SEED-Web: Self-Evolution Experience Distillation for Web Agents

Anonymous ACL submission

Abstract

Existing web-agent training methods improve performance through prompting, reinforcement learning, or planner-executor decomposition, but often rely on coarse terminal rewards, auxiliary reward models, or additional supervision, leaving intermediate action quality weakly controlled. In this work, we propose SEED, a teacher-initialized self-evolution framework that refines successful trajectories into reliable step-level supervision through offline filtering. SEED first uses a stronger teacher policy to generate diverse rollouts and retains task-completing trajectories. Unlike prior success-only training, SEED further performs fine-grained trajectory screening to filter out unreliable intermediate interactions and redundant action traces, yielding cleaner action traces for behavior-cloning warm-up. The student is further improved through an offline self-evolution loop: it generates new rollouts, filters them with the same screening procedure, and learns from the retained trajectories via supervised distillation. This design enables continual self-improvement without online reward-gradient optimization, separately trained reward models, or additional human supervision. On WebArena-Lite, SEED achieves **57.6%** and **61.2%** success rates with 7B and 14B backbones, respectively, setting a new SOTA among comparable open-source agents.[1]

1 Introduction

Autonomous web navigation is a key capability for building general-purpose digital agents, as many real-world user tasks require agents to operate directly within interactive web environments (Xie et al., 2023; Yao et al., 2022). To complete such tasks, an agent must interpret user instructions, ground them in dynamically changing webpage states, and issue executable actions such as clicking, typing, selecting, and stopping (Gur et al., 2024; Zhang et al., 2025; Pawlowski et al., 2024; Lo et al., 2023). Compared with static vision-language or text-only tasks, web navigation is substantially more challenging: agents must make multi-step decisions under long interaction histories, dynamic state transitions, partial observability, and sparse task-level feedback (Furuta et al., 2023; Zheng et al., 2024a; Kil et al., 2024).

Recent studies have advanced web agents through structured prompting, supervised adaptation, and reinforcement learning. WebRL and WebAgent-R1 optimize agents from sampled interaction trajectories with task-level feedback, while PLAN-AND-ACT decomposes long-horizon decision-making into planning and execution with synthetic planning supervision (Bai et al., 2024; Erdogan et al., 2025; Yang et al., 2024). Despite their effectiveness, existing RL-based methods still face several limitations. They often require human-collected demonstrations or annotated data for policy warm-up, and WebRL-style approaches further rely on training an additional reward model (Qi et al., 2025), introducing substantial annotation costs and pipeline complexity. Moreover, WebAgent-R1 primarily defines rewards based on whether the final task is successfully completed (Wei et al., 2025), providing limited supervision for intermediate action quality and making it difficult to control the reasoning and interaction process.

These limitations suggest that the key is not to abandon exploration, but to make it more controllable. RL-based methods reveal an important principle: repeated sampling can expose successful behaviors that may be missed by a single rollout (Snell et al., 2025). However, in web navigation, a successful trajectory is not necessarily a reliable one, as it may contain redundant clicks, accidental transitions, hallucinated element references, or unsupported rationales. We therefore reinterpret multi-sample exploration as offline trajectory dis-

[1]The code will be released upon publication.

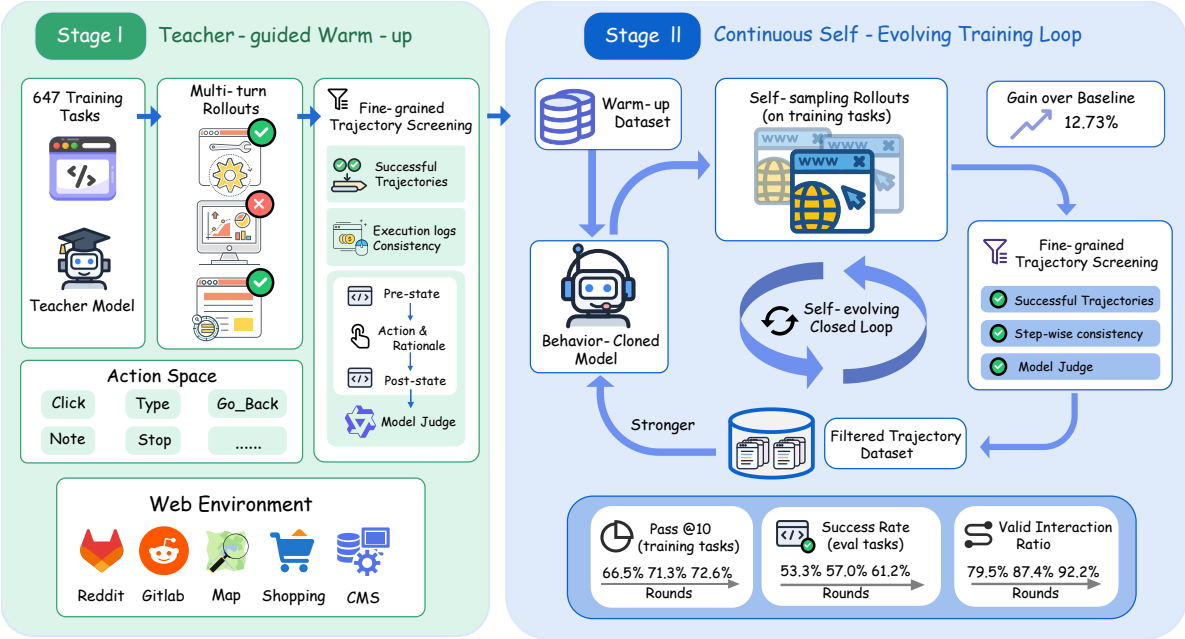


Figure 1: Overview of SEED. Stage I bootstraps the student with teacher-generated multi-rollout trajectories, filtered for task success and local consistency. Stage II iteratively improves the student through offline trajectory expansion, filtering, and supervised distillation. The bottom row tracks representative metrics across rounds.

covery rather than online reward optimization. To turn discovered behaviors into reliable supervision, self-improvement should retain trajectories only when they are both globally successful and locally coherent.

Following this motivation, we propose SEED, a teacher-initialized self-evolution framework for training web agents under sparse trajectory-level feedback. SEED first performs teacher-guided capability bootstrapping, where a stronger teacher policy generates multiple rollouts for each training task. Since terminal success only provides coarse trajectory-level supervision, we first retain task-completing rollouts and then apply Fine-grained Trajectory Screening to reduce noisy supervision. The resulting cleaned trajectories are used to warm up the student policy through behavior cloning. After initialization, SEED enters a continual self-evolution stage, where the student repeatedly generates rollouts, applies the same trajectory screening procedure, and distills the retained trajectories into the next policy. This closed-loop process enables the model to progressively acquire new successful behaviors without online reward-gradient optimization or a separately trained reward model.

On WebArena-Lite, SEED achieves a 57.6% success rate with a 7B backbone and 61.2% with a 14B backbone. At the commonly user open-weight 7B–9B scale (Grattafiori et al., 2024; Hui et al., 2024; Yang et al., 2025a), SEED achieves the best performance among all compared finetuned web agents, outperforming the strongest same-scale baseline by 4.3% points. The 14B variant further achieves the best result among the compared trained web-agent methods. These results demonstrate that SEED effectively improves compact web agents through high-quality self-evolved supervision and scales favorably as backbone capacity increases.

Our contributions are summarized as follows:

- We propose SEED, a teacher-initialized offline self-evolution framework that combines teacher-guided capability bootstrapping, student-generated trajectory expansion, and iterative supervised distillation without relying on online reward-gradient optimization.
- We introduce a *Fine-grained Trajectory Screening* mechanism that refines success-only rollouts by detecting log-based failures and abnormal interaction patterns to remove potentially flawed trajectories, improving the reliability of supervision from rollouts.
- Our experiments on WebArena-Lite show that SEED achieves SOTA performance among the compared finetuned web agents, while continual self-evolution and trajectory screening consistently provide empirical gains.

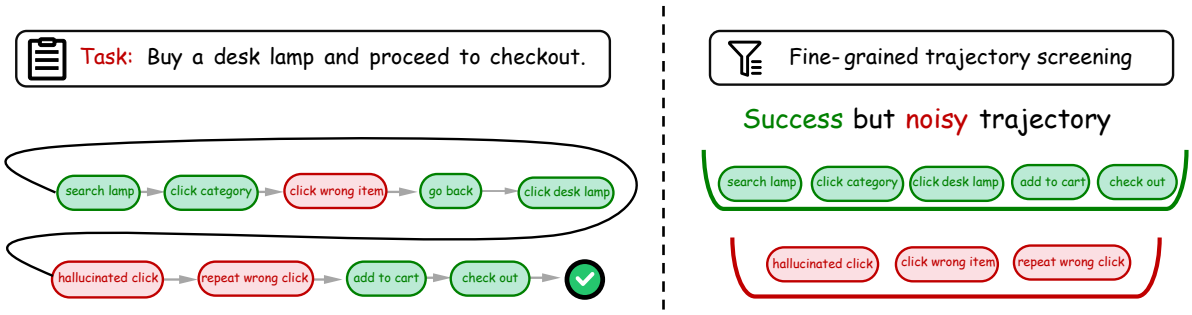


Figure 2: Fine-grained trajectory screening filters noisy intermediate actions from successful web-navigation trajectories, retaining reliable interactions for supervised distillation.

2 Methodology

Problem Formulation. Autonomous web navigation can be formulated as a goal-conditioned POMDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{O}, \mathcal{A}, P, R, p_0 \rangle$ (Sutton and Barto, 2018). Given a task goal g, at step t the agent receives a webpage observation o_t, conditions on the history $h_t = \{(o_i, a_i)\}_{i=1}^{t-1}$, and predicts an action:

$$a_t \sim \pi_\theta(\cdot \mid g, o_t, h_t).$$

An episode yields a trajectory $\tau = \{(o_t, a_t)\}_{t=1}^{T}$. Under the WebArena (Zhou et al., 2024), only a terminal binary reward $R(g, \tau) \in \{0, 1\}$. The learning objective is

$$\max_{\theta} \; \mathbb{E}_{g \sim p(g),\, \tau \sim \pi_\theta(\cdot|g)} \left[R(g, \tau) \right].$$

The filtering signals below are used only for training-data selection, not as environment rewards.

Method Overview. We propose SEED, an offline self-evolution framework for improving web agents under sparse task-level feedback. Instead of relying on human demonstrations, online reward-gradient optimization, or an auxiliary reward model (Erdogan et al., 2025; Qi et al., 2025), SEED converts sampled rollouts into high-quality supervised data through trajectory-level success filtering and fine-grained reliability screening.

As shown in Figure 1, SEED consists of two stages. First, a stronger teacher policy generates multiple rollouts for each training task. Successful and reliable trajectories are retained to warm-start the student via behavior cloning. Second, the initialized student repeatedly samples its own rollouts offline. The same filtering and screening procedure is applied to the student-generated trajectories, which are then distilled back into the policy through supervised learning. This process gradually shifts the source of supervision from teacher-generated seed data to student-generated experience, enabling iterative policy improvement without additional human annotations or online RL updates.

2.1 Teacher-guided Capability Bootstrapping

To obtain a stable initialization without human expert demonstrations, we first use a stronger teacher policy π_{T} to generate candidate trajectories. For each training task $g \in \mathcal{G}$, the teacher performs K independent rollouts:

$$\{\tau_g^{(k)}\}_{k=1}^{K} \sim \pi_{\mathrm{T}}(\cdot \mid g).$$

These rollouts provide initial examples of valid observation-action mappings under the target web-interaction action space. After applying the unified filtering function, the warm-up dataset is

$$\mathcal{D}_{\mathrm{warm}} = \big\{ (g, \tau_g^{(k)}) \mid g \in \mathcal{G},\ 1 \le k \le K, \\ \phi_{\mathrm{filt}}(g, \tau_g^{(k)}) = 1 \big\}.$$

The student policy π_θ is initialized by behavior cloning on the filtered teacher trajectories. For a retained trajectory $\tau = \{(o_t, a_t)\}_{t=1}^{L}$, the warm-up objective is

$$\mathcal{L}_{\mathrm{warm}}(\theta) = -\sum_{(g,\tau) \in \mathcal{D}_{\mathrm{warm}}} \sum_{t=1}^{L} \log \pi_\theta(a_t \mid g, o_t, h_t)$$

This stage transfers reliable teacher behaviors to the student and provides a strong starting point for subsequent self-generated training.

2.2 Fine-grained Trajectory Screening

Terminal task success is a necessary but insufficient indicator of trajectory quality (Ou et al., 2025). In web navigation, a successful rollout may still contain unreliable intermediate behaviors, such as invalid clicks, failed form submissions, repeated attempts, or redundant detours. Directly cloning

Table 1: Task success rate (SR) comparison across different methods at different scales in WebArena-Lite (Qi et al., 2025; Wei et al., 2025). The best score are highlighted in bold.

Method	Backbone	Size	Average SR
No-training baselines	GPT-4-Turbo	-	45.7
	Qwen2.5-Coder	7B	8.5
	Qwen2.5-Coder	14B	23.0
	Qwen3-Coder-480B-A35B	480B-A35B	48.5
Finetuned methods: 7B–9B scale			
DigiRL (Bai et al., 2024)	Llama3.1	8B	30.3
WebRL (Qi et al., 2025)	Llama3.1	8B	42.4
WebRL (Qi et al., 2025)	GLM-4	9B	43.0
WebAgent-R1 (Wei et al., 2025)	Llama3.1	8B	44.8
PLAN-AND-ACT (Erdogan et al., 2025)	Llama3.1	8B	53.3
SEED (Ours)	Qwen2.5-Coder	7B	**57.6**
Additional results at other scales			
WebRL (Qi et al., 2025)	Llama3.1	70B	49.1
PLAN-AND-ACT (Erdogan et al., 2025)	QWQ	32B	54.9
PLAN-AND-ACT (Erdogan et al., 2025)	Llama3.3	70B	57.6
SEED (Ours)	Qwen2.5-Coder	14B	**61.2**

such trajectories can introduce noisy supervision and degrade the student policy. We therefore introduce a fine-grained screening module to retain only successful and reliable trajectories.

Given a candidate trajectory set $\mathcal{D}$, where each example contains a task g, a trajectory τ, and its execution log ℓ_τ, we first keep success-only rollouts:

$$\mathcal{D}_{\text{succ}} = \{(g, \tau, \ell_\tau) \in \mathcal{D} \mid R(g, \tau) = 1\}.$$

We then apply a screening module $\mathcal{F}_{\text{screen}}$ to identify trajectories with explicit failure messages or abnormal interaction patterns. The module combines rule-based log parsing with LLM-based error classification. Trajectories judged unreliable are removed, and near-duplicate trajectories are further pruned to reduce redundant supervision:

$$\mathcal{D}_{\text{clean}} = \text{Prune}_{\text{dup}}\left(\mathcal{F}_{\text{screen}}(\mathcal{D}_{\text{succ}})\right).$$

Equivalently, we define a binary selection function $\phi_{\text{filt}}(g, \tau) \in \{0, 1\}$, where $\phi_{\text{filt}}(g, \tau) = 1$ iff the trajectory is retained in $\mathcal{D}_{\text{clean}}$. The same filtering function is used in both teacher bootstrapping and student self-evolution. Detailed parsing rules, error categories, duplicate-pruning strategy, and classifier prompts are provided in Appendix D.

2.3 Continuous Offline Iterative Self-evolution

After warm-up, SEED no longer depends on the teacher. At iteration i, the current student policy π_{θ_i} performs multiple stochastic rollouts on the training tasks, producing a candidate set $\mathcal{D}_i^{\text{raw}}$. The same success filtering and fine-grained trajectory screening procedure is then applied:

$$\mathcal{D}_i = \{(g, \tau) \mid (g, \tau) \in \mathcal{D}_i^{\text{raw}}, \phi_{\text{filt}}(g, \tau) = 1\}.$$

The retained student-generated trajectories are distilled back into the policy through supervised learning:

$$\mathcal{L}_i(\theta) = -\frac{1}{|\mathcal{D}_i|} \sum_{(g,\tau)\in\mathcal{D}_i} \sum_{t=1}^{L_\tau} \log \pi_\theta(a_t \mid g, o_t, h_t).$$

Minimizing $\mathcal{L}_i$ yields $\pi_{\theta_{i+1}}$, used for rollouts in the next iteration.

This closed loop of self-sampling, trajectory screening, and supervised distillation enables the agent to improve from its own successful experience while avoiding the instability of online RL under sparse terminal rewards.

3 Experiments

3.1 Experimental Setup

Environments. We evaluate on WebArena and WebArena-Lite following recent web-agent training protocols (Zhou et al., 2024; Qi et al., 2025; Wei et al., 2025). WebArena is a realistic, self-hostable benchmark spanning functional websites, including Reddit, GitLab, a CMS, OpenStreetMap, and OneStopShop. Each task requires completing a natural-language goal through multi-step web interactions. We report task success rate (SR), measured by rule-based evaluators that judge whether the final environment state satisfies the goal.

Table 2: Domain-wise Success Rate Across SEED Training Stages on WebArena-Lite.

Policy Stage	Domain SR (%)					Avg. SR (%)	Δ Avg.
	GitLab(#32)	Map(#31)	Reddit(#21)	Shopping(#46)	CMS(#35)		
Teacher policy	59.38	16.13	52.38	54.35	57.14	48.48	–
SEED-R1	59.38	29.03	47.62	65.22	57.14	53.33	+4.85
SEED-R2	56.25	**35.48**	**57.14**	**67.39**	62.86	56.97	+8.49
SEED-R3	**65.62**	**35.48**	**57.14**	**67.39**	**74.29**	**61.21**	**+12.73**

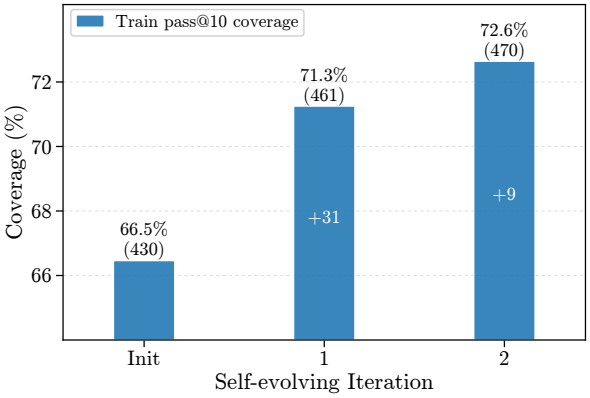


Figure 3: Train pass@10 coverage across continual self-evolution iterations. The increasing coverage indicates that self-evolution progressively expands the set of training tasks from successful trajectories.

Dataset Partitioning. The original WebArena contains 812 tasks (Zhou et al., 2024). Since full-suite evaluation is costly, we adopt the WebArena-Lite split used by WebRL (Qi et al., 2025): the 165 human-verified WebArena-Lite tasks are held out exclusively for final evaluation, while the remaining 647 tasks form the training pool for teacher rollouts and iterative self-evolution. We strictly separate training and evaluation tasks throughout.

Baselines. We compare SEED with both no-training and finetuned baselines on WebArena-Lite. No-training baselines include GPT-4-Turbo (Achiam et al., 2023), Qwen2.5-Coder-7B/14B (Hui et al., 2024), and Qwen3-Coder-480B-A35B (Yang et al., 2025a), which are evaluated without web-specific training. Finetuned baselines include recent web-agent methods such as DigiRL (Bai et al., 2024), WebRL (Qi et al., 2025), WebAgent-R1 (Wei et al., 2025), and PLAN-AND-ACT (Erdogan et al., 2025). We report same-scale comparisons for 7B–9B models and additionally include larger-scale results for reference.

We provide further details on the web environment, rollout protocol, trajectory screening procedure, and training implementation in Appendices A–E.

3.2 Main Results

Table 1 shows that SEED achieves the highest success rates within the reported comparison groups on WebArena-Lite, reaching **57.6 SR** with Qwen2.5-Coder-7B in the 7B–9B fine-tuned group and **61.2 SR** with Qwen2.5-Coder-14B among the trained web-agent methods listed in the table.

At the 7B–9B scale, SEED-7B achieves the highest SR among the fine-tuned methods reported in Table 1. It surpasses WebAgent-R1-8B by **+12.8 points** and the strongest same-scale baseline, PLAN-AND-ACT-8B, by **+4.3 points** (57.6 vs. 53.3). Within the same table, SEED-7B also matches the reported PLAN-AND-ACT-70B result and exceeds WebRL-70B and PLAN-AND-ACT-QWQ-32B, suggesting competitive performance despite its smaller scale.

Increasing the backbone from 7B to 14B improves SR from 57.6 to **61.2**, yielding a **+3.6-point** gain and the highest score among the trained web-agent methods in Table 1. This indicates that SEED remains effective with a larger backbone.

Compared with the corresponding no-training Qwen2.5-Coder baselines, SEED brings substantial gains of **+49.1 points** for 7B and **+38.2 points** for 14B, highlighting the importance of web-specific training and the effectiveness of converting sparse task-level feedback into filtered supervised trajectories.

3.3 Effect of Continual Self-Evolution Across Iterations

Table 2 shows that SEED benefits from repeated self-evolution beyond the teacher-distilled warm-up policy. With the 14B backbone on WebArena-Lite, average SR increases from 48.48% for the teacher policy to 61.21% at SEED-R3, yielding a cumulative gain of 12.73 points. The gains are distributed across domains rather than concentrated in a single website: SEED-R3 obtains the best or tied-best result on all five domains. The largest improvements appear on Map and CMS, suggesting

Table 3: Overall comparison on Qwen2.5-Coder-7B with and without the trajectory screening module.

Setting	Eval SR (%)	# Data	Avg Tokens
W/o filtering	53.3	24321	5871
With filtering	**57.6**	**17554**	**5440**

that self-generated trajectory expansion particularly helps tasks requiring multi-step grounding, state tracking, and structured administrative actions.

The intermediate rounds further indicate that self-evolution provides incremental supervision rather than a one-shot improvement. SEED-R1 and SEED-R2 improve average SR over the warm-up policy, while SEED-R3 brings additional gains, including recovery on GitLab after a small drop in SEED-R2. This pattern suggests that later rounds can refine behaviors not consistently improved in earlier iterations while preserving gains on domains such as Map, Reddit, and Shopping.

Figure 3 provides a complementary training-side view. As the policy improves, pass@10 coverage over the training task pool increases from 66.5% to 72.6%, expanding the number of tasks yielding successful trajectories for distillation. Thus, SEED does not merely imitate trajectories available to the warm-up policy; it gradually enlarges the set of solvable tasks from which reliable supervision can be harvested. Together, Table 2 and Figure 3 suggest that iterative sampling, filtering, and distillation convert sparse trajectory-level success signals into progressively stronger web-agent policies.

3.4 Effect of trajectory screening

Table 3 evaluates the proposed fine-grained trajectory screening module on Qwen2.5-Coder-7B. Although both settings start from task-successful rollouts, applying screening improves Eval SR from 53.3% to 57.6% while retaining fewer trajectories and slightly shorter sequences. This suggests that terminal task success alone is an insufficient criterion for constructing supervision: successful trajectories may still contain explicit execution errors, abnormal interaction patterns, or redundant behaviors that weaken supervised distillation.

The gain is therefore attributable to data quality rather than data scale. As illustrated in Figure 2, the screening module first keeps successful rollouts, then removes unreliable trajectories through log-based error detection and LLM-based error classification, followed by duplicate pruning. This process reduces the training set from 24,321 to 17,554 trajectories, indicating that a substantial portion of nominally successful data is either noisy or redundant. The resulting improvement shows that filtering such trajectories provides more reliable supervision than simply maximizing the amount of self-generated data.

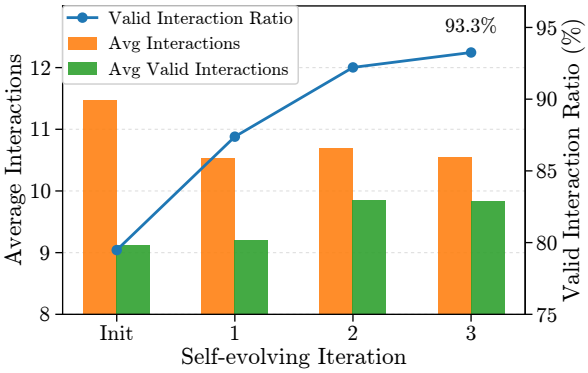


Figure 4: Interaction quality across continual self-evolution iterations under the Fine-grained trajectory filtering. The valid interaction ratio rises to 93.3%.

Figure 4 further supports this interpretation across self-evolution rounds. The valid interaction ratio rises from 79.4% at initialization to 93.3% after three iterations, while the average number of total interactions remains broadly stable. Thus, self-evolution does not mainly improve by producing longer trajectories; instead, the screened data increasingly contains a larger proportion of reliable interactions. Overall, fine-grained trajectory screening serves as a quality-control mechanism for offline self-evolution, preventing unreliable or redundant student-generated behaviors from being repeatedly distilled back into the policy.

3.5 Analysis of Trajectories Across Self-Evolution Rounds

We further analyze the rejected erroneous interactions to understand how self-evolution changes policy behavior. Each error is annotated with a semantic reason using GLM-4.6 and localized into one of ten relative progress buckets within the trajectory. Table 4 shows that SEED substantially reduces erroneous interactions across rounds, from 1158 for the teacher policy to 362 after SEED-R3, a 68.7% decrease. The reductions are not uniform. SEED-R1 removes many coarse execution failures, especially target selection, premature stopping, and navigation errors, suggesting improved grounding and task continuation. SEED-R2 further reduces errors requiring structured interaction, most notably form filling and target selection, indicating better input

Table 4: Semantic error-reason distribution across SEED self-evolution stages. The semantic error reason is annotated by GLM-4.6 using the predefined error taxonomy (Team GLM et al., 2024).

Stage	Target Selection Error	Premature Stop	Loop / Repetition	Form Filling Error	Navigation Error	Recovery Error	Format Error	Stale Context Error	Observation Misread	Over-exploration	Overall
Teacher policy	456	207	190	105	68	62	27	25	16	2	1158
SEED-R1	207	97	139	122	28	28	5	12	28	2	668
SEED-R2	108	58	139	27	17	28	4	9	19	0	409
SEED-R3	118	25	110	27	14	19	12	18	16	3	362

generation and element-level decisions. SEED-R3 yields a smaller overall gain but continues to reduce premature stopping and loop/repetition errors, suggesting that later rounds mainly refine long-horizon control rather than only filtering simple mistakes.

The residual error distribution also reveals the remaining bottlenecks. Target selection remains a major source of failure, while loop/repetition decreases more slowly than other categories and accounts for a larger share of errors after SEED-R3. This shift suggests that, once early grounding and stopping failures are mitigated, the remaining failures increasingly involve harder interaction-control problems such as recovery, repeated actions, and deciding when the task is complete.

Figure 5 provides a complementary temporal view. SEED sharply reduces early-trajectory failures: errors in the first 30% of progress drop from 792 under the teacher policy to 175 after SEED-R3. Meanwhile, the relative share of late-stage errors increases, indicating that improved policies more often pass the initial grounding and navigation phases before failing in advanced task states.

Overall, the analysis suggests that multi-round self-evolution is not redundant. Early rounds mainly improve grounding and execution stability, while later rounds add gains on longer-horizon behaviors. The joint changes in error type and position offer a finer-grained explanation for the downstream improvements obtained by SEED.

3.6 Evolution of Empirical Task Difficulty

We further examine how self-evolution changes the empirical solvability of the training task pool. For each task, we compute the rollout pass ratio $r = n_{\text{succ}}/n_{\text{attempt}}$ and assign it to four policy-dependent difficulty levels: *zero* ($r = 0$), *hard* ($0 < r \leq 0.4$), *mid* ($0.4 < r \leq 0.7$), and *easy* ($r > 0.7$). Because the labels are recomputed each iteration, they reflect empirical solvability under the current policy, not intrinsic task difficulty.

Figures 6b and 6a reveal a clear shift toward higher pass-ratio regimes throughout self-evolution.

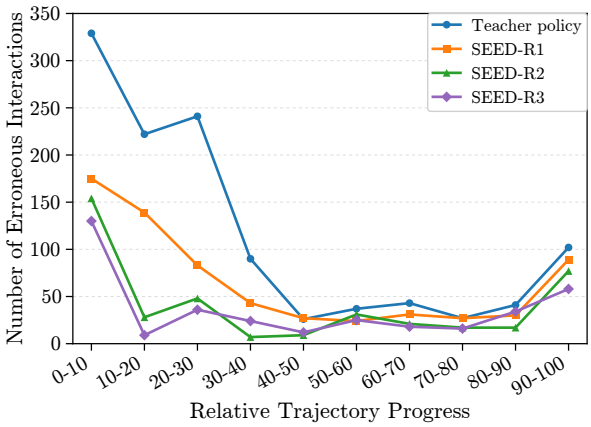


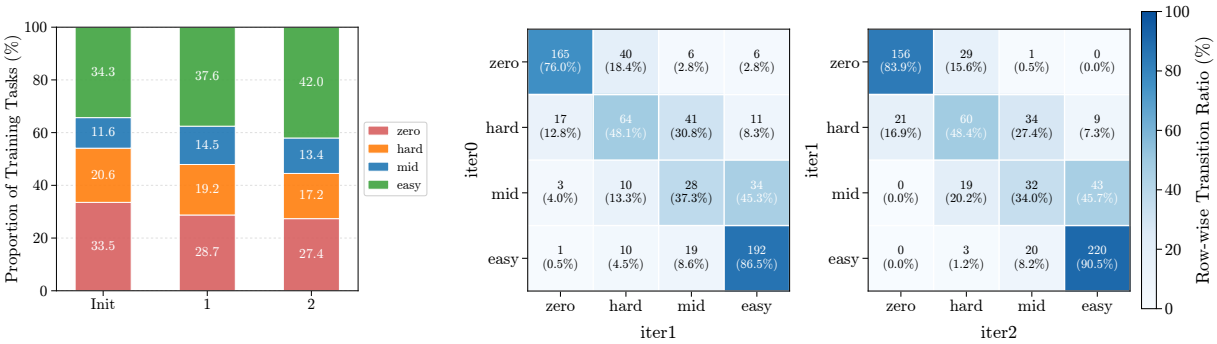
Figure 5: Distribution of erroneous interactions along the relative trajectory progress. Each trajectory is divided into ten progress buckets according to the position of the erroneous interaction.

The first iteration primarily expands coverage, converting many previously unsolved or difficult tasks into solvable ones. Subsequent iterations further consolidate this improvement by making already solvable tasks more reliable, with more tasks moving into the *easy* group and fewer easy tasks falling back to lower regimes.

This trend is also reflected in the aggregate distribution. Across iterations, the proportion of *zero* tasks drops from 33.5% to 27.4%, while the *easy* portion increases from 34.3% to 42.0%. Meanwhile, the combined *mid+easy* share grows steadily, indicating that a larger fraction of the training pool becomes solvable with moderate or high reliability. Together, these results suggest that SEED improves not only isolated rollout success but also the overall quality and stability of the training distribution used for later distillation.

4 Related Work

Web Agents. Large language models have been increasingly used as web agents that interpret natural-language instructions, ground them in dynamic webpage states, and execute multi-step actions in interactive environments (Deng et al., 2023; Zhou et al., 2024; Liu et al., 2025). Existing



(a) Difficulty distribution

(b) Difficulty transitions

Figure 6: Evolution of empirical task difficulty across self-evolution iterations. Left: aggregate task-difficulty distribution, showing a gradual shift toward higher empirical solvability. Right: row-wise transition matrices between consecutive iterations, where each cell reports the task count and row-normalized transition ratio.For example, the entry "40" in the first row and second column of the left matrix indicates that, from `iter0` to `iter1`, 40 previously unsolved tasks (i.e., tasks in the *zero* category) were reclassified as *hard*.

methods can be broadly categorized into training-free and training-based approaches. Training-free agents rely on prompting, tool-use protocols, retrieval-augmented context, or inference-time planning to elicit the capabilities of pretrained LLMs or MLLMs (Koh et al., 2024; Pawlowski et al., 2024). While easy to deploy, their performance is largely bounded by the underlying foundation model and often depends on costly proprietary APIs (Song et al., 2025). Training-based methods instead adapt model parameters using demonstrations, annotated webpages, recorded trajectories, or planning supervision (Deng et al., 2023; Lai et al., 2024; Erdogan et al., 2025). However, such supervision can be costly to obtain and limited in coverage, while static imitation may fail to capture long-term consequences in dynamic web environments (Verma et al., 2025).

Learning from Interaction. To overcome the limitations of static supervision, recent work explores improving agents from interaction experience collected in digital or web environments (Lai et al., 2024; Rawles et al., 2023). DigiRL trains device-control agents with an offline-to-online reinforcement learning paradigm (Bai et al., 2024). WebRL introduces a self-evolving online curriculum framework with an outcome-supervised reward model to address sparse feedback and policy drift (Qi et al., 2025). WebAgent-R1 further applies end-to-end multi-turn reinforcement learning to web navigation using task-level success signals (Wei et al., 2025). These methods show that repeated interaction can uncover successful behaviors beyond static imitation. However, terminal success provides only coarse trajectory-level supervision and successful rollouts may still contain redundant clicks, accidental transitions, hallucinated element references, or unsupported rationales. Existing interaction-learning methods therefore typically rely on some combination of online optimization, reward or evaluator modeling, curriculum design, or additional planning supervision to make exploration learnable. In contrast, SEED treats multi-rollout exploration as offline trajectory discovery and distills only screened successful trajectories into step-level supervision.

5 Conclusion

We presented SEED, a teacher-initialized offline self-evolution framework for training web agents under sparse task-level feedback. The core idea is to reinterpret multi-rollout exploration as offline trajectory discovery, and to convert successful but noisy rollouts into reliable step-level supervision through fine-grained trajectory screening. Starting from filtered teacher-generated trajectories, the student policy is warm-started by behavior cloning and then iteratively improved with its own screened rollouts via supervised distillation, without online reward-gradient optimization or an auxiliary reward model. Experiments on WebArena-Lite show that SEED achieves strong performance at 7B and 14B scales, while ablations and trajectory analyses confirm that its gains come from cleaner supervision, expanded task coverage, and improved interaction reliability. These results highlight reliable self-generated supervision as an effective path for building stronger open-source web agents.

Limitations

Although SEED achieves strong performance, two limitations remain. First, our current trajectory screening module does not fully exploit multimodal webpage information. In WebArena-style environments, the agent can encounter rich visual signals, including screenshots, page layout, icons, and visual state changes. However, SEED primarily performs screening based on execution logs, structured observations, and action-level consistency. This design is simple and effective, but it may miss errors that require visual understanding, such as incorrect element grounding caused by layout ambiguity or visually similar buttons. Future work could incorporate screenshots or vision-language models into the screening process to better verify visual grounding and state transitions.

Second, the experiments are conducted in the closed and sandboxed WebArena environment, following recent web-agent training methods such as WebRL and WebAgent-R1. This setting enables reproducible evaluation with resettable websites and rule-based success evaluators, but it does not cover all constraints of real-world web deployment. In open web environments, agents may handle private user information, submit forms, change settings, modify online content, or trigger irreversible operations. Therefore, practical deployment of SEED-style agents would require additional safety mechanisms, such as user permission control, confirmation for sensitive actions, privacy protection, audit logs, rollback support, and compliance with website access policies.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Hao Bai, Yifei Zhou, Mert Cemri, Jiayi Pan, Alane Suhr, Sergey Levine, and Aviral Kumar. 2024. Digirl: Training in-the-wild device-control agents with autonomous reinforcement learning. *Advances in Neural Information Processing Systems*, 37:12461–12495.

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

Lutfi Eren Erdogan, Nicholas Lee, Sehoon Kim, Suhong Moon, Hiroki Furuta, Gopala Anumanchipalli, Kurt Keutzer, and Amir Gholami. 2025. Plan-and-act: Improving planning of agents for long-horizon tasks. *arXiv preprint arXiv:2503.09572*.

Hiroki Furuta, Kuang-Huei Lee, Ofir Nachum, Yutaka Matsuo, Aleksandra Faust, Shixiang Shane Gu, and Izzeddin Gur. 2023. Multimodal web navigation with instruction-finetuned foundation models. *arXiv preprint arXiv:2305.11854*.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Izzeddin Gur, Hiroki Furuta, Austin Huang, Mustafa Safdari, Yutaka Matsuo, Douglas Eck, and Aleksandra Faust. 2024. A real-world webagent with planning, long context understanding, and program synthesis. In *International Conference on Learning Representations*, volume 2024, pages 52690–52717.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, and 1 others. 2024. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*.

Jihyung Kil, Chan Hee Song, Boyuan Zheng, Xiang Deng, Yu Su, and Wei-Lun Chao. 2024. Dual-view visual contextualization for web navigation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14445–14454.

Jing Yu Koh, Stephen McAleer, Daniel Fried, and Ruslan Salakhutdinov. 2024. Tree search for language model agents. *arXiv preprint arXiv:2407.01476*.

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.

Xiao Liu, Tianjie Zhang, Yu Gu, Iat Long Iong, Song XiXuan, Yifan Xu, Shudan Zhang, Hanyu Lai, Jiadai Sun, Xinyue Yang, and 1 others. 2025. Visualagentbench: Towards large multimodal models as visual foundation agents. In *International Conference on Learning Representations*, volume 2025, pages 95650–95707.

Robert Lo, Abishek Sridhar, Frank F Xu, Hao Zhu, and Shuyan Zhou. 2023. Hierarchical prompting assists large language model on web navigation. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 10217–10244.

Tianyue Ou, Wanyao Guo, Apurva Gandhi, Graham Neubig, and Xiang Yue. 2025. Agentdiagnose: An open toolkit for diagnosing llm agent trajectories. In

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, pages 207–215.

Pawel Pawlowski, Krystian Zawistowski, Wojciech Lapacz, Adam Wiacek, Marcin Skorupa, Sebastien Postansque, and Jakub Hoscilowicz. 2024. Tinyclick: Single-turn agent for empowering gui automation. *arXiv preprint arXiv:2410.11871*.

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.

Christopher Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, and Timothy Lillicrap. 2023. Androidinthewild: A large-scale dataset for android device control. *Advances in Neural Information Processing Systems*, 36:59708–59728.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2025. Scaling llm test-time compute optimally can be more effective than scaling parameters for reasoning. In *International Conference on Learning Representations*, volume 2025, pages 10131–10165.

Yueqi Song, Frank F Xu, Shuyan Zhou, and Graham Neubig. 2025. Beyond browsing: Api-based web agents. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 11066–11085.

Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction*, 2 edition. The MIT Press, Cambridge, MA.

Team GLM, Aohan Zeng, Bin Xu, Bowen Wang, Chenhui Zhang, Da Yin, Dan Zhang, Diego Rojas, Guanyu Feng, Hanlin Zhao, Hanyu Lai, and 1 others. 2024. ChatGLM: A family of large language models from GLM-130B to GLM-4 all tools. *arXiv preprint arXiv:2406.12793*.

Gaurav Verma, Rachneet Kaur, Nishan Srishankar, Zhen Zeng, Tucker Balch, and Manuela Veloso. 2025. Adaptagent: Adapting multimodal web agents with few-shot learning from human demonstrations. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 20635–20651.

Zhepei Wei, Wenlin Yao, Yao Liu, Weizhi Zhang, Qin Lu, Liang Qiu, Changlong Yu, Puyang Xu, Chao Zhang, Bing Yin, and 1 others. 2025. Webagent-r1: Training web agents via end-to-end multi-turn reinforcement learning. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 7920–7939.

Tianbao Xie, Fan Zhou, Zhoujun Cheng, Peng Shi, Luoxuan Weng, Yitao Liu, Toh Jing Hua, Junning Zhao, Qian Liu, Che Liu, and 1 others. 2023. Openagents: An open platform for language agents in the wild. *arXiv preprint arXiv:2310.10634*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025a. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

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

Zonghan Yang, Peng Li, Ming Yan, Ji Zhang, Fei Huang, and Yang Liu. 2024. React meets actre: Autonomous annotation of agent trajectories for contrastive self-training. In *First Conference on Language Modeling*.

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

Yao Zhang, Zijian Ma, Yunpu Ma, Zhen Han, Yu Wu, and Volker Tresp. 2025. Webpilot: A versatile and autonomous multi-agent system for web task execution with strategic exploration. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 23378–23386.

Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. 2024a. GPT-4V(ision) is a generalist web agent, if grounded. In *Forty-first International Conference on Machine Learning*.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, and Zheyan Luo. 2024b. Llamafactory: Unified efficient fine-tuning of 100+ language models. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 3: system demonstrations)*, pages 400–410.

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 Web Environment

WebArena. We conduct experiments in WebArena (Zhou et al., 2024), a realistic and self-hostable benchmark for evaluating language-guided web agents. Unlike static web navigation datasets, WebArena provides fully functional websites whose states can be changed through browser interactions and reset between episodes. The benchmark contains 812 long-horizon tasks, each specified by a high-level natural-language intent. Completing a task often requires multi-step planning, information retrieval, form filling, content editing, or configuration changes across realistic web interfaces.

Observation and interaction interface. Each episode starts from an initial browser state and a natural-language instruction. At every step, the agent receives the current webpage observation, including the page content and interactable elements represented by a structured textual view with element identifiers. The agent then emits one discrete browser action. We implement the agent on top of AgentOccam (Yang et al., 2025b), which standardizes web interaction into a compact action schema rather than allowing arbitrary low-level browser scripts (Zhou et al., 2024).

Action Space. Following AgentOccam (Yang et al., 2025b), agents interact with the web environment through a compact set of predefined browser actions. Each action is generated in a structured textual format, and element-level actions must specify a valid numerical element identifier from the current observation. The resulting action space includes:

Action	Arguments	Purpose
Click	`id`	Click an element by numerical ID.
Type	`id`, `content`, `press_enter_after`	Fill a field and optionally submit with Enter.
Go back	–	Return to the previous page.
Note	`content`	Store intermediate information.
Stop	`answer`	End the episode and optionally return an answer.

We use this AgentOccam-style action interface for all rollout collection and evaluation.

WebArena-Lite split. We follow the WebArena-Lite split used by mainstream RL-based web-agent methods (Qi et al., 2025; Wei et al., 2025). From the 812 WebArena tasks, the 165 human-verified WebArena-Lite tasks are held out for evaluation across five domains (Table 5); the remaining 647 tasks form the training pool for teacher rollouts and iterative self-evolution.

Table 5: WebArena-Lite evaluation tasks by domain.

Domain	GitLab	Map	Reddit	Shopping	CMS
Tasks	32	31	21	46	35

We enforce this split strictly throughout the pipeline. Evaluation tasks are never used for teacher rollout collection, self-evolution, reward construction, supervised fine-tuning, or hyperparameter selection; all training-time interactions are restricted to the 647 non-evaluation tasks to prevent leakage of task instructions, final answers, or task-specific evaluators.

Evaluation metric. We report task success rate (SR), the standard WebArena metric for functional correctness. A task is counted as successful only when the official evaluator verifies that the final answer or environment state satisfies the goal. This makes WebArena evaluation more demanding than simple string matching: many tasks require the agent to create, edit, or configure persistent website state, such as adding a database-backed record or modifying page content. The official rule-based evaluators mainly include:

- **String Match**: checks whether the agent's final textual answer matches the expected output.
- **URL Match**: checks whether the agent navigates to a target page or satisfies a reference URL constraint.
- **Program Execution**: runs a task-specific script to verify the final webpage, database, API response, or other persistent environment state.

Thus, success is assigned to any trajectory that achieves the intended functional outcome, rather than to a trajectory that merely matches a reference action sequence.

B Detailed Pipeline of SEED

This appendix provides the detailed pseudo-code of SEED together with the concrete instantiation used in our experiments. The training task set $\mathcal{G}$ contains the 647 non-evaluation WebArena tasks. We instantiate the teacher policy π_{T} with Qwen3-Coder-480B-A35B-Instruct (Yang et al., 2025a),

and initialize the student from Qwen2.5-Coder-Instruct backbones (7B or 14B, with the main self-evolution analysis using 14B) (Hui et al., 2024). The same filtering function is used in both teacher-guided bootstrapping and student self-evolution. Importantly, all screening signals are used only for offline data selection, not as reward signals for online policy optimization.

Algorithm 1 describes the teacher-guided bootstrapping stage. Here, $\mathcal{G}$ denotes the training task set, $\mathcal{E}$ is the WebArena browser environment, π_{T} is the Qwen3-Coder-480B-A35B-Instruct teacher policy, and π_{θ_0} is the initial Qwen2.5-Coder student policy. In our rollout collection, $K_{\mathrm{T}} = 10$ stochastic teacher rollouts are sampled for each task, and $T_{\max} = 20$ browser actions. R is the official WebArena terminal success evaluator, $\mathcal{F}_{\text{screen}}$ is the trajectory-screening module, and δ controls duplicate pruning among retained trajectories. We use $\mathcal{L}_{\mathrm{BC}}(\theta; \mathcal{D})$ to denote the behavior-cloning loss computed on trajectory set $\mathcal{D}$.

Algorithm 1 Teacher-guided Capability Bootstrapping

Require: $\mathcal{G}$, $\mathcal{E}$, π_{T}, π_{θ_0}, K_{T}, $T_{\max}$, R, $\mathcal{F}_{\text{screen}}$, δ.
Ensure: Warm-started policy π_{θ_0} and replay buffer $\mathcal{B}$.

```
Initialize D_T^raw ← ∅.
for all task g ∈ G do
    for k = 1 to K_T do
        (τ, ℓ_τ) ←
            ROLLOUT(π_T, E, g, T_max).
        Add (g, τ, ℓ_τ) to D_T^raw.
    end for
end for
D_warm ←
    FILTERTRAJECTORIES(D_T^raw,
    R, F_screen, δ).
θ*_0 ←
    arg min_θ L_BC(θ; D_warm).
Set π_θ0 ← π_θ*0.
Set B ← D_warm.
return π_θ0 and B.
```

Algorithm 2 gives the continuous self-evolution stage. The warm-started student π_{θ_0} and replay buffer $\mathcal{B}$ are initialized by Algorithm 1. In the main 14B setting, the student performs three offline self-evolution rounds ($N = 3$), reported as SEED-R1, SEED-R2, and SEED-R3. In each round i, the current Qwen2.5-Coder student π_{θ_i} samples multiple stochastic rollouts per training task, with $K_{\mathrm{S}} = 10$ attempts per task. The same terminal evaluator R, screening module $\mathcal{F}_{\text{screen}}$, and pruning threshold δ are used to select reliable trajectories before distillation into the next student.

Algorithm 2 Continuous Offline Iterative Self-Evolution

Require: $\mathcal{G}$, $\mathcal{E}$, π_{θ_0}, $\mathcal{B}$, K_{S}, N, $T_{\max}$, R, $\mathcal{F}_{\text{screen}}$, δ.
Ensure: Final student policy π_{θ_N}.

```
for i = 0 to N − 1 do
    Initialize D_i^raw ← ∅.
    for all task g ∈ G do
        for k = 1 to K_S do
            (τ, ℓ_τ) ←
                ROLLOUT(π_θi, E, g, T_max).
            Add (g, τ, ℓ_τ) to D_i^raw.
        end for
    end for
    D_i ←
        FILTERTRAJECTORIES(D_i^raw,
        R, F_screen, δ).
    if |D_i| = 0 then
        Set π_θi+1 ← π_θi and continue.
    else
        B ← UPDATEBUFFER(B, D_i).
        θ_i+1 ←
            arg min_θ L_BC(θ; B).
    end if
end for
return π_θN.
```

C Rollout Collection Details

This section specifies how rollouts are collected for teacher bootstrapping and student self-evolution. Each rollout is a complete WebArena episode: the agent receives a task instruction and a structured browser observation, predicts one browser action at a time, and continues until it emits `stop` or reaches the interaction budget. The resulting trajectory, execution log, and terminal task outcome are stored as the raw data later used for success filtering and fine-grained trajectory screening. Table 6 summarizes the rollout settings that directly affect episode length, observation construction, action generation, and decoding.

We use the same rollout protocol across teacher-generated and student-generated trajectories. The stochastic decoding setting is used to encourage diverse attempts across repeated rollouts, while the

Table 6: Rollout hyperparameters used for teacher and student trajectory collection.

Setting	Value	Description
Episode and Observation		
Maximum episode length	20 browser actions	Upper bound on the number of environment interactions in one rollout.
Observation representation	Full-page accessibility tree, pruned to at most 500 browser rows	Textual browser state provided to the policy at each decision step.
Interaction history window	Most recent 3 interaction steps	Recent context included in the actor prompt during rollout generation.
Action Generation		
Action space	`click`, `type`, `stop`, `note`, `go_back`	Executable high-level browser actions available to the agent.
Decoding temperature	0.8	Stochastic sampling setting used to encourage diverse rollout attempts.
Top-p	1.0	Nucleus sampling threshold; no additional probability-mass truncation is applied.
Maximum output length	512 tokens per action prediction	Upper bound on each model response before parsing the predicted action.
Stop sequence	None	Generation is not terminated by an explicit stop string.
Context Management		
Context handling	31K-token context budget; 1,024-token reserve	Long prompts are truncated before querying the model while preserving output budget.

fixed interaction budget and observation truncation keep trajectory collection comparable across tasks and self-evolution rounds.

D Trajectory Screening

D.1 Log-based Filtering

Trajectory screening removes rollouts whose terminal success signal is unreliable because the execution log contains explicit failure signals. In addition to the official WebArena success reward, we parse the wrapper-level logs produced during rollout collection. A trajectory is discarded when it exceeds the interaction budget, produces an action that cannot be parsed into executable browser commands, triggers a browser execution exception, or fails during official evaluation. Table 7 summarizes these deterministic screening rules.

These rules make the retained data conservative: a rollout is used as positive supervision only when it completes the task and contains no explicit wrapper-level failure signal.

D.2 Error Classification

Beyond deterministic log filtering, we classify failed input/output pairs to analyze why a trajectory becomes unreliable. The classifier receives task metadata, the parsed `ACTION`, the original task prompt with interaction history and current observation, and the model output to be diagnosed. It assigns one primary error type and returns a structured JSON record with confidence, rationale, evidence, and optional secondary labels. Table 8 shows the system prompt used for this classifier.

Table 7: Wrapper-level failure signals used for trajectory screening.

Failure signal	Screening treatment
Step budget exceeded	Mark the rollout as failed once the action count exceeds the maximum episode length.
Invalid or empty action	Remove trajectories containing actions that cannot be parsed into executable browser commands.
Browser execution error	Treat environment exceptions during action execution as unreliable interaction steps.
Evaluator exception	Assign zero reward and exclude the trajectory from positive training data.

E Training Implementation Details

We train the student model with LLaMA-Factory on 8 NVIDIA H200 GPUs (Zheng et al., 2024b). We perform full-parameter supervised fine-tuning on the screened successful trajectories using the Qwen chat template. Training uses bf16 precision, FlashAttention-2, gradient checkpointing, and

Table 8: System prompt for trajectory error classification.

Error Classification Prompt

You are an expert annotator for browser-agent failure analysis. Classify one failed browser-agent input/output pair into exactly one PRIMARY error type. Use exactly one of the following English labels as `wrong_type`, and do not translate labels:

- `Navigation Error`: clicks the wrong menu/link, enters the wrong page, or follows an incorrect path.
- `Target Selection Error`: the workflow direction is right, but the model clicks or types into the wrong element.
- `Form Filling Error`: the model fills an incorrect date, query, filter, quantity, name, or other field value.
- `Premature Stop`: the model stops before the task is complete, or the final answer misses required information.
- `Observation Misread`: the current observation contains the needed information, but the model misunderstands it.
- `Stale Context Error`: the model relies on a previous page/state and ignores the changed current observation.
- `Loop / Repetition`: the model repeats similar ineffective actions without making progress.
- `Over-exploration`: enough information is available, but the model keeps searching or opening irrelevant pages.
- `Recovery Error`: a recovery action such as `go_back` is used incorrectly, or the model fails to recover from a known wrong state.
- `Format Error`: the action format is invalid, forbidden, unparsable, or missing required output fields.

Tie-breaking guidance.

1. Choose `Format Error` only when the output command or required sections are structurally invalid.
2. Choose `Premature Stop` when `ACTION` is `stop` and the answer or task is incomplete.
3. Choose `Recovery Error` when a recovery action is the direct mistake.
4. Choose `Loop / Repetition` when the output repeats a recent ineffective action.
5. Choose `Stale Context Error` when the model refers to old elements or page facts that conflict with the current observation.
6. Choose `Observation Misread` when the needed information is visible in the current observation but interpreted incorrectly.
7. Choose `Target Selection Error` when the intended operation is right but the element target is wrong.
8. Choose `Form Filling Error` when the intended field is right but the typed content or value is wrong.
9. Choose `Navigation Error` when the next page or state direction is wrong.
10. Choose `Over-exploration` when enough information is already available but the model continues unnecessary search.

Output rules. Return raw JSON only. Do not output Markdown fences, prose, comments, or `<think>...</think>`. The JSON must have exactly these keys:

```
{
"wrong_type": "<one label from the list>",
"confidence": <number from 0 to 1>,
"rationale": "<one concise sentence>",
"evidence": "<short quote or paraphrase from the input/output>",
"secondary_wrong_types": ["<optional labels>"]
}
```

DeepSpeed ZeRO-3 to support long-context optimization. Table 9 summarizes the main hyperparameters.

Table 9: **Training hyperparameters.** We list the LLaMA-Factory SFT configuration used for the 14B student model.

Hyperparameter	**Value**
General Configuration	
Training framework	LLaMA-Factory
Hardware	$8 \times$ NVIDIA H200 GPUs
Precision	bfloat16
Chat template	Qwen
Max sequence length	16,384 tokens
Full-parameter SFT	
Fine-tuning type	Full
Optimizer	AdamW
DeepSpeed strategy	ZeRO-3
Flash attention	FA2
Gradient checkpointing	Enabled
Per-device batch size	2
Gradient accumulation	4
Global batch size	64
Learning rate	5×10^{-5}
LR scheduler	Cosine
Warmup ratio	0.03
Epochs	3
Data Pipeline	
Preprocessing workers	32
Dataloader workers	8
Overwrite cache	Enabled