Beyond Domains: Reusing Web Skills via Transferable Interaction Patterns

Shiqi He[1]* **Yue Cui**[2] **Feijie Wu**[3] **Xinyu Ma**[4]
Jiaheng Lu[5] **Yaliang Li**[2] **Bolin Ding**[2] **Mosharaf Chowdhury**[1]

[1]University of Michigan [2]Alibaba Token Hub, Alibaba Group
[3]Purdue University [4]McMaster University [5]University of Pennsylvania

Abstract

Large language model (LLM) web agents are usually deployed as *tool callers*: each turn, the model reads a fresh page observation and emits one structured tool action. When every action is a low-level primitive, horizons grow quickly and so do policy-facing LLM completions, dominating latency and cost on benchmarks such as Mind2Web and WebArena. Recent systems therefore wrap repeated interaction fragments as *web skills*: callable tools built from successful trajectories or induced programs, so one call can replace several primitives. However, prior skill libraries are still triggered mainly by instruction similarity or coarse site metadata, which yields low *skill reuse* on held-out sites and leaves much of the potential step and token reduction on the table.

We present SKILLMIGRATOR, an agent that learns reusable web skills and transfers them across sites by matching layout structure rather than specific element references. Each induced skill is stored as a *transferable interaction pattern* (TIP): the skill paired with a structural sketch of the snapshot at induction time. At test time, SKILLMIGRATOR retrieves TIPs by layout similarity and grounds their references on the live page. The rest of the stack is standard: accessibility-snapshot observations with stable references, and fixed tool calling over primitives plus skill invocations. Compared with the state-of-the-art approaches, SKILLMIGRATOR reduces the average LLM-action count on successful trajectories by 8–10% across both WebArena and Mind2Web.

1 Introduction

Web agents translate a user's natural-language goal into a sequence of browser actions such as searching, clicking, typing, and submitting forms, providing a general interface for automating tasks that are difficult to script manually. Recent benchmarks such as WebShop [1], Mind2Web [2], and WebArena [3] cover e-commerce interaction, open-domain websites, and realistic self-hosted environments, highlighting both the practical value and the difficulty of this setting. However, most existing web agents rely on an LLM-centered decision loop that repeatedly queries the LLM to predict the next action from the current webpage state, often following reasoning-and-acting paradigms such as ReAct [4]. This design is flexible but expensive at deployment, since each task may require many sequential LLM calls, with cost and latency growing in the length and number of interaction trajectories [5]. There is therefore a need for a *cost-effective web agent* that reduces reliance on the LLM.

Recent work pursues this goal through reusable web skills, which store procedural knowledge from prior web interactions and reapply it in future tasks [6–10]. A successful interaction trajectory is

*Correspondence to: shiqihe@umich.edu. Work done during an internship at Alibaba Group.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

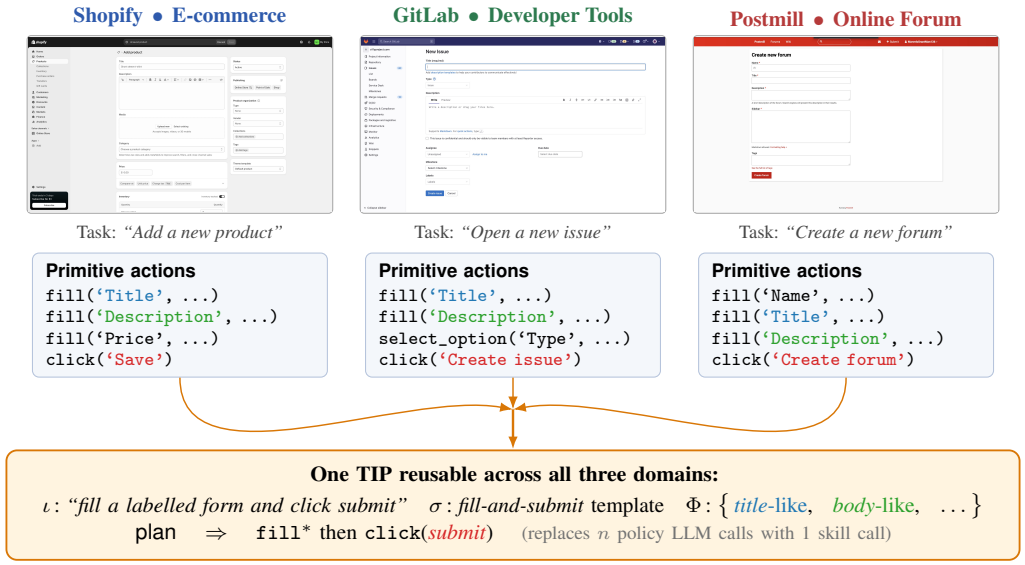


Figure 1: **Cross-domain skill reuse motivates SKILLMIGRATOR.** Three websites drawn from very different domains—*Shopify* (e-commerce), *GitLab* (developer tools), and *Postmill* (online forum)—use different page layouts, field vocabularies, and submit-button labels. Yet the three subtasks reduce to the same programmatic pattern: fill a few labelled inputs, then click a single submit button. Same-colour fields (*title*-like, *body*-like, *submit*) are paraphrases of the same abstract slot across all three sites. SKILLMIGRATOR stores *one* TIP—intent ι, operation template σ, slot schema Φ, and the induction-time tree skeleton τ—and reuses it on all three pages, replacing many policy LLM steps with a single skill call.

abstracted into a skill that the agent can retrieve and execute when it encounters a similar goal or webpage state. This replaces many primitive LLM decisions with a single higher-level operation, reducing LLM calls and shortening the interaction trajectory. Because the skill encodes a validated action pattern, it also mitigates compounding errors in long-horizon navigation [7–9, 11].

Existing skill-reuse methods fall into two types: reusing web skills under *the same website*, where the skill is specialized to a specific site and its interface [8–10], and reusing web skills *under the same domain*, where the skill is transferred across websites that share similar task structures, such as shopping, maps, forums, or code repositories [6, 7, 12]. Both directions have important limitations. Same-website methods suffer from low reuse rates, because each skill is tied to the specific interface, DOM structure, and interaction pattern of the site where it was learned [8–10]. The agent benefits only when future tasks revisit that site, which is restrictive in open-web settings where user requests span diverse websites. Same-domain methods such as PolySkill [12] address this through polymorphic abstractions that separate shared skill interfaces from site-specific implementations, enabling reuse across sites within a domain. However, they still confine reuse to a single domain and miss the fact that sites across different domains often share strong interaction patterns, as illustrated in Figure 1. As a result, current skill-reuse methods remain narrower than necessary, motivating a more general form of reusable web skills that can transfer beyond both *the same website* and *the same domain*.

Obtaining reusable web skills beyond both the same website and the same domain is challenging because of unreliable skill retrieval. Existing methods retrieve candidates using instruction similarity, intent labels, or website metadata [6, 8, 12]. However, these signals are insufficient for cross-domain transfer: two tasks with different wording may require the same interaction program, while two textually similar tasks may require different DOM-level control flow. As a result, the agent may fail to retrieve useful skills or mistakenly execute unsuitable ones, forcing it to fall back to primitive action generation with frequent LLM calls.

We propose SKILLMIGRATOR, a cost-effective web agent that enables skill reuse beyond both *the same website* and *the same domain*. SKILLMIGRATOR follows the standard programmatic-skill setting, where observations are accessibility snapshots with stable element references and actions are issued through a fixed tool-calling API over primitive actions and skill invocations [8, 9, 12]. Its memory unit is a transferable interaction pattern (TIP) , which pairs each induced skill with a structural

sketch of the webpage snapshot where the skill was validated. At inference time, SKILLMIGRATOR retrieves skills from a single global library by combining layout similarity with text signals, then grounds the matched abstract constraints to live element references before replaying the skill. This design allows the agent to identify reusable interaction patterns across websites with different wording, interfaces, and domains, while avoiding the execution of weakly matched skills.

Contributions.

- To our knowledge, this is the first work to study reusable web skills across websites beyond domains. This setting is non-trivial because similar interaction patterns may appear with different layouts, labels, and DOM structures.
- We propose SKILLMIGRATOR[2] for cross-domain skill matching. It stores induced skills as TIPs, each pairing a validated skill with the structural sketch of its source webpage. For a new task, SKILLMIGRATOR retrieves relevant TIPs using layout and text signals, grounds them to live webpage elements, and falls back to primitive control when no reliable match is found.
- We empirically compare SKILLMIGRATOR against existing web-agent baselines on Mind2Web and WebArena, reducing average LLM-action count on successful trajectories by about 8–10% relative to the state-of-the-art baselines.

2 Background and Motivation

2.1 Preliminaries and Problem Formulation

Web Agent Environment. We follow the BrowserGym and WebArena convention [13, 3]: at each timestep t, the agent receives a Playwright-style accessibility snapshot o_t with stable refs, roles, names, and state attributes, and emits one tool call $a_t \sim \pi_\theta(\cdot \mid q, o_{0:t}, a_{0:t-1})$ from the action space $\mathcal{A}$ in Table 1. For benchmarks, we consider Mind2Web [2] (cross-task, cross-website, cross-domain splits) and WebArena [3] (812 executable tasks across shopping, admin, reddit, gitlab, map, multisite). The default policy input is text-only and screenshot-grounded or pixel-space agents are outside our scope [14, 15]. A full formulation is in Appendix A.

Table 1: Representative primitive tools. e is an element ref in the text snapshot. Skill calls expand into grounded primitive calls.

Action	Description
`noop`	Wait or synchronize.
`click(`e`)`	Activate control e.
`fill(`e, *text*`)`	Type into an input.
`press(`*key*`)`	Press a key or key combination.
`scroll(`e, *dir*`)`	Scroll element or viewport.
`select_option(`e, *value*`)`	Choose an option.
`tab_focus(`*index*`)`	Switch browser tab.
`go_back` / `go_forward`	History navigation.
`send_msg_to_user(`*text*`)`	Final user-visible answer.
`stop`	Terminate with success or failure.
`call_skill(`σ`,` *args*`)`	Invoke skill σ and execute its implementation.

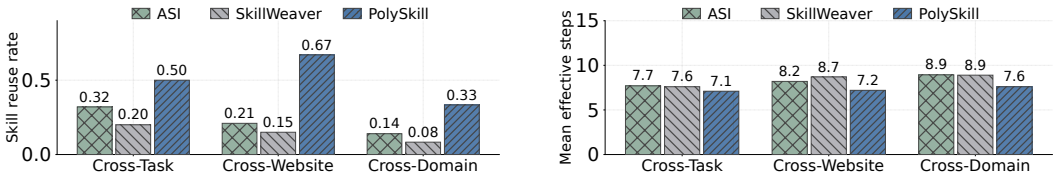
Skill library. Beyond primitive browser actions, recent web agents [7, 8, 12] are equipped with a skill library $\mathcal{K}$. Each skill $k \in \mathcal{K}$ is a temporally extended routine [16] that maps a subtask s and observation o to a short action sequence $k(s, o) = \langle \tilde{a}_1, \ldots, \tilde{a}_n \rangle$ over Table 1, exposed to the policy as a callable high-level macro, which reuses recurring interaction patterns such as opening a menu, filling a form, searching, or filtering.

Problem Formulation. Given an instruction q, a planner decomposes it into a sequence of subtasks $\mathbf{s}(q) = \{s_1, \ldots, s_{T_q}\}$. Let $\tilde{o}_0$ be the initial observation and $\tilde{o}_i$ the observation after finishing s_i. We define $\mathcal{N}(s, o \mid \mathcal{K}, \pi_\theta)$ as the number of *new* primitive actions emitted by π_θ to complete s from o: if the subtask is fully covered by a retrieved skill, $\mathcal{N} = 0$. Otherwise the agent falls back to π_θ and $\mathcal{N} = n$ where n is the length of the fallback trajectory. Our goal is to construct a compact skill library that minimises the expected number of LLM-generated primitive actions across tasks:

$$\min_{\mathcal{K}} \mathbb{E}_{q \sim Q} \left[\sum_{i=1}^{T_q} \mathcal{N}(s_i, \tilde{o}_{i-1} \mid \mathcal{K}, \pi_\theta) \right] + \lambda\, C(\mathcal{K}), \tag{1}$$

where Q is the task distribution, $C(\mathcal{K})$ is the library cost, and λ trades off action savings against library size.

[2]Code is available at `https://github.com/SymbioticLab/SkillMigrator`.



(a) Skill reuse rate.

(b) Mean effective steps.

Figure 2: Motivating comparison of ASI, SkillWeaver, and PolySkill on a cumulative Mind2Web-style setup: cross-task (same website), cross-website (same domain, new sites), and cross-domain (new domains). Subfigure a reports per-phase skill reuse and b shows successful-step cost.

2.2 Skill Induction Methods

Existing Work. Recent work on reusable procedural knowledge for web and tool-use agents includes textual workflow memory (AWM [6]), verified programmatic skills callable as high-level actions (ASI [7]), self-induced skill APIs from exploration (SkillWeaver [8]), polymorphic cross-site abstractions (PolySkill [12]), and discovered website tools (WALT [9]). Together, these methods compress repeated primitive sequences into reusable abstractions, shortening interaction horizons and reducing policy LLM calls.

Limitations. Despite these advances, skill retrieval remains a bottleneck on unseen websites because existing methods rely on semantic keys like task descriptions, workflow summaries, skill names and API descriptions. Web tasks often preserve interaction structure while changing surface wording, so purely semantic retrieval under-retrieves reusable skills—raising $\mathcal{N}(s, o \mid \mathcal{K}, \pi_\theta)$—and over-retrieves skills whose execution context is incompatible with the current page. Figure 2 illustrates this on a 60-task Mind2Web subset evaluated with GPT-4.1: skill reuse drops sharply for ASI, SkillWeaver, and PolySkill as the test trace moves from cross-website to cross-domain (Figure 2a), and average successful-step cost rises in lock-step (Figure 2b). This motivates augmenting retrieval with a layout-conditioned signal that uses the current observation as additional evidence for reuse beyond semantic similarity.

3 Design

SKILLMIGRATOR reduces $\mathcal{N}(s_i, \tilde{o}_{i-1} \mid \mathcal{K}, \pi_\theta)$ in the objective of §2 by reusing skills in $\mathcal{K}$ across websites and domains whose surface wording and labels differ. At each pair $(s_i, \tilde{o}_{i-1})$, the agent (i) summarises the live snapshot and retrieves the most similar skill from $\mathcal{K}$ (§3.1), (ii) parses one *value* for each slot of that skill from the user's instruction, and (iii) binds each slot to a concrete control on the live page so that the skill expands into grounded primitive actions without an LLM call (§3.2). Figure 4 summarises this runtime control flow.

3.1 Skill Record and Retrieval

Snapshot. At time t, the observation o_t is a Playwright accessibility snapshot, a YAML-like serialization of the agent-visible accessibility tree [13]. Each node exposes a semantic role (`textbox`, `button`, `link`, `heading`, ...), an accessible name, and a stable agent-addressable reference `ref=eN` that primitive actions use as their target. A deterministic rule scans o_t once and produces (i) a one-line *page summary* $\rho(o_t)$ that lists the page heading, the labelled controls inside the principal form, and the primary button labels, and (ii) the list $V(o_t)$ of all interactive nodes with their roles and names.

Skill record. Rather than storing a literal action recipe, which would not survive relabelling on a new site, each induced skill is stored as a transferable interaction pattern (TIP)—a tuple

$$k \;=\; \big(\, \iota_k,\; \sigma_k,\; \Phi_k,\; \tau_k \,\big), \tag{2}$$

where ι_k is a one-sentence natural-language *intent*, σ_k is the skill's *operation template*, drawn from a set Σ of templates mined offline by clustering training trajectories on action shape (e.g. a single `fill` on a `searchbox` followed by a submit, or several `fills` followed by a labelled `button`),

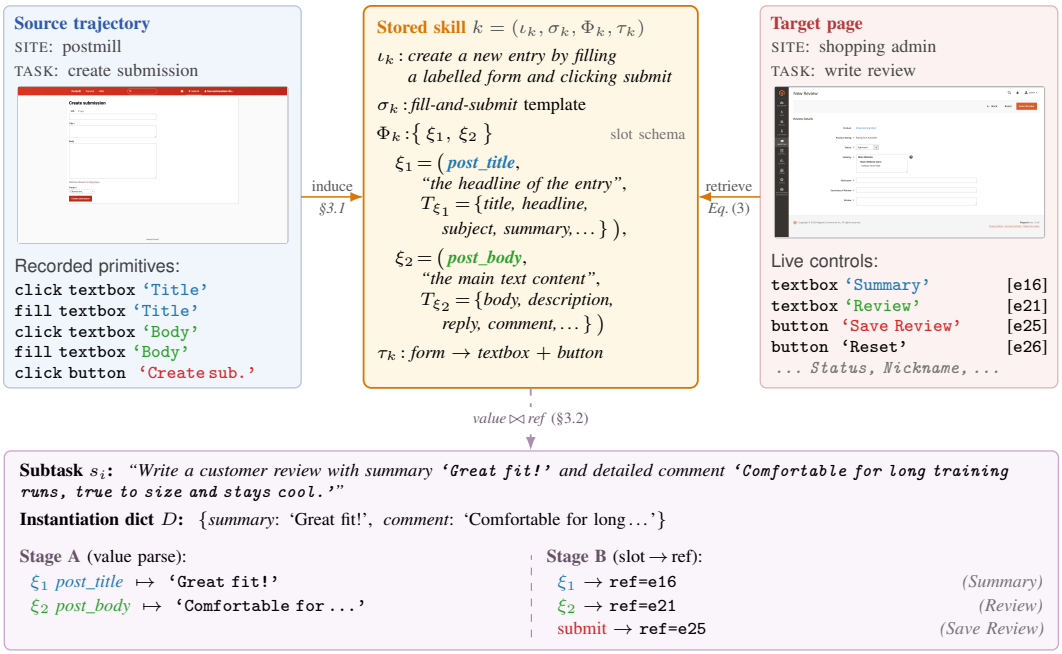


Figure 3: End-to-end cross-domain example. A skill induced from a Postmill *create-submission* trajectory (left) is retrieved on a shopping-admin *write-review* page (right): *post_title* and *post_body* rebind to *Summary* and *Review* through the synonym pools in Φ_k, and the bottom panel walks through Stage A and Stage B for one subtask. Same-colour fields (*title*-like, *body*-like, *submit*) denote the same abstract slot across both pages.

$\Phi_k = \{\xi_1, \ldots, \xi_m\}$ is the *slot schema*, and τ_k is the cleaned accessibility-tree skeleton of the induction-time snapshot, kept as a small labelled tree carrying role and name per node and used by the layout signal in Eq. (3). Each slot ξ stores a key (e.g. *post_title*), a one-line descriptor d_ξ such as *"the headline of the new entry"*, and a small synonym set T_ξ mined from co-clustered training trajectories (e.g. *title* → {*title, headline, subject, name, summary, topic*}). The record contains no `ref=eN` identifiers, which are properties of the live page at test time. Each operation template σ comes with a fixed deterministic *plan* that calls `fill`/`click` primitives once Φ_k is bound (Algorithm 1). Reusing the same template across sites is what makes a skill cross-domain rather than a per-trajectory replay.

Retrieval. Given the current pair $(s_i, \tilde{o}_{i-1})$, we score each skill $k \in \mathcal{K}$ by combining a *text* signal that captures *what kind of subtask* is being attempted with a *layout* signal that captures *what kind of page* the agent is on:

$$\text{score}(k,\, s_i,\, \tilde{o}_{i-1}) \;=\; \alpha \underbrace{\mathbf{e}\big(s_i \,\|\, \rho(\tilde{o}_{i-1})\big)^{\top} \mathbf{e}(\delta_k)}_{\text{text signal}} + (1-\alpha) \underbrace{\mathcal{L}\big(k,\, \tilde{o}_{i-1}\big)}_{\text{layout signal}}, \tag{3}$$

with $\alpha \in [0, 1]$. Here $\mathbf{e}(\cdot)$ is a frozen sentence encoder [17], and $\delta_k = \iota_k \,\|\, \bigcup_{\xi \in \Phi_k} (d_\xi \,\|\, T_\xi)$ is a *rich descriptor* concatenating the skill's intent with every slot descriptor and synonym, so paraphrased subtasks across sites stay close in embedding space [18, 19]. The layout signal $\mathcal{L}(k, \tilde{o}_{i-1})$ uses the live accessibility-tree structure of $\tilde{o}_{i-1}$ to ground retrieval in what the page actually exposes. The *tree edit distance (TED)* term $1 - \text{TED}(\tau_k, \tau(\tilde{o}_{i-1}))/\max(|\tau_k|, |\tau(\tilde{o}_{i-1})|)$ is computed by APTED on small trees [20, 21]. This is the dominant signal when source and target sites use different labels for structurally analogous forms.

We open *skill mode* only if $\text{score}(k^\star, s_i, \tilde{o}_{i-1}) \geq \beta$ for the top-1 skill $k^\star$. Otherwise the agent stays in react mode [4]. This gate matters because not every test subtask has a transferable analog in $\mathcal{K}$, and forcing a weakly matched skill on an unrelated page would silently corrupt the trajectory. α and β are tuned on held-out training trajectories.

Algorithm 1 SKILLMIGRATOR inference for one subtask $(s_i, \tilde{o}_{i-1})$.

Require: subtask s_i, snapshot $\tilde{o}_{i-1}$, library $\mathcal{K}$, encoder $\mathbf{e}$, instantiation dict D, gate threshold β.

compute summary $\rho(\tilde{o}_{i-1})$ and node list $V(\tilde{o}_{i-1})$
$k^\star \leftarrow \arg\max_{k \in \mathcal{K}} \text{score}(k, s_i, \tilde{o}_{i-1})$ via Eq. (3)
if $\text{score}(k^\star, s_i, \tilde{o}_{i-1}) < \beta$ **then**
    return $\pi_\theta(s_i, \tilde{o}_{i-1})$ ▷ fall back to react mode
end if
Stage A: Hungarian-solve $\Phi_{k^\star} \times D$ for slot values. For any unbound slot, extract a span from s_i
Stage B: Hungarian-solve $\Phi_{k^\star} \times V(\tilde{o}_{i-1})$ for slot→ref bindings
emit primitives per the plan of $\sigma_{k^\star}$. Escalate any unbound *required* slot to π_θ

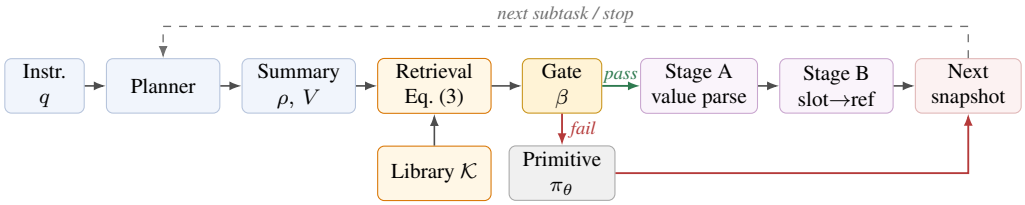


Figure 4: Runtime control flow. Passing the gate β routes the subtask through skill mode (Stage A then Stage B). Failing falls back to a single primitive step from π_θ.

3.2 Slot Binding and Execution

Once $k^\star$ is chosen, the agent must associate each slot $\xi \in \Phi_{k^\star}$ with a concrete *value* string before binding it to a control on the page. Inspired by prior work on cross-domain slot filling [18, 19], we match candidate keys to slots using each slot's NL descriptor d_ξ and synonym pool T_ξ. Exact key matches are not required.

Stage A: parsing slot values from the instruction. The planner forces each subtask s_i to expose an *instantiation dict* $D = \{(\eta_j, v_j)\}_j$ in the task spec [3], where η_j is a key (*subject*, *message*, *from_location*, ...) and v_j is the corresponding value. We score every candidate pair (ξ, η_j) by

$$\mathbf{e}(d_\xi \,\|\, T_\xi)^\top \mathbf{e}(\eta_j) \;+\; \lambda_{\text{lit}} \cdot \mathbb{1}[\eta_j \in T_\xi],$$

and run a Hungarian solve [22, 23] on the resulting $|\Phi_{k^\star}| \times |D|$ matrix to obtain a one-to-one assignment. The literal bonus λ_{lit} is the same disambiguation mechanism long used in slot filling [24]: when η_j already appears verbatim inside T_ξ the pair receives a literal-match bonus. In the example of Figure 3, *summary* appears in T_{post_title} and *comment* in T_{post_body}, so the source-trajectory keys (*post_title*, *post_body*) are correctly bound to the target keys (*summary*, *comment*).

When D is empty or covers fewer slots than $\Phi_{k^\star}$ needs, we extract candidate spans $\mathcal{C}(s_i)$ directly from the instruction text using a fixed cue grammar (quoted strings, list literals, dates, URLs, *from X to Y*, *titled X*, *named X*, capitalised proper-noun phrases, ...). Each span c comes with a small prior reflecting how distinctive its cue is (e.g. a quoted string is a stronger signal than a bare capitalisation). We then score $\mathbf{e}(d_\xi \,\|\, T_\xi)^\top \mathbf{e}(\text{ctx}(c))$ on the surrounding context window with the candidate replaced by a placeholder, and run a Hungarian assignment. The two paths are run in order: dict-match first, then instruction-extract for any still-unbound slot.

Stage B: binding each slot to a control on the page. The bound values must finally be typed into actual page controls. For every interactive node $v \in V(\tilde{o}_{i-1})$ we build a deterministic *control descriptor* $d(v) = \text{role}(v) \,\|\, \text{name}(v)$, score every (slot, control) pair by $\mathbf{e}(d_\xi \,\|\, T_\xi)^\top \mathbf{e}(d(v))$, restrict to controls of compatible role (textbox-like for fillable slots), and run a second Hungarian solve. An assignment is accepted only above a similarity threshold. Below it we declare the slot *unbound* and fall back to π_θ for that slot.

The critical effect of this stage is that the *number of fills* emitted on the live page is determined by the live page and the matched slots, not by the action count of the source trajectory. On a five-field form like *Create new forum* (Name, Title, Description, Sidebar, Tags), an instantiation dict that supplies only three values fills only the three matching textboxes and leaves the other two empty. On

Table 2: WebArena per-domain task success rate (SR, %) and average LLM-action count $\bar{\mathcal{N}}$ on successful trajectories. $\bar{\mathcal{N}}$ rows are extracted from trajectory logs under one unified counter.

	SHOP		ADMIN		REDDIT		GITLAB		MAP		MULTI		AVG.	
METHOD	SR	$\bar{\mathcal{N}}$	SR	$\bar{\mathcal{N}}$	SR	$\bar{\mathcal{N}}$	SR	$\bar{\mathcal{N}}$	SR	$\bar{\mathcal{N}}$	SR	$\bar{\mathcal{N}}$	SR	$\bar{\mathcal{N}}$
ReAct [4]	37.4	6.1	44.0	7.0	66.0	5.0	38.9	7.5	16.4	4.5	10.3	10.5	38.5	6.4
SkillWeaver [8]	39.3	5.7	48.2	6.5	71.2	4.7	50.3	7.0	17.2	4.2	16.3	9.8	43.6	6.0
ASI [7]	46.3	5.8	53.6	6.6	73.7	4.8	46.8	7.1	21.5	4.3	15.1	10.0	46.5	6.1
PolySkill [12]	51.4	5.5	54.8	6.3	73.2	4.5	54.2	6.8	18.9	4.0	18.9	9.5	49.3	5.8
SKILLMIGRATOR	45.5	**4.8**	52.7	**5.6**	72.6	**4.5**	46.7	**6.7**	19.3	**3.0**	16.7	**9.4**	45.7	**5.4**

a two-field form like *New issue* (Title, Description) the same skill emits two fills. This handles the variable-arity issue without touching the stored skill record.

Plan execution. Each operation template $\sigma \in \Sigma$ has a fixed plan keyed on the bound slots. A *search* template emits a `fill` on the search input and a `click` on the submit button, or submits with Enter when no submit button binds. A *fill-and-submit* template emits one `fill` per bound slot in declared order, then a `click` on the button whose label matches the global submit-keyword pool (*create, submit, post, send, save, update, publish*) mined from training. Any unbound *required* slot escalates to π_θ.

4 Experiments

We evaluate SKILLMIGRATOR with a focus on the LLM-action count $\mathcal{N}(s_i, \tilde{o}_{i-1} \mid \mathcal{K}, \pi_\theta)$ from Eq. (1) of §2, corresponding to the *average successful tool steps* metric used in prior work [7, 12]. Three research questions organise the evaluation:

- **RQ1.** Does SKILLMIGRATOR reduce $\mathcal{N}$ while preserving task success rate on different splits where previous baselines are restricted to within-domain reuse?
- **RQ2.** Does SKILLMIGRATOR *compose* with existing skill libraries: when its gate falls back, does plugging an ASI or PolySkill library underneath add to the gain?
- **RQ3.** Which component is responsible for the gain, and how sensitive are the results to the weight α and gate threshold β?

4.1 Setup

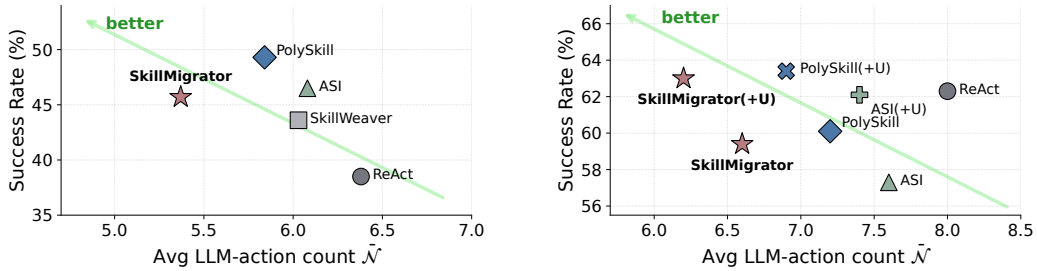
Benchmarks. *Mind2Web* [2] spans 137 websites across 31 domains and is partitioned into three generalisation splits: cross-task (same website, new task), cross-website (same domain, new website), and cross-domain (new domain). Task success is adjudicated using an LLM judge, following prior work [12, 25]. *WebArena* [3] provides 812 executable tasks across shopping, shopping-admin, reddit, gitlab, map, and multi-site columns, scored with the official programmatic validators. We follow the same accessibility-snapshot and tool-calling protocol used by PolySkill, which keeps the action schema fixed across compared systems.

Baselines. Baselines are *ReAct* [4] (no skill library), SkillWeaver [8], ASI [7], and PolySkill [12]. On Mind2Web, ASI, PolySkill, and SKILLMIGRATOR are each reported in both a *static* regime (library fixed before evaluation) and a *+Update* regime (skills induced online during evaluation), following PolySkill.

Metrics. For SKILLMIGRATOR, $\bar{\mathcal{N}}$ is the average *LLM-action count* over successful trajectories. Each trajectory contributes $1 + \sum_i \mathcal{N}(s_i, \tilde{o}_{i-1} \mid \mathcal{K}, \pi_\theta)$, including one initial planning call and counting one LLM call per primitive tool action and zero per retrieved skill. We additionally report task success rate (SR), skill reuse rate, and library size $|\mathcal{K}|$.

Table 3: Mind2Web success rate, average LLM-action count $\bar{\mathcal{N}}$, and library size $|\mathcal{K}|$ on the three generalisation splits with GPT-4.1. Methods are reported in two regimes: a *static* library (rows without the *(+Update)* suffix) fixed before evaluation, and a *+Update* regime (rows tagged *(+Update)*) in which new skills are induced online during evaluation.

	CROSS-TASK			CROSS-WEBSITE			CROSS-DOMAIN			
METHOD	SR	$\bar{\mathcal{N}}$	$\lvert\mathcal{K}\rvert$	SR	$\bar{\mathcal{N}}$	$\lvert\mathcal{K}\rvert$	SR	$\bar{\mathcal{N}}$	$\lvert\mathcal{K}\rvert$	REUSE %
ReAct [4]	53.8	7.0	–	56.2	7.5	–	62.3	8.0	–	–
ASI [7]	52.3	6.6	50	54.9	7.1	47	57.3	7.6	33	17.4
PolySkill [12]	55.4	6.3	43	57.6	6.8	44	60.1	7.2	36	22.7
SKILLMIGRATOR	54.8	**5.8**	**35**	57.1	**6.2**	**38**	59.4	**6.6**	**31**	**28.1**
ASI [7] (+Update)	59.4	6.4	66	58.7	6.9	71	62.1	7.4	66	23.6
PolySkill [12] (+Update)	63.2	6.0	47	61.3	6.5	53	63.4	6.9	56	31.0
SKILLMIGRATOR (+Update)	62.7	**5.5**	**41**	60.5	**5.9**	**47**	63.0	**6.2**	**49**	**35.4**



(a) WebArena average.

(b) Mind2Web cross-domain.

Figure 5: Success rate against average LLM-action count $\bar{\mathcal{N}}$ (upper-left is better). SKILLMIGRATOR sits left of every baseline on both benchmarks, using fewer LLM calls per successful trajectory.

Configuration. Unless stated otherwise we use $\alpha = 0.6$ for the text/layout mixing weight in Eq. (3) and $\beta = 0.20$ for the gate threshold of §3.1. Both were selected on a 10% subset of training trajectories without overlapping with any test set.

Implementation. Our system is implemented on top of the Playwright MCP framework [3] and AgentScope [26] for unified agent orchestration and reasoning. In addition, we use `sentence-transformers/all-MiniLM-L6-v2` [17] as the frozen sentence encoder $\mathbf{e}(\cdot)$, with no fine-tuning on SKILLMIGRATOR data. All embedding similarities reported in Eq. (3) are cosine similarities of L2-normalised vectors. Tree-edit distance for the layout signal is computed by APTED [21] on the cleaned accessibility-tree skeletons. We store TIPs as individual JSON records in a global library and cache their descriptor embeddings in memory. For each subtask, we scan the library and select the highest-scoring TIP using the text-and-layout score in Eq. (3), subject to the gate threshold β.

Skill induction parity. Skills are induced from the same training trajectories used by ASI [7] and PolySkill [12], with their respective induction pipelines. Each baseline follows the setup and pipeline reported in its paper, without adding SKILLMIGRATOR's planner.

4.2 Main Results

Tables 2 and 3, together with the scatter in Figure 5, answer RQ1. On WebArena, aggregate SR is 45.7%, 3.6 percentage points lower than PolySkill's 49.3%. $\bar{\mathcal{N}}$ drops from 5.84 to 5.37, a reduction of about 8%. On Mind2Web the same trade-off shows up: SKILLMIGRATOR (+Update) achieves a 63.0% SR on cross-domain at $\bar{\mathcal{N}} = 6.2$, against PolySkill (+Update)'s 63.4% SR and $\bar{\mathcal{N}} = 6.9$.

[3] `https://github.com/microsoft/playwright-mcp`

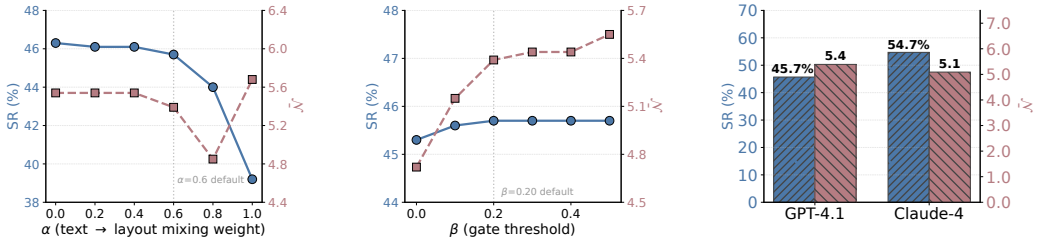
Table 4: Orthogonality study on WebArena (a) and component ablations on WebArena and Mind2Web cross-domain (b).

(a) WebArena average when the gate of SKILLMIGRATOR falls back to a secondary library.

CONFIGURATION	SR %	$\bar{\mathcal{N}}$
ASI alone [7]	46.5	6.1
PolySkill alone [12]	49.3	5.8
SKILLMIGRATOR alone (gate→ReAct)	45.7	5.4
SKILLMIGRATOR + ASI	47.2	5.3
SKILLMIGRATOR + PolySkill	**47.7**	**5.3**

(b) Each row modifies one component of SKILLMIGRATOR. Lower $\bar{\mathcal{N}}$ is better.

	WA AVG.		M2W CD	
VARIANT	SR %	$\bar{\mathcal{N}}$	SR %	$\bar{\mathcal{N}}$
Full SKILLMIGRATOR	**45.7**	**5.4**	**59.4**	**6.6**
text only ($\alpha=1$)	39.2	5.7	55.8	7.3
no synonyms	45.4	5.4	57.0	7.1
zero threshold ($\beta=0$)	45.3	4.7	57.8	6.3



(a) Sweep α. (b) Sweep β. (c) Backbone.

Figure 6: Sensitivity to the mixing weight α, the gate threshold β, and the LLM backbone, on the WebArena. Blue (left axis) is success rate. Rose (right axis) is the average LLM-action count $\bar{\mathcal{N}}$.

These averages are computed over each method's own successful trajectories. The reuse rate further supports this result: cross-domain reuse rises to 35.4% versus PolySkill's 31%, while the library size is consistently smaller. Overall, SKILLMIGRATOR reduces LLM calls per successful trajectory and improves skill reuse on previously unseen domains.

4.3 Orthogonality with Existing Skill Libraries

RQ2 asks whether SKILLMIGRATOR *competes* with baselines or *composes* with them. When the gate falls back, the agent does not need to revert to raw ReAct: control can pass to a secondary library trained by another method. Table 4a shows the result. The hybrid SKILLMIGRATOR+PolySkill row reaches the lowest $\bar{\mathcal{N}}$ in the table (5.3, vs. 5.4 alone and 5.8 for PolySkill alone), and its 47.7% SR sits between the two single-library numbers (45.7 and 49.3). The two libraries cover different slices of the test set: when a layout match exists, the layout signal triggers a SKILLMIGRATOR skill. When no such match exists, control routes to PolySkill underneath. In practice, SKILLMIGRATOR can behave as a retrieval layer that sits on top of existing skill-induction systems rather than competing with them.

4.4 Ablation Study

For RQ3, Table 4b isolates three ingredients: the layout signal $\mathcal{L}$ (*text only* pushes α to 1), the slot-synonym pool T_ξ (*no synonyms* removes paraphrase coverage), and the gate threshold ($\beta=0$). The biggest move is on the WebArena average SR: removing $\mathcal{L}$ drops 6.5 points (45.7→39.2) and lands almost on top of the no-skill baseline (38.5%), because instruction text alone cannot tell apart same-wording, different-structure pages. On Mind2Web cross-domain the same ablation costs 3.6 points (59.4→55.8), and *no synonyms* costs another 2.4 (59.4→57.0): paraphrase coverage matters most when the target site renames its labels. Setting $\beta=0$ is the only ablation that lowers $\bar{\mathcal{N}}$ (5.4→4.7 on WebArena, 6.6→6.3 on Mind2Web) at almost no SR cost (−0.4 on WebArena), so the default $\beta=0.20$ errs on the conservative side—a deployment that tolerates more skill triggers can push for further LLM-call savings.

4.5 Sensitivity Analysis

We sweep $\alpha \in \{0.0, 0.2, \ldots, 1.0\}$ and $\beta \in \{0.0, 0.1, \ldots, 0.5\}$ on WebArena with GPT-4.1, and re-run the headline configuration on Claude-4. Hyperparameters were chosen on a held-out 10% of training trajectories, never on the test set. Figure 6a shows that SR is essentially flat for $\alpha \in [0.0, 0.6]$ (within 0.6 points of the default 45.7%) and collapses to 39.2% at $\alpha = 1.0$, the same drop seen in the *text only* ablation—layout signal is doing the heavy lifting. Figure 6b shows the β sweep behaving as a trade-off between SR and $\bar{\mathcal{N}}$: $\bar{\mathcal{N}}$ slides from 5.6 down to 4.7 as $\beta \to 0$ while SR moves only between 45.3% and 45.7%, so β is a deployment knob rather than a critical hyperparameter. Figure 6c swaps the LLM backbone: Claude-4 gives 54.7% SR and $\bar{\mathcal{N}} = 5.1$ versus 45.7% SR and $\bar{\mathcal{N}} = 5.4$ on GPT-4.1, a $+9$-point SR shift. The gains come from the retrieval design and ride on top of whatever absolute performance the backbone supplies.

5 Conclusion

We presented SKILLMIGRATOR, a web agent that keeps the standard *text snapshot + tool calling* interface used by recent skill-induction systems [7, 12, 8] but indexes induced skills as *transferable interaction patterns (TIP)*—pairing each skill with a structural sketch of the snapshot at induction time, and retrieving globally by layout similarity before grounding to live element references. On Mind2Web and WebArena, this reduces the average LLM-action count on successful trajectories by about 8–10% over the state-of-the-art methods.

Acknowledgments

We thank the reviewers and members of SymbioticLab for their helpful feedback. This work was supported by the Alibaba Research Intern Program. This work was also supported in part by NSF grants CCF-2450085 and CNS-2106184, and by grants from Ford and Cisco. We declare no additional competing interests beyond the affiliations and funding disclosed above.

References

[1] Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35:20744–20757, 2022.

[2] Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2web: Towards a generalist agent for the web. *Advances in Neural Information Processing Systems*, 36:28091–28114, 2023.

[3] 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. Webarena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, 2024.

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

[5] Ke Yang, Yao Liu, Sapana Chaudhary, Rasool Fakoor, Pratik Chaudhari, George Karypis, and Huzefa Rangwala. Agentoccam: A simple yet strong baseline for llm-based web agents. *arXiv preprint arXiv:2410.13825*, 2024.

[6] Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. *arXiv preprint arXiv:2409.07429*, 2024.

[7] Zora Zhiruo Wang, Apurva Gandhi, Graham Neubig, and Daniel Fried. Inducing programmatic skills for agentic tasks. *arXiv preprint arXiv:2504.06821*, 2025.

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

[9] Viraj Prabhu, Yutong Dai, Matthew Fernandez, Jing Gu, Krithika Ramakrishnan, Yanqi Luo, Silvio Savarese, Caiming Xiong, Junnan Li, Zeyuan Chen, and Ran Xu. WALT: Web agents that learn tools. *arXiv preprint arXiv:2510.01524*, 2025.

[10] Zhaoyang Wang, Qianhui Wu, Xuchao Zhang, Chaoyun Zhang, Wenlin Yao, Fazle Elahi Faisal, Baolin Peng, Si Qin, Suman Nath, Qingwei Lin, et al. Webxskill: Skill learning for autonomous web agents. *arXiv preprint arXiv:2604.13318*, 2026.

[11] Zijian Lu, Yiping Zuo, Yupeng Nie, Xin He, Weibei Fan, Lianyong Qi, and Shi Jin. Contractskill: Repairable contract-based skills for multimodal web agents. *arXiv preprint arXiv:2603.20340*, 2026.

[12] Simon Yu, Gang Li, Weiyan Shi, and Peng Qi. PolySkill: Learning generalizable skills through polymorphic abstraction for continual learning. In *International Conference on Learning Representations*, 2026.

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

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

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

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

[17] Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing*, pages 3982–3992, 2019.

[18] Zihan Liu, Genta Indra Winata, Peng Xu, and Pascale Fung. Coach: A coarse-to-fine approach for cross-domain slot filling. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 19–25, 2020.

[19] Liwen Wang, Xuefeng Li, Jiachi Liu, Keqing He, Yuanmeng Yan, and Weiran Xu. Bridge to target domain by prototypical contrastive learning and label confusion: Re-explore zero-shot learning for slot filling. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 9474–9480, 2021.

[20] Kuo-Chung Tai. The tree-to-tree correction problem. *Journal of the ACM*, 26(3):422–433, 1979.

[21] Mateusz Pawlik and Nikolaus Augsten. Tree edit distance: Robust and memory-efficient. *Information Systems*, 56:157–173, 2016.

[22] Harold W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1–2):83–97, 1955.

[23] James Munkres. Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics*, 5(1):32–38, 1957.

[24] Ankur Bapna, Gokhan Tür, Dilek Hakkani-Tür, and Larry Heck. Towards zero-shot frame semantic parsing for domain scaling. In *Interspeech 2017*, pages 2476–2480, 2017. doi: 10.21437/Interspeech.2017-518.

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

[26] Dawei Gao, Zitao Li, Yuexiang Xie, Weirui Kuang, Liuyi Yao, Bingchen Qian, Zhijian Ma, Yue Cui, Haohao Luo, Shen Li, Lu Yi, Yi Yu, Shiqi He, Zhiling Luo, Wenmeng Zhou, Zhicheng Zhang, Xuguang He, Ziqian Chen, Weikai Liao, Farruh Isakulovich Kushnazarov, Yaliang Li, Bolin Ding, and Jingren Zhou. Agentscope 1.0: A developer-centric framework for building agentic applications. *CoRR*, abs/2508.16279, 2025.

[27] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1–2):99–134, 1998.

[28] Tianlin Shi, Andrej Karpathy, Linxi Fan, Jonathan Hernandez, and Percy Liang. World of bits: An open-domain platform for web-based agents. In *International Conference on Machine Learning*, pages 3135–3144. PMLR, 2017.

[29] Shiqi He, Yue Cui, Xinyu Ma, Yaliang Li, Bolin Ding, and Mosharaf Chowdhury. Branch-and-browse: Efficient and controllable web exploration with tree-structured reasoning and action memory. *arXiv preprint arXiv:2510.19838*, 2025.

[30] Feijie Wu, Zitao Li, Fei Wei, Yaliang Li, Bolin Ding, and Jing Gao. Talk to Right Specialists: Iterative Routing in Multi-agent Systems for Question Answering. *arXiv preprint arXiv:2501.07813*, 2025. doi: 10.48550/arXiv.2501.07813. Version 2, revised April 2026.

[31] Feijie Wu, Weiwu Zhu, Yuxiang Zhang, Soumya Chatterjee, Jiarong Zhu, Fan Mo, Rong Luo, and Jing Gao. PORTool: Importance-Aware Policy Optimization with Rewarded Tree for Multi-Tool-Integrated Reasoning. In *Advances in Neural Information Processing Systems*, 2026.

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

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

Appendix

A Detailed Problem Formulation

This appendix expands the description of the web-agent setting that we summarised in §2.

Web agent environment. We model web navigation as a partially observable sequential decision process, following the standard view of interactive agents in uncertain environments [27]. A task is specified by a natural-language instruction q. Since a real web application may contain hidden browser state, server-side state, and asynchronous page updates, the agent only observes an agent-visible representation of the environment. We use an observation-level abstraction and denote the observation space by $\mathcal{S}$. At time step t, the agent receives an observation $o_t \in \mathcal{S}$, which may include the rendered viewport, the DOM, the accessibility tree, and other browser metadata. This representation is consistent with prior web-agent benchmarks, where agents interact with web pages through visual and structural observations [28, 2, 3, 15].

Given the current observation, the agent selects an executable action using an LLM π_θ via $a_t \sim \pi_\theta(\cdot \mid q, o_{0:t}, a_{0:t-1})$, where $o_{0:t}$ and $a_{0:t-1}$ are observation and action histories, respectively. All actions are chosen from the valid action space $\mathcal{A}$ defined in Table 1, such as `click(`e`)` and `fill(`e`,` *text*`)`. After execution, the web environment returns a new observation, $o_{t+1} = \mathcal{T}(o_t, a_t)$, where $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is the transition function induced by the browser and the web application. For simplicity, we write $\mathcal{T}$ as deterministic, although real web transitions may be stochastic due to network latency, dynamic content, or nondeterministic server responses.

Subtasks and the LLM-action count. Given an instruction q, the planner decomposes it into subtasks $\mathbf{s}(q) = \{s_1, \ldots, s_{T_q}\}$. Let $\tilde{o}_i$ denote the observation after finishing subtask s_i, so $\tilde{o}_{i-1}$ is the observation before executing s_i. For each pair $(s_i, \tilde{o}_{i-1})$, the agent retrieves a relevant skill from $\mathcal{K}$ when available. If the subtask is fully covered by a retrieved skill, then $\mathcal{N}(s_i, \tilde{o}_{i-1} \mid \mathcal{K}, \pi_\theta) = 0$. Otherwise the agent falls back to π_θ and generates a trajectory $\tau = (o^{(0)}, a_1, o^{(1)}, \ldots, a_n, o^{(n)})$, where $o^{(0)} = \tilde{o}_{i-1}$, $o^{(j)} = \mathcal{T}(o^{(j-1)}, a_j)$, and $\mathcal{N}(s_i, \tilde{o}_{i-1} \mid \mathcal{K}, \pi_\theta) = n$. The reported $\bar{\mathcal{N}}$ for SKILLMIGRATOR also includes the initial planning call. Equation (1) of the main paper directly minimises the expectation of $\sum_i \mathcal{N}(s_i, \tilde{o}_{i-1} \mid \mathcal{K}, \pi_\theta)$ plus a library-cost regulariser $\lambda\, C(\mathcal{K})$.

B Additional Experiments

B.1 WebArena Results Across Three Runs

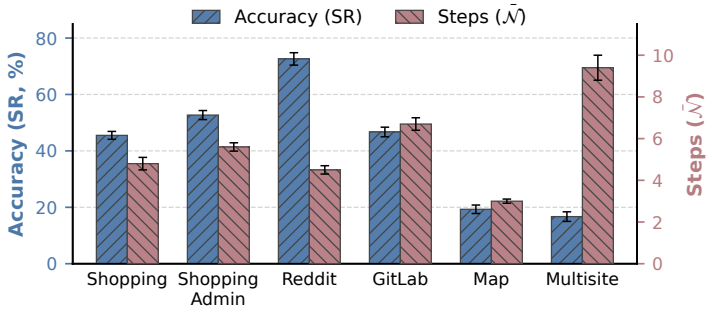
We evaluate SKILLMIGRATOR on the same 812 WebArena tasks with three random seeds, using the settings of Table 2. Figure 7b reports accuracy as task success rate and steps as the average LLM-action count $\bar{\mathcal{N}}$ on successful trajectories, including the initial planning call. Across the six domains, the standard deviation ranges from 1.4 to 2.2 percentage points for accuracy and from 0.1 to 0.6 for steps. Reddit has the highest mean accuracy at 72.6%, while Multisite has the lowest at 16.7% and the largest mean step count at 9.4. The variation across runs is small relative to the overall accuracy range across domains.

B.2 Token Consumption

To complement the LLM-action count, we compare average input and output tokens per task instance on Mind2Web for ASI, PolySkill, and SKILLMIGRATOR under shared model and environment settings. This metric captures both the number of model calls and the length of their inputs and outputs. As shown in Figure 7a, SKILLMIGRATOR uses 43,990.5 tokens per instance, including its initial planning call. This is 24.2% lower than ASI and 16.7% lower than PolySkill. These savings are consistent with executing retrieved TIPs through deterministic plans, reducing the need for repeated policy calls.

Method	Tokens
ASI	58,046.6
PolySkill	52,840.6
SKILLMIGRATOR	**43,990.5**

(a) Mind2Web token consumption.



(b) SKILLMIGRATOR on WebArena across three runs.

Figure 7: (a) Average input and output tokens per Mind2Web task instance. Lower is better. (b) Mean accuracy (blue, left axis) and mean LLM-action count on successful WebArena trajectories (rose, right axis). Error bars indicate ± 1 standard deviation across three random seeds. Both SKILLMIGRATOR cost metrics include the initial planning call.

B.3 Migration Latency

We benchmark the full layout-conditioned skill-migration pipeline on 100 Mind2Web subtasks while varying the library size $|\mathcal{K}|$. Larger libraries are constructed by duplicating existing induced TIPs. Timings include text scoring, APTED, and Stages A and B. Table 5 shows that mean latency per subtask increases from 78.7 ms with one TIP to 420.7 ms with 100 TIPs. These measurements capture local algorithmic overhead. Duplicating TIPs isolates latency scaling, while retrieval quality with more diverse libraries requires separate evaluation. Our static Mind2Web libraries contain 31–38 TIPs, and the rising latency motivates more efficient migration algorithms for larger libraries.

Table 5: Skill-migration latency averaged over 100 Mind2Web subtasks. Larger libraries are constructed by duplicating existing induced TIPs.

Library size $\lvert\mathcal{K}\rvert$	1	25	50	75	100
Mean latency (ms/subtask)	78.7	129.3	188.5	297.2	420.7

B.4 Robustness to Structural Noise

We evaluate retrieval stability on 400 cached Mind2Web subtask snapshots. Each snapshot is perturbed separately by duplicating up to three named controls, relabeling named nodes with probability 0.25, appending five unrelated controls with roles sampled from button, link, and textbox, or prepending four consent nodes (a heading, two buttons, and a link). We then rerun skill migration on each perturbed snapshot with the original retrieval configuration. Figure 8 illustrates the four cases, with added or edited nodes outlined in red. We measure top-1 stability as the percentage of perturbed snapshots that retrieve the same TIP as their original, unperturbed counterparts. Table 6 shows that top-1 stability remains above 90% for duplication and relabeling, but drops to 80.5% for extra controls and 68.8% for context interference. Added nodes can alter both the tree skeleton and page summary, suggesting that filtering irrelevant page content could improve retrieval robustness.

Table 6: Top-1 TIP stability under accessibility-snapshot perturbations on 400 Mind2Web subtasks. Each perturbed input is compared with its unperturbed counterpart.

Perturbation	Top-1 stability (%)
Original (reference)	100.0
Duplicate controls	96.8
Relabel nodes	90.5
Extra controls	80.5
Context interference	68.8

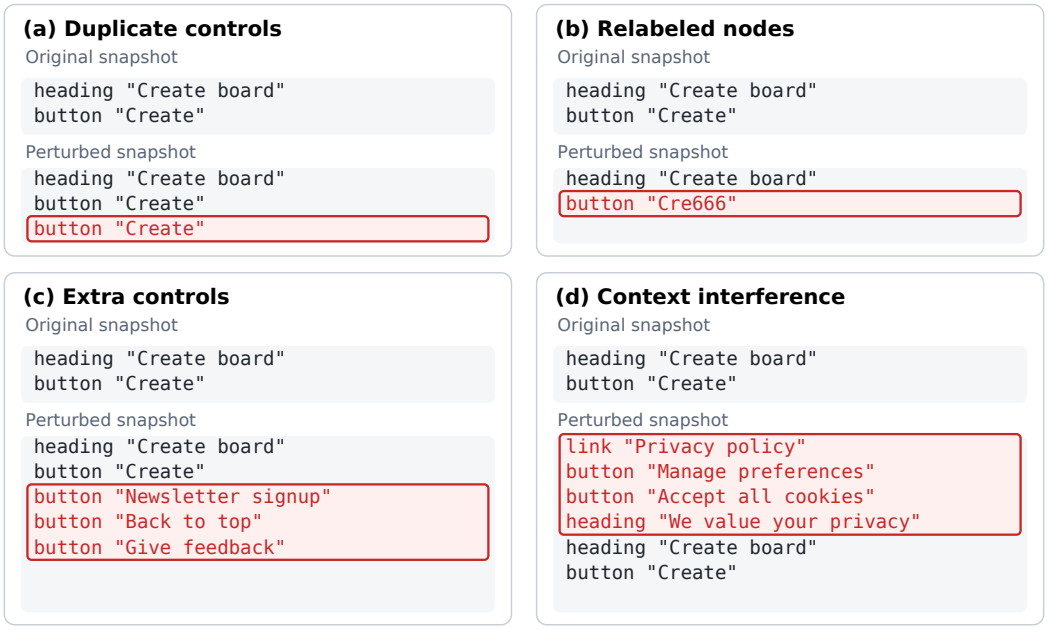


Figure 8: Simplified accessibility snapshots generated using the original perturbation functions, shown before preprocessing. Red boxes mark added or renamed nodes. Only button additions are shown in (c). Consent nodes are prepended in (d). These are synthetic illustrations, not original test snapshots.

C Related Work

C.1 LLM-based Agents and Evaluation Environments

Web agents aim to translate natural-language instructions into executable browser actions over interactive webpages. This problem has been studied through increasingly realistic benchmarks and environments, including WebShop [1], Mind2Web [2], WebArena [3], and Branch-and-Browse [29]. In these settings, agents typically operate in a step-wise decision loop, observing the current webpage state and predicting one low-level action at a time, often following reasoning-and-acting paradigms such as ReAct [4]. This design provides broad flexibility, but it also makes deployment costly on long-horizon tasks, since each primitive action may require a separate policy-facing LLM call. Beyond browser interaction, RIRS [30] routes queries to specialist RAG agents using embedding summaries of their local knowledge and iteratively refines queries using their responses. PORTool [31] assigns step-level credit from rewarded rollout trees to optimize multi-tool reasoning policies. Web-agent evaluation has also expanded to multimodal and realistic settings, such as WebVoyager [14], VisualWebArena [15], WorkArena [13], Odysseys [32], and MolmoWeb [33]. Recent analysis has also highlighted that benchmark improvements do not always imply robust general web competence, motivating more careful measurement of efficiency and generalization in addition to task success [25].

C.2 Reusable Web Skills

To reduce repeated low-level reasoning, a growing line of work studies reusable web skills, which compress frequently occurring primitive action sequences into higher-level abstractions. One line of work stores reusable procedural knowledge in textual form. Agent Workflow Memory (AWM) [6] induces workflow memories from successful episodes and reuses them to guide future problem solving, improving long-horizon behavior through experience reuse. Another line of work emphasizes executable skill representations. ASI [7] induces programmatic skills from successful trajectories and exposes them directly as callable actions. SkillWeaver [8] studies self-improvement through exploration, discovery, and iterative honing of reusable web skills. WALT [9] further argues that agents should exploit higher-level website tools rather than rely entirely on brittle primitive UI interactions.

PolySkill [12] improves transfer by introducing polymorphic abstractions across websites, while more recent work explores richer skill formulations, such as combining executability with interpretability or improving repairability and verification [10, 11]. Across these methods, the common goal is to shorten interaction horizons, reduce policy LLM calls, and improve robustness by reusing validated behavioral structure rather than regenerating every action from scratch.

Our work is most closely related to recent programmatic-skill web agents [7–9, 12], but differs in what it treats as the main bottleneck. Rather than focusing primarily on how to induce or refine the internal implementation of a skill, we focus on how to retrieve and ground previously induced skills under broader transfer conditions. In particular, we target reuse beyond both the same website and the same domain. To do so, we index each skill not only by text but also by a structural sketch of the webpage snapshot at induction time, and retrieve from a global library using layout-aware matching before grounding abstract constraints to live element references. In this sense, our method complements prior work on workflow memory, executable skill induction, and polymorphic abstraction by emphasizing transferable interaction patterns as the retrieval key for cross-website and cross-domain reuse.

D Limitations

Layout-conditioned retrieval assumes that pages sharing accessibility-tree structure also share interaction semantics. This holds for some workflows we evaluate but weakens on visually isomorphic pages whose control flow diverges, where lightweight multimodal cues [14, 15] would be a natural complement to our text-only setting. In addition, the operation templates are mined from observed action shapes, so modalities outside this distribution (e.g. drag-and-drop, modal dialogs) fall back to react mode until the inventory is extended. Clustering does not bound library growth, which increases migration overhead (Appendix B.3). Indexed retrieval and library pruning remain future work. SKILLMIGRATOR reduces the LLM-call cost of web automation, which can improve efficiency and latency for repeated tasks. At the same time, lower-cost automation may increase the scale of automated web actions. Deployments should respect host-site usage policies, rate limits, authentication boundaries, and user consent requirements, and should not use skill reuse to bypass access controls or generate abusive traffic.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: Claims are supported by §4.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes].

   Justification: Limitations are discussed in Appendix D.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

Justification: The paper does not include theoretical results.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Setup is described in §4.1.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: Code and instructions are available at `https://github.com/SymbioticLab/SkillMigrator`. The evaluated benchmarks are publicly available.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Setup is described in §4.1.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: We report standard deviations across three seeds for SkillMigrator on WebArena in Appendix B.1. Due to limited resources, error bars are not reported for the other main comparisons.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).

- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Compute is dominated by API calls to the LLM backbone. The encoder is used frozen at inference (§4.1).

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The work uses only public benchmarks and a frozen public encoder (§4.1).

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: Discussed in Appendix D. The method reduces LLM-call cost of web automation but should be deployed under host-site usage policies.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The contribution is an algorithm evaluated on existing public benchmarks.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: Assets are cited at the point of use under their public licenses.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.

- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: No new datasets or models are released.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: No human subjects are involved.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: No human-subject research is conducted.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: LLMs are core to the methodology as policy backbone and as the judge, both described in §4.1.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.