicts the next skill name conditioned only on $(I, o_{0:t})$, encouraging the model to internalize *when* to invoke each abstraction. We use LoRA (Hu et al., 2022) adapters and a constant learning rate of 1×10^{-5} for stability across iterations. Examples are first re-parsed against the compacted library $\mathcal{L}_{k+1}$, with merged calls rewritten through the alias map and trajectories invoking pruned skills dropped (3.6% on average), so deprecated skills are never reinforced.

The distillation step is what makes SCAFFOLD's improvement compound: at iteration $k+1$ the base policy itself has improved, so the new trajectories it produces (with the larger library) push the frontier of solvable tasks higher, yielding richer skills to induce in iteration $k+2$. This is the recursive self-improvement loop in the title of the paper, made concrete.

4 Experiments

4.1 Experiments Setup

4.1.1 Datasets

We evaluate on three benchmarks chosen to cover both controlled and live web environments: **WebArena** (Zhou et al., 2024): 812 long-horizon tasks across four self-hosted sites (E-commerce, social forum, collaborative development, content management systems). Evaluation is execution-based with a deterministic verifier per task. **VisualWebArena** (Koh et al., 2024): 910 tasks built on top of WebArena that explicitly require visual reasoning; we use the full benchmark. **Online-Mind2Web** (Xue et al., 2025): 300 tasks on 136 websites across 12 domains; we partition by sites into a 9-domain training pool and a 3-domain held-out test set (Jobs & Careers, Travel & Transportation, Government & Services) in main experiments, enabling a controlled measurement of skill transfer.

4.1.2 Baselines

We compare SCAFFOLD against three families: **(A) Strong zero-shot agents**: REACT (Yao et al., 2023), SEEACT (Zheng et al., 2024a), WEBVOYAGER (He et al., 2024), and AGENTOCCAM (Yang et al., 2025), all instantiated on top of the same Qwen2.5-VL-7B (Yang et al., 2024) base for a fair comparison; we additionally report numbers from the stronger UI-TARS-7B (Qin et al., 2025) backbone. **(B) Memory- and workflow-augmented agents**: REFLEXION (Shinn et al., 2023) (in-context reflection), SYNAPSE (Zheng et al., 2024b) (trajectory exemplars), MEM0 (Chhikara et al., 2025) (long-term memory), EXPEL (Zhao et al., 2024) (experience distillation), and Agent Workflow Memory (Wang et al., 2025b). **(C) Skill-induction and skill-RL agents**: SKILLWEAVER (Zheng et al., 2025), APPAGENTX (Jiang et al., 2025), EVOLVER (Wu et al., 2025a), SKILLRL (Xia et al., 2026), and WEBRL (Qi et al., 2025). For every baseline we use the same base model, the same task pool, the same number of self-improvement iterations (K=5), and the same evaluation protocol to isolate algorithmic differences.

4.1.3 Evaluation Metrics

We report: **(1) Success rate (SR)**: fraction of tasks for which the verifier returns 1. **(2) Step efficiency**: average steps per successful task (lower is better). **(3) Library statistics**: total skill count $|\mathcal{L}|$, mean abstraction depth $\bar{d}$, skill reuse rate (fraction of skills called by ≥ 2 tasks). **(4) Cross-site transfer SR**: SR on evaluation sites different from training ones. **(5) Iteration scaling**: SR as a function of k. To control for environment stochasticity (dynamic DOM, server load, LLM sampling), we report mean $\pm$ standard deviation over 3 independent runs with different seeds, and use paired bootstrap tests for significance, marking $p < 0.05$ with †.

Method	WebArena	VWA	OM2W-X
(A) Zero-shot agents (Qwen2.5-VL-7B)			
REACT (Yao et al., 2023)	8.1 ± 0.6	6.2 ± 0.4	11.2 ± 0.9
SEEACT (Zheng et al., 2024a)	12.4 ± 0.5	9.7 ± 0.6	14.0 ± 1.0
WEBVOYAGER (He et al., 2024)	15.3 ± 0.7	11.8 ± 0.5	17.5 ± 1.2
AGENTOCCAM (Yang et al., 2025)	16.8 ± 0.4	12.4 ± 0.6	18.1 ± 0.8
(B) Memory/Workflow-augmented			
REFLEXION (Shinn et al., 2023)	14.0 ± 0.8	10.5 ± 0.7	15.3 ± 1.0
SYNAPSE (Zheng et al., 2024b)	17.6 ± 0.6	13.1 ± 0.5	18.4 ± 0.9
MEM0 (Chhikara et al., 2025)	18.2 ± 0.7	13.9 ± 0.6	19.3 ± 1.1
EXPEL (Zhao et al., 2024)	17.9 ± 0.5	13.6 ± 0.7	19.0 ± 1.0
AWM (Wang et al., 2025b)	24.3 ± 0.6	18.7 ± 0.7	23.6 ± 1.0
(C) Skill-induction & skill-RL			
WEBRL (Qi et al., 2025)	28.7 ± 0.9	19.4 ± 0.8	24.1 ± 1.3
APPAGENTX (Jiang et al., 2025)	22.5 ± 0.7	17.2 ± 0.6	22.8 ± 1.1
EVOLVER (Wu et al., 2025a)	26.4 ± 0.8	19.0 ± 0.7	24.4 ± 1.2
SKILLWEAVER (Zheng et al., 2025)	29.1 ± 0.7	21.5 ± 0.8	25.8 ± 1.1
SKILLRL (Xia et al., 2026)	31.6 ± 0.6	22.7 ± 0.7	26.3 ± 1.0
SCAFFOLD (ours)	$\mathbf{42.7} \pm 0.8^{\dagger}$	$\mathbf{36.3} \pm 0.9^{\dagger}$	$\mathbf{43.5} \pm 1.2^{\dagger}$
Upper-ref: SCAFFOLD *on UI-TARS-7B* (Qin et al., 2025)	51.2 ± 0.8	44.6 ± 1.0	49.1 ± 1.3

Table 1: Main results (Success Rate %) on WebArena, VisualWebArena (VWA), and Online-Mind2Web held-out test split (OM2W-X). Mean $\pm$ std over 3 seeds. $\dagger$: $p < 0.05$ over the strongest competitor (SKILLRL) by paired bootstrap. All rows in (A)–(C) and the bold SCAFFOLD row use Qwen2.5-VL-7B as the base; the last row is included as an upper reference with a stronger backbone.

4.1.4 Implementation Details

Base model is Qwen2.5-VL-7B-Instruct (Yang et al., 2024) unless stated otherwise; we also report results with UI-TARS-7B (Qin et al., 2025) as a stronger backbone. The skill inducer and the MDL refactor proposer are GPT-4o-2024-08, queried with temperature 0.3. Trajectories are collected with temperature 0.7 and a 30-step horizon. We use $n_{\min}$=2 for multi-instance induction, M=2 for compaction interval, ρ=0.9 for behavioral-equivalence threshold, $\ell_{\min}$=3 and $r_{\min}$=3 for the refactor operator. Distillation uses LoRA (rank 64, α=128) on top of the base model for 1 epoch per iteration. The learning rate is set as 1×10^{-5}. All experiments run on 8×A100-80GB; a complete 5-iteration run on WebArena takes nearly 36 hours. Two protocol points deserve emphasis. Every skill- and workflow-induction baseline (SKILLWEAVER, SKILLRL, APPAGENTX, EVOLVER, AWM) uses the same GPT-4o-2024-08 checkpoint and temperature for its synthesis component, as in the original SKILLWEAVER setup, so comparisons isolate algorithmic differences rather than inducer strength; and the 30-step horizon is counted in *primitive* actions for every method, a skill invocation being a macro action whose primitives draw on the same shared budget (§4.9).

4.2 Main Results

Table 1 reports main results. SCAFFOLD achieves 42.7% SR on WebArena, 36.3% on VisualWebArena, and 43.5% on the OM2W held-out test split, improving over the strongest skill-augmented competitor (SKILLRL) by 11.1, 13.6, and 17.2 absolute points respectively, all statistically significant under paired bootstrap. Three trends are worth highlighting. First, skill-induction and skill-RL methods systematically outperform both zero-shot and memory/workflow-augmented agents on all three benchmarks. This indicates that explicit, reusable skill abstraction is the right unit of accumulated experience for web agents. Second, the advantage of SCAFFOLD on OM2W-X is most significant, achieving 17.2 points over SKILLRL, confirming that the recursive parametric skill abstraction coupled with library compaction transfers materially better than two-tier prompt-based skills. Third, replacing the base model with UI-TARS-7B yields a further 7.5-point lift averaged across all three benchmarks, suggesting that SCAFFOLD is complementary to advances in GUI-specified model backbones rather than a substitute for them.

Variant	WA	VWA	OM2W-X
Full SCAFFOLD	**42.7**	**36.3**	**43.5**
– multi-instance ($n_{\min}$=1)	38.1	31.7	38.4
– recursive composition (depth≤1)	36.4	30.5	36.1
– MDL compaction	39.6	33.8	40.0
– distillation (in-context only)	34.7	28.4	34.0
– holdout validation in induction	39.0	32.6	39.7
– all four innovations (= SkillWeaver)	29.1	21.5	25.8

Table 2: Ablation on the four core innovations. Each row removes a single component while keeping the rest unchanged. Last row is one of strongest baselines.

4.3 Ablation Study

Table 2 isolates each of SCAFFOLD's four innovations. Removing distillation has the largest effect (−8.0 points on WebArena), confirming our central claim: a prompt-only skill library, however well structured, is fundamentally limited by the frozen base policy. Recursive composition is the second-largest contributor (−6.3 points on WebArena, −7.4 on OM2W-X), which we attribute to its ability to express long-horizon procedures (e.g., `book_full_trip` composing `search_flight`, `select_seat`, `enter_payment`) without runaway prompt length. On WebArena, the multi-instance induction and holdout validation contribute 4.6 and 3.7 points respectively, smaller absolute effects. But they are essential for stability across iterations as we show next. MDL compaction contributes a modest 3.1 points but is what keeps the library size bounded (cf. Fig. 3). The drops sum to 25.7 against a total gap of 13.6, so the components are synergistic rather than additive.

4.4 Skill Abstraction vs. Iterative Fine-Tuning

Since distillation produces the largest ablation gap, one may ask whether the gains come from skill abstraction or merely from iterative fine-tuning on self-generated data. Table 6 in Appendix E decouples the two with the trajectories held fixed. A STaR-style loop (Zelikman et al., 2022) without any library reaches 30.4, since fewer hard tasks are ever solved; abstraction without distillation reaches 34.7, since a frozen policy caps what is reachable however good the library is. Standard SFT on the identical trajectories with skill calls flattened to primitives reaches 38.6, so 4.1 of the remaining points come from the plan-augmented format and the auxiliary loss that teach the policy *when* to invoke *which* abstraction, and the effect is not LoRA-specific (42.9 with full-parameter tuning). Abstraction keeps distillation paying off; distillation makes abstraction cumulative rather than prompt-bound.

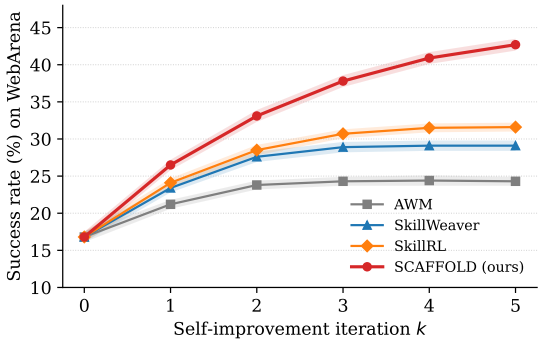

Figure 2: Success rate vs. self-improvement iteration on WebArena. SCAFFOLD continues to improve through K=5 while baselines saturate around k=2–3. Shaded regions are 1 std over 3 seeds.

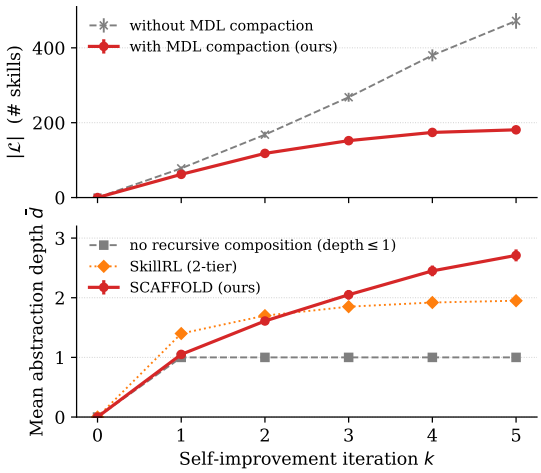

Figure 3: Library growth dynamics across iterations on WebArena. *Top*: total skill count without compaction (dashed) the library balloons; with compaction (solid) it stabilizes near 180 skills. *Bottom*: mean abstraction depth, which grows monotonically as recursive composition kicks in.

4.5 Iteration Scaling

A natural worry with any self-improvement scheme is that gains saturate quickly. Figure 2 plots SR against iteration k. Baselines that maintain a flat skill cache (AWM, SkillWeaver) plateau near k=2, after which library bloat slows further learning. SKILLRL continues to improve through k=3 but flattens thereafter. SCAFFOLD continues to improve through k=5, reflecting two interacting effects: distillation strengthens π_k, allowing it to solve harder tasks; and those harder tasks furnish new high-depth skills that, after compaction, expand the library's expressive reach without diluting it. We did not run beyond k=5 due to compute, but the slope at k=5 remains positive, suggesting further improvement is achievable.

4.6 Library Depth and Reuse

Figure 3 shows two complementary growth dynamics. The *top panel* contrasts library size with and without MDL compaction: without compaction $|\mathcal{L}|$ grows almost linearly to ≈ 470 skills by $k{=}5$, the majority being near-duplicates differing only in selectors or argument names (e.g., `login_v1`, `login_v2`, `login_with_email`); with compaction the library follows a concave trajectory and saturates near 181 skills, a $\sim 2.6\times$ reduction driven primarily by the *merge* and *prune* operators (jointly responsible for 76% of removed skills in our run logs). The *bottom panel* plots mean depth $\bar{d}$: SCAFFOLD climbs from 1.05 at $k{=}1$ to 2.71 at $k{=}5$ (max depth 5), while the - *recursive composition* ablation flatlines at $\bar{d}{=}1$ and SKILLRL's two-tier hierarchy plateaus at $\bar{d}{\leq}2$. Together with a reuse rate of 64% at $k{=}5$ (vs. $\approx 21\%$ without compaction; most skills singletons), these statistics support the view that composition and compaction are two halves of the same mechanism: composition produces the depth that makes long-horizon skills expressible, while compaction prevents that depth from being undermined by redundancy. The 3.1-point SR cost of disabling compaction (Table 2) is the empirical price of an uncontrolled library.

4.7 Cross-Site Transfer

Figure 4 compares site-to-site transfer for the two methods, beyond the granularity of domain split. Averaging over the 30 off-diagonal cells, SKILLWEAVER reaches 16.6% mean transfer SR versus SCAFFOLD's 35.9%, a 19.3-point gap; on the 15 cells under the three held-out sites in main experiments, SCAFFOLD stays in 27–47% while SKILLWEAVER falls to 11–28%, with representative pairs amazon→ziprecruiter (38% vs. 19%) and landwatch→ticketmaster (47% vs. 28%) sharing the search→filter→select→confirm structure but differing in DOM and layout. Inspecting the library shows the mechanism: generic procedural skills such as `filter_results(query, sort_by, max_price)`, `login(site, credentials)`, and `paginate_until(condition)` carry over via parameter re-binding, while SKILLWEAVER's opaque APIs encode selectors directly and must be rediscovered per site. The github row shows where this still leaves a gap. GitHub's issue/PR navigation is structurally unlike e-commerce flows, so SCAFFOLD's general skills transfer only modestly (27–31%) and SKILLWEAVER's effectively do not (11–14%). This is consistent with the transfer-ceiling limitation discussed in the *Limitations* section.

Method	Clean	Latency	DOM-shuf.	+7 d
SKILLWEAVER	29.1	25.3	24.7	26.5
SKILLRL	31.6	28.0	27.6	28.9
SCAFFOLD	**42.7**	**40.9**	**40.6**	**41.4**
Drop relative to clean (Δ in points)				
SKILLWEAVER	n/a	-3.8	-4.4	-2.6
SKILLRL	n/a	-3.6	-4.0	-2.7
SCAFFOLD	n/a	$\mathbf{-1.8}$	$\mathbf{-2.1}$	$\mathbf{-1.3}$

Table 3: Stability under environment perturbations on WebArena (SR %). *Clean*: unperturbed baseline. *Latency*: 0–3s random delay per action. *DOM-shuf.*: non-functional DOM attributes randomly permuted on each page load. *+7 d*: same task suite re-evaluated 7 days later. SCAFFOLD degrades by at most 2.1 points across all regimes, roughly half the sensitivity of the strongest baselines. Each cell is mean over 3 seeds (std ≤ 1.0).

4.8 Stability Under Stochasticity

Real web environments are noisy: DOMs change between page loads, A/B tests rotate elements, and servers throttle under load. To probe robustness we re-ran the WebArena evaluation under three perturbation regimes: (a) *Latency injection*: we add a uniformly sampled 0–3s delay before every action, simulating slow network conditions; (b) *DOM-attribute shuffling*: we randomly permute non-functional attributes (`id`, `class`, `data-*`) on each page load, breaking memorized selectors while preserving semantics; (c) *Temporal drift*: we re-run the same task suite 7 days later, capturing whatever natural drift WebArena's containerized sites exhibit between snapshots. Table 3 reports each method's success rate under all three regimes.

The pattern in Table 3 is consistent across all three perturbations: SCAFFOLD degrades by at most 2.1 points (DOM-shuffle), compared to 3.6–4.4 for SKILLWEAVER and SKILLRL, roughly half the sensitivity. We attribute this to two structural properties of our skills. First, *parametric grounding*: a skill that locates an element by semantic role ("the search input") is robust to attribute renaming, whereas a skill that memorized a specific `id` selector fails the moment that `id` is rewritten. Second, *postcondition checks*: even if the body of a skill misfires under latency or transient state, the postcondition test triggers a retry rather than blindly committing to a wrong final state. The DOM-shuffle column is where the gap is widest, which is exactly what these two mechanisms are designed to handle. We view this as evidence that

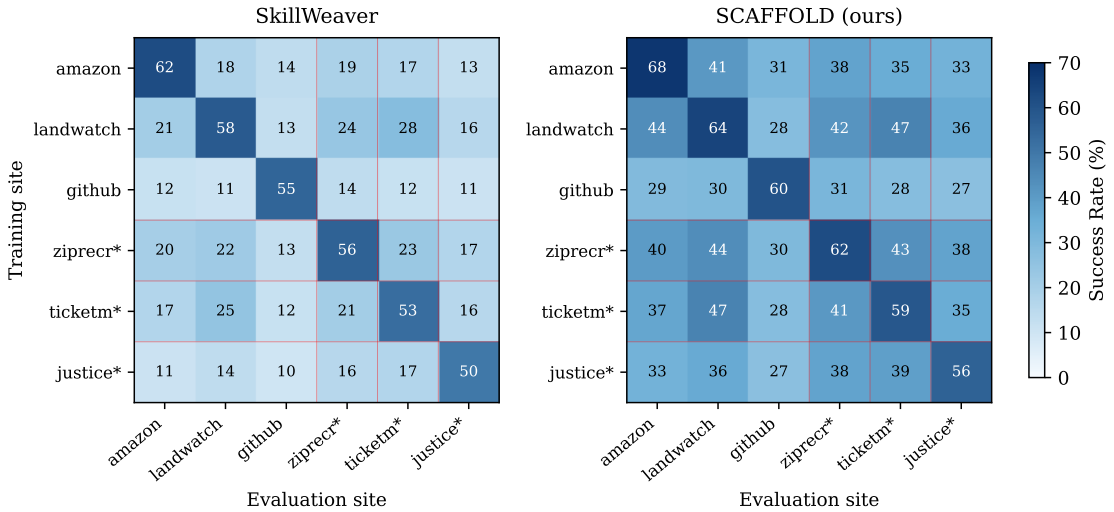

Figure 4: Cross-site transfer on Online-Mind2Web. Cell (i, j) shows SR (%) on test site j after training only on site i. The ziprecr and ticketm are the abbreviations for ziprecruiter and ticketmaster, respectively. The sites in redlines are the held-out test ones in main experiments, also indicated with *.

the structural choices in SCAFFOLD are not just accuracy-oriented but also confer a non-trivial robustness benefit at no additional inference cost.

4.9 Action Budget and Inference Cost

Because a skill invocation executes several primitive actions, one might worry that SCAFFOLD enjoys a larger effective action budget. It does not: the horizon is counted in primitive environment actions for every method, and primitives inside a skill body draw on the same budget (§4.1.4). Average primitive actions per successful task is 12.4 for SCAFFOLD against 14.3 for SKILLRL, 15.1 for SKILLWEAVER and 16.9 for AWM, so SCAFFOLD needs fewer actions, not more; capping the budget at 10/20/30 steps gives 11.5/24.0/29.1, 13.2/26.4/31.6 and 19.8/36.9/42.7 respectively, so the margin is widest at the tightest cap, the opposite of what budget inflation predicts.

Skills also trade expensive planning calls for cheaper ones: per successful task SCAFFOLD issues 5.3 policy calls against 9.6 for SKILLWEAVER and 18.7 for REACT, plus 7.1 far cheaper grounder and 4.6 judge calls, so total tokens fall from 224K and 118K to 79K and wall clock from 102s and 74s to 61s. Induction is offline and amortized (about 500 calls per run). Appendix F gives breakdown and a selector cache that removes 62% grounder calls.

4.10 Inducer and Verifier Dependence

Two components sit outside the loop. Swapping the GPT-4o inducer for the open-weight Qwen2.5-72B-Instruct with everything else fixed gives 40.8 SR on WebArena, 1.9 points below GPT-4o and still 9.2 above SKILLRL with the GPT-4o inducer, so the framework does not hinge on proprietary-API access. Replacing the ground-truth verifier with a model-based judge for filtering, following the WebJudge protocol of Xue et al. (2025), gives 40.2 on WebArena and 41.0 on OM2W-X against 42.7 and 43.5. Multi-instance induction and holdout re-execution filter much of this label noise, so SCAFFOLD degrades gracefully as verification weakens.

5 Conclusions and Future Work

In this paper, we presented SCAFFOLD, a self-improving framework for web agents that combines multi-instance parametric skill induction, recursive hierarchical composition, MDL-driven library compaction, and weight-level distillation into a single closed loop. On WebArena, VisualWebArena, and a held-out Online-Mind2Web split, SCAFFOLD outperforms the strongest skill-induction and skill-RL baselines by 11.1–17.2 absolute points while continuing to improve through five iterations and keeping the library size bounded. In the future, we can consider such directions: (i) replacing the GPT-4o-based inducer with a self-trained, lighter induction model to remove proprietary-API dependence; (ii) extending the framework to OSWorld-style full-desktop agents where the action space includes file I/O and shell commands; (iii) integrating an environment-grounded verifier that does not rely on per-task ground truth, enabling fully autonomous deployment; and (iv) developing a formal convergence analysis of the MDL-compaction operator under the recursive composition regime.

Limitations

We identify five concrete limitations of SCAFFOLD that we believe warrant attention. **(1) Inducer dependence.** Both skill induction and the refactor proposer of the MDL compactor rely on a strong proprietary model (GPT-4o); we estimate the inducer accounts for $\sim 70\%$ of total dollar cost in our experiments. Replacing it with a self-trained inducer is a clear next step but was outside our compute budget. **(2) Verifier dependence.** SCAFFOLD assumes access to a per-task verifier V to label trajectory success during the rollout stage. While this is realistic for WebArena and VisualWebArena (which ship deterministic verifiers), it does not extend to fully autonomous deployment in the wild, where success is rarely binary or self-evident. **(3) Greedy MDL approximation.** The compaction step optimizes Eq. 2 greedily; we have no convergence guarantee and observe occasional plateaus where a non-local refactor would further reduce $\mathcal{F}$. A more principled approximation, e.g., simulated annealing over skill rewrites, may help. **(4) Cross-site transfer ceiling.** Although parametric skills generalize meaningfully better than site-specific APIs (§4.7), our gains shrink when held-out sites differ qualitatively from training ones (e.g., training only on e-commerce, testing on government portals). The framework reuses *procedures*, not *web ontology*, so radical distributional shift remains hard. **(5) Distillation forgetting.** LoRA-based distillation preserves base capabilities reasonably well, but we observe a small (≤ 1.2 point) regression on a subset of held-out tasks that the original π_0 could already solve. A replay buffer of original task supervision could mitigate this; we leave a careful study to future work.

Ethical Considerations

SCAFFOLD enables web agents to learn reusable procedural skills through autonomous exploration. We highlight three ethical considerations. *Misuse*: more capable web agents can be deployed for spam, scraping at scale, credential stuffing, or evading rate limits. Our experiments use only WebArena, VisualWebArena, and a small Online-Mind2Web subset on permitted sites; we do not provide site-specific bypass skills. *Bias amplification through self-improvement*: any bias in the induced skills (e.g., always defaulting to certain payment methods or geographic regions) is reinforced across iterations through distillation. Future deployments should audit the skill library at each iteration. *Energy cost*: a full 5-iteration run on a single benchmark consumes $\sim$36 A100-hours. We report this transparently and discourage hyperparameter searches that re-run the full loop unnecessarily.

References

Zixiang Chen, Yihe Deng, Huizhuo Yuan, Kaixuan Ji, and Quanquan Gu. 2024. Self-play fine-tuning converts weak language models to strong language models. In *International Conference on Machine Learning*, pages 6621–6642. PMLR.

Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Li YanTao, Jianbing Zhang, and Zhiyong Wu. 2024. Seeclick: Harnessing GUI grounding for advanced visual GUI agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9313–9332.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. 2025. Mem0: Building production-ready AI agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413*.

Kevin Ellis, Catherine Wong, Maxwell Nye, Mathias Sablé-Meyer, Lucas Morales, Luke Hewitt, Luc Cary, Armando Solar-Lezama, and Joshua B Tenenbaum. 2021. DreamCoder: Bootstrapping inductive program synthesis with wake-sleep library learning. In *Proceedings of the 42nd acm sigplan international conference on programming language design and implementation*, pages 835–850.

Gaodan Fang, Vatche Isahagian, KR Jayaram, Ritesh Kumar, Vinod Muthusamy, Punleuk Oum, and Gegi Thomas. 2026. Trajectory-informed memory generation for self-improving agent systems. *arXiv preprint arXiv:2603.10600*.

Apurva Gandhi and Graham Neubig. 2026. Go-browse: Training web agents with structured exploration. In *International Conference on Learning Representations*, volume 2026, pages 138973–138992.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, and 1 others. 2025. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. 2024. 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)*, pages 6864–6890.

Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, and 1 others. 2024. CogAgent: A visual language model for GUI agents. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14281–14290. IEEE.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. Lora: Low-rank adaptation of large language models. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

Wenjia Jiang, Yangyang Zhuang, Chenxi Song, Xu Yang, Joey Tianyi Zhou, and Chi Zhang. 2025. AppAgentX: Evolving GUI agents as proficient smartphone users. *arXiv preprint arXiv:2503.02268*.

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)*, pages 881–905.

Hanyu Lai, Xiao Liu, Iat Long Iong, Shuntian Yao, Yuxuan Chen, Pengbo Shen, Hao Yu, Hanchen Zhang, Xiaohan Zhang, Yuxiao Dong, and 1 others. 2024. AutoWebGLM: A large language model-based web navigating agent. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 5295–5306.

Ajay Patel, Markus Hofmarcher, Claudiu Leoveanu-Condrei, Marius-Constantin Dinu, Chris Callison-Burch, and Sepp Hochreiter. 2024. Large language models can self-improve at web agent tasks. *arXiv preprint arXiv:2405.20309*.

Zehan Qi, Xiao Liu, Iat Long Iong, Hanyu Lai, Xueqiao Sun, Jiadai Sun, Xinyue Yang, Yu Yang, Shuntian Yao, Wei Xu, and 1 others. 2025. WebRL: Training LLM web agents via self-evolving online curriculum reinforcement learning. In *International Conference on Learning Representations*, volume 2025, pages 79791–79821.

Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, and 1 others. 2025. UI-TARS: Pioneering automated GUI interaction with native agents. *arXiv preprint arXiv:2501.12326*.

Jorma Rissanen. 1978. Modeling by shortest data description. *Automatica*, 14(5):465–471.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, and 1 others. 2024. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652.

Richard S Sutton, Doina Precup, and Satinder Singh. 1999. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2024a. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*.

Haoming Wang, Haoyang Zou, Huatong Song, Jiazhan Feng, Junjie Fang, Junting Lu, Longxiang Liu, Qinyu Luo, Shihao Liang, Shijue Huang, and 1 others. 2025a. UI-TARS-2 technical report: Advancing GUI agent with multi-turn reinforcement learning. *arXiv preprint arXiv:2509.02544*.

Jiongxiao Wang, Qiaojing Yan, Yawei Wang, Yijun Tian, Soumya Smruti Mishra, Zhichao Xu, Megha Gandhi, Panpan Xu, and Lin Lee Cheong. 2026. Reinforcement learning for self-improving agent with skill library. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1529–1550.

Zhiruo Wang, Graham Neubig, and Daniel Fried. 2024b. Trove: Inducing verifiable and efficient toolboxes for solving programmatic tasks. In *International Conference on Machine Learning*, pages 51177–51191. PMLR.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. 2025b. Agent workflow memory. In *International Conference on Machine Learning*, pages 63897–63911. PMLR.

Rong Wu, Xiaoman Wang, Jianbiao Mei, Pinlong Cai, Daocheng Fu, Cheng Yang, Licheng Wen, Xuemeng Yang, Yufan Shen, Yuxin Wang, and 1 others. 2025a. EvolveR: Self-evolving LLM agents through an experience-driven lifecycle. *arXiv preprint arXiv:2510.16079*.

Zhiyong Wu, Zhenyu Wu, Fangzhi Xu, Yian Wang, Qiushi Sun, Chengyou Jia, Kanzhi Cheng, Zichen Ding, Liheng Chen, Paul Pu Liang, and 1 others. 2025b. Os-atlas: Foundation action model for generalist GUI agents. In *International Conference on Learning Representations*, volume 2025, pages 5090–5108.

Peng Xia, Jianwen Chen, Hanyang Wang, Jiaqi Liu, Kaide Zeng, Yu Wang, Siwei Han, Yiyang Zhou, Xujiang Zhao, Haifeng Chen, and 1 others. 2026. Skillrl: Evolving agents via recursive skill-augmented reinforcement learning. In *ICLR 2026 Workshop on Lifelong Agents: Learning, Aligning, Evolving*.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, and 1 others. 2024. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094.

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

Zishan Xu, Yifu Guo, Yuquan Lu, Fengyu Yang, Zhiyuan Yao, Jiaye Lin, Ruyi Gong, and Lihua Cai. 2026. Skillevo: An experience learning framework with reinforcement learning for skill evolution.

Tianci Xue, Weijian Qi, Tianneng Shi, Chan Hee Song, Boyu Gou, Dawn Song, Huan Sun, and Yu Su. 2025. An illusion of progress? assessing the current state of web agents. *arXiv preprint arXiv:2504.01382*.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, and 1 others. 2024. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*.

Ke Yang, Yao Liu, Sapana Chaudhary, Rasool Fakoor, Pratik A Chaudhari, George Karypis, and Huzefa Rangwala. 2025. AgentOccam: A simple yet strong baseline for LLM-based web agents. In *International Conference on Learning Representations*, volume 2025, pages 97533–97565.

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

Simon Yu, Gang Li, Weiyan Shi, and Peng Qi. 2026. Polyskill: Learning generalizable skills through polymorphic abstraction for continual learning. In *International Conference on Learning Representations*, volume 2026, pages 140298–140326.

Eric Zelikman, Georges Harik, Yijia Shao, Varuna Jayasiri, Nick Haber, and Noah D Goodman. 2024. Quiet-STaR: Language models can teach themselves to think before speaking. *arXiv preprint arXiv:2403.09629*.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. 2022. STaR: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, 35:15476–15488.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. 2024. ExpeL: LLM agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19632–19642.

Boyuan Zheng, Michael Y Fatemi, Xiaolong Jin, Zora Zhiruo Wang, Apurva Gandhi, Yueqi Song, Yu Gu, Jayanth Srinivasa, Gaowen Liu, Graham Neubig, and 1 others. 2025. Skillweaver: Web agents can self-improve by discovering and honing skills. *arXiv preprint arXiv:2504.07079*.

Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. 2024a. Gpt-4v (ision) is a generalist web agent, if grounded. In *International Conference on Machine Learning*, pages 61349–61385. PMLR.

Longtao Zheng, Rundong Wang, Xinrun Wang, and Bo An. 2024b. Synapse: Trajectory-as-exemplar prompting with memory for computer control. In *International Conference on Learning Representations*, volume 2024, pages 19036–19066.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, and 1 others. 2024. WebArena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, volume 2024, pages 15585–15606.

A Prompt Templates

We provide the three core prompts used by SCAFFOLD below. Full prompts (including system messages and few-shot examples) are released with the code.

Skill inducer (multi-instance abstraction). Given a cluster $C = \{\zeta_1, \ldots, \zeta_n\}$ of $n \geq 2$ successful trajectories with embedding-similar instructions, the inducer is prompted to produce a parameterized skill. The prompt structure is:

> `You are given` n `successful agent trajectories that accomplish similar goals on a website. Your task is to abstract them into a single parametric, executable skill.`
>
> `[Trajectory 1: instruction, action sequence, screenshots] . . .`
>
> `[Trajectory n: instruction, action sequence, screenshots]`
>
> `Available existing skills (you may call them):` $\{\mathcal{L}_k\}$
>
> `Output a Python function with: (a) a typed parameter list capturing what varies across trajectories; (b) a precondition expressed as a predicate over observation obs; (c) a body using primitives {click, type, scroll, wait} or any existing skill; (d) a postcondition.`

The model is then asked to justify its choice of parameters by pointing to specific token positions in

each trajectory that vary, which improves abstraction quality and gives us a cheap diagnostic when induction fails.

Refactor proposer (MDL compaction). Given a recurring primitive-action subsequence detected across $\geq r_{\min}$ existing skills, the refactor proposer is asked whether the subsequence should be promoted to its own skill:

> The following action subsequence appears in m existing skills (shown below). Decide whether to refactor it into a new mid-level skill. If yes, name it, parametrize it, and rewrite all m existing skills to call the new one. If no, briefly explain why (e.g., the subsequence is too short or too site-specific to be reusable).

We accept the proposal only if (i) the MDL functional $\mathcal{F}$ decreases and (ii) the rewritten skills pass behavioral-equivalence checks on a held-out trajectory set.

Pre/Postcondition validator. At runtime, before invoking a skill we verify its precondition by passing $(\text{obs}_t, \text{pre}_\sigma)$ to a lightweight VLM judge with a yes/no output. Same protocol for postconditions after execution. The judge is a separate, cheaper model from the inducer to keep inference cost bounded.

Trajectory clustering. Instructions are embedded with `gte-large-en-v1.5`. Trajectories are first partitioned by site and then clustered by agglomerative clustering with average linkage on cosine similarity, using a merge threshold of 0.82. Clusters larger than 8 trajectories are subsampled by proximity to the centroid so that the inducer context stays bounded, and singleton clusters (below $n_{\min}$) are held in a buffer and re-clustered in later iterations as semantically similar trajectories accumulate, so no successful trajectory is permanently discarded. Sensitivity to the threshold is mild, within 1 SR point across the range 0.78 to 0.86.

B A Worked Three-Level Recursive Skill

To make the recursive composition concrete, we trace the construction of a depth-3 skill `checkout_cheapest_in_category` on the E-commerce site of WebArena, induced at iteration $k{=}3$.

Level 1 (depth $d{=}1$): atomic skills, induced at $k{=}1$.

```
def click_text(obs, text: str):
   pre:  any(e.text == text for e in obs.dom)
   body: el = find(obs.dom, text=text)
         primitive_click(el.bbox)
   post: page changed OR el.state == 'pressed'

def type_in_field(obs, label: str, value: str):
   pre:  exists_input(obs.dom, label)
   body: f = find_input(obs.dom, label)
         primitive_click(f.bbox)
         primitive_type(value)
   post: f.value == value
```

Level 2 (depth $d{=}2$): mid-level skills, induced at $k{=}2$.

```
def search_and_filter(obs, query: str,
                      category: str,
                      sort: str = 'price_asc'):
  pre: url contains 'shop' AND has_search_box(obs)
   body: type_in_field(obs, 'Search', query)
         click_text(obs, 'Search')
         click_text(obs, category)
         click_text(obs, f'Sort: {sort}')
   post: results visible AND sort_label == sort
```

Level 3 (depth $d{=}3$): composite skill, induced at $k{=}3$.

```
def checkout_cheapest_in_category(
        obs, query: str, category: str,
        payment_token: str):
   pre:  logged_in(obs)
   body: search_and_filter(obs, query, category,
                           sort='price_asc')
         click_text(obs, 'first_result')
         click_text(obs, 'Add to Cart')
         click_text(obs, 'Checkout')
         type_in_field(obs, 'PaymentToken',
                       payment_token)
         click_text(obs, 'Place Order')
   post: order_confirmed(obs) AND
         purchased_item.category == category
```

The depth-3 skill calls one depth-2 skill (`search_and_filter`) and two depth-1 skills (`click_text`, `type_in_field`). At iteration $k{=}4$, the inducer further abstracts this skill into a depth-4 `restock_pantry(items, budget)` that calls `checkout_cheapest_in_category` in a loop, illustrating how depth grows organically across iterations.

C Per-Site Analysis

Table 4 breaks down performance by the canonical site grouping used in each benchmark. WebArena's four primary sites cover e-commerce (OneStopShop), social-forum discussions (Reddit), collaborative software development (GitLab), and content management (CMS) (Zhou et al., 2024); VisualWebArena spans three live-style sites (Classifieds, Reddit, Shopping) (Koh et al., 2024). SCAFFOLD's gains are largest on *GitLab* and *CMS*, the WebArena sites with the longest action horizons, dominated by multi-step configuration and code-collaboration tasks. The gain is also obvious

Site	SkillRL	SCAFFOLD	Δ
WebArena			
OneStopShop	30.5	42.6	+12.1
Reddit	33.2	41.9	+8.7
GitLab	28.4	45.1	**+16.7**
CMS	27.0	42.9	+15.9
VisualWebArena			
Classifieds	22.6	32.1	+9.5
Reddit	24.3	35.8	+11.5
Shopping	21.2	41.0	**+19.8**

Table 4: Per-site success rate (%) on WebArena (Zhou et al., 2024) (four primary sites) and VisualWebArena (Koh et al., 2024). SCAFFOLD's largest per-site gains occur on long-horizon, multi-step sites (GitLab, CMS, VWA Shopping), where recursive composition contributes most. Differences are bootstrap-significant ($p < 0.05$) for all rows.

on VWA's *Shopping*, where parametric search/filter/checkout skills compose well. Gains on *Reddit* are more modest, consistent with social-forum tasks being on average shorter and more retrieval-oriented, where flat libraries already perform well.

Qualitative diagnoses. Inspecting failure modes shared by both methods reveals two patterns. First, *verifier brittleness*: a non-trivial fraction of WebArena tasks reward only exact-string answers, so a semantically correct trajectory can be marked wrong; this affects both systems equally. Second, *visual occlusion under modals*: when a cookie banner or login dialog covers the target element, both agents struggle; here SCAFFOLD's precondition checks help but do not fully solve the problem, suggesting that an explicit modal-dismissal mid-level skill would be a high-value addition to the library.

D Hyperparameter Sensitivity

We probe sensitivity to the four SCAFFOLD-specific hyperparameters; full sweep results in the supplementary material. Table 5 shows that the method is broadly robust: performance varies by ≤ 2.4 SR points across the tested ranges, with the multi-instance threshold $n_{\min}$ being the most sensitive (smaller values admit spurious skills; larger values delay skill creation).

E Supervision Format and Component Build-Up

Table 2 removes one component at a time from the full system; Table 7 takes the opposite direction and adds one component at a time on top of SKILLWEAVER, which makes the marginal value of each component visible in isolation from the others. The two views agree, and together they support reading the four components as one mechanism rather than four independent add-ons: induction is the abstraction phase, MDL compaction the compression phase, distillation the consolidation phase, and multi-instance validation is what keeps abstraction sound, in the spirit of wake-sleep library learning (Ellis et al., 2021). The components also interact by design. Figure 3 shows that composition creates the depth that makes long-horizon skills expressible while compaction keeps that depth from being buried in redundancy, and §3.5 explains why compaction is also what protects the distillation signal from dilution. All configurations hold the base model, task pool, iteration count, inducer, and evaluation protocol fixed, so the differences cannot be attributed to engineering scale or extra compute.

F Efficiency and Grounding Measurements

Tables 8 and 9 report the two measurements summarised in §4.9: success rate under a capped primitive-action budget, and the per-task call, token and wall-clock breakdown. Charging SCAFFOLD one extra primitive step for every skill invocation, a pessimistic accounting of invocation overhead, still leaves it at 42.3. The rest of this appendix asks how far the residual model calls inside a skill body can be removed.

Semantic grounding is the one place where SCAFFOLD calls a model inside a skill body, so it is worth asking how much of it can be replaced by cheaper machinery such as regular expressions over the accessibility tree. We implement a hybrid mode with *selector memoization*: the first successful grounding of a semantic reference caches the resolved accessibility-tree path together with a structural signature (role, tag, text pattern), keyed by site, skill and parameter; subsequent invocations first try the cached selector with a cheap regex and accessibility-tree validation, and fall back to VLM grounding only on mismatch. Table 10 compares the three regimes. The hybrid retains clean and perturbed accuracy while removing most grounder calls, and the static-selector variant makes the trade-off visible: it is the cheapest and the most brittle, losing 7.1 points under DOM-shuffle, which confirms that runtime semantic grounding is the source of the robustness gap in Table 3. One caveat on the premise that skills are site-specific: after com-

Hyperparameter	Range tested	ΔSR (WA)
Multi-instance threshold $n_{\min}$	$\{1, 2, 3, 4\}$	-3.1 / 0.0 / -0.8 / -2.4
Compaction interval M	$\{1, 2, 3, 5\}$	-0.6 / 0.0 / -0.4 / -1.7
Equivalence threshold ρ	$\{0.7, 0.8, 0.9, 0.95\}$	-1.2 / -0.3 / 0.0 / -0.4
LoRA rank for distillation	$\{16, 32, 64, 128\}$	-1.6 / -0.5 / 0.0 / -0.2

Table 5: Sensitivity to SCAFFOLD's main hyperparameters. ΔSR is relative to the default setting (bold values in the experiments section: $n_{\min}$=2, M=2, ρ=0.9, rank=64).

Training signal (same trajectories, K=5)	WA
No SFT, in-context library only (Tab. 2)	34.7
Standard SFT, trajectories flattened to primitives	38.6
Plan-augmented SFT, no auxiliary skill-name loss	41.6
Full SCAFFOLD distillation	**42.7**
STaR-style loop: SFT on raw trajectories, no library	30.4

Table 6: Supervision-format study on WebArena (SR %). All rows share the base model, the rollout pipeline and the exact same successful trajectories, differing only in how those trajectories supervise the policy.

Configuration	WA
SKILLWEAVER	29.1
+ multi-instance induction with holdout validation	31.9
+ recursive composition	34.0
+ distillation	39.6
+ MDL compaction (= full SCAFFOLD)	**42.7**

Table 7: Incremental build-up on WebArena (SR %), starting from SKILLWEAVER and adding one component at a time. Every addition is bootstrap-significant ($p < 0.05$). The intermediate point 39.6 coincides with the "− MDL compaction" row of Table 2 by construction.

Max primitive steps	10	20	30
SKILLWEAVER	11.5	24.0	29.1
SKILLRL	13.2	26.4	31.6
SCAFFOLD	**19.8**	**36.9**	**42.7**

Table 8: WebArena SR (%) under a capped primitive-action budget (§4.9). The margin is largest at the tightest cap, the opposite of what an inflated action budget would produce.

Method	Pol.	Grd.	Jdg.	Tok. (K)	Time (s)
REACT	18.7	0	0	224	102
SKILLWEAVER	9.6	0	0	118	74
SCAFFOLD	**5.3**	7.1	4.6	**79**	**61**

Table 9: Inference cost per successful WebArena task: policy, grounder and judge calls, total tokens and wall clock. A policy call costs about 12K tokens (screenshot, DOM and history), a grounder call about 1.5K, and a judge call about 0.8K.

paction the most valuable skills are not, given the 64% reuse rate and the transfer results of §4.7, and those skills cannot be bound to any single site's selectors. The cache gives them per-site fast paths while preserving transfer.

G Skill Lifecycle and Distillation Hygiene

This appendix expands the two mechanisms sketched in §3.5 and §3.6.

Runtime failure handling. Preconditions gate invocation: when a precondition is unmet, for instance because an unexpected pop-up occludes the target, the skill is simply not fired and the policy continues with primitives or an alternative skill. Postconditions are checked after execution; a violation triggers one retry after a state refresh, after which control returns to the policy with the failure noted in context so that the episode can still recover. All failures are logged together with the observation and feed the next induction round.

Outdated or degrading skills. Beyond the prune operator, which removes skills unused for M iterations, the implementation tracks a rolling execution success rate per skill. A skill that falls below 60% over its last 20 invocations is quarantined out of the retrieval index and queued for re-induction from fresh trajectories, which naturally repairs drift because new trajectories reflect the updated page. WebArena's containerized sites drift little, as the +7 d column of Table 3 shows, so this mechanism matters mainly for live deployment.

Modal occlusion. Following the diagnosis in Appendix C, we added an explicit `dismiss_blocking_modal` mid-level skill in a separate run: WebArena SR improves by 0.9 points and modal-related failures fall from 6.1% to 2.3% of episodes. This is a case where reading the library's failure log suggests the missing abstraction directly.

Distillation hygiene. Three properties keep supervision current. Trajectories are re-parsed against the post-compaction library before training, as de-

Grounding mode	Clean	DOM-shuf.	Grd./task
VLM grounding (main paper)	**42.7**	**40.6**	7.1
Hybrid with selector cache	42.5	40.1	2.7
Static selectors only	41.2	33.5	**0**

Table 10: Grounding modes on WebArena (SR %) and grounder calls per successful task. The hybrid cache removes 62% of grounder calls at essentially no cost in accuracy or robustness, whereas static selectors reintroduce baseline brittleness under DOM-shuffle.

scribed in §3.6, so the model is never supervised toward deprecated skills; the auxiliary skill-name loss uses the current namespace as its label space, so retired names cannot be reinforced; and each iteration trains on freshly collected trajectories rather than accumulating stale data. Consistent with Limitation (5), we still observe at most a 1.2-point regression on previously solved tasks, and a small replay buffer of re-parsed earlier trajectories removes most of it in a preliminary run.

H Comparison with Concurrent Frameworks

Table 11 places SCAFFOLD beside POLYSKILL (Yu et al., 2026) and SKILLEVO (Xu et al., 2026) along the four axes that motivate our design. POLYSKILL shares our generalization goal and reaches it by maintaining per-site implementations under a shared interface, whereas our skills keep one implementation and re-bind semantic references at run time; its library is prompt-side only, grows without a compression objective, and induces skills without a multi-instance support requirement. SKILLEVO improves the policy with GRPO against a learned reasoning-and-execution reward model, which is closer in spirit to SKILLRL than to our SFT-based consolidation, and its skill path graph evolves but is not governed by a compression objective with behavioral-equivalence merging and a validation safety net.

Its reported numbers are also not directly comparable to Table 1, since it is evaluated with text-only LLMs (Llama-3.1-8B, GLM-4-9B) on WebArena-Lite (165 tasks), while we target visual web agents on full WebArena (812 tasks), VisualWebArena and live websites. As a reference point we ran SCAFFOLD on WebArena-Lite under our visual setting and obtained 63.8 SR with Qwen2.5-VL-7B, against the 60.4 reported for SKILLEVO with Llama-3.1-8B, with the explicit caveat that backbone and observation space differ. Finally, the two lines of work are complementary rather than competing: POLYSKILL's interface typing could serve as the type system for our parameter lists, and SKILLEVO's fine-grained reward model could replace our binary success filter, which would relax the verifier dependence discussed in §4.10.

Aspect	POLYSKILL	SKILLEVO	SCAFFOLD
Skill unit	abstract interface plus per-site implementations	skill paths in a graph	parametric executable program with pre/postconditions
Composition	compositional, prompt-side	composite skills over the skill path graph	recursive, unbounded depth (5 observed)
Library governance	none	graph updates, no compression objective	MDL merge / refactor / prune with validation safety net
Policy weights	frozen	GRPO with a learned reward model	SFT distillation of skill plans
Setting	HTML agents, Mind2Web / WebArena	text agents, WebArena-Lite	visual agents, WA / VWA / OM2W

Table 11: Component-level comparison with the two concurrent skill-centric frameworks discussed in §2.