COUNTERFACTUAL REPAIR: TRAINING WEB AGENTS ON VERIFIED FIXES OF THEIR OWN MISTAKES

Anonymous authors
Paper under double-blind review

ABSTRACT

Web agents built on small language models often fail a task because of a single decisive mistake. Training on teacher demonstrations spends the teacher on every step and on states the agent never reaches, and reinforcement learning needs many interactions. We present *CounterFactual Repair* (CFR), which teaches an agent the right action exactly where it failed. At the last state from which a failed task can still be completed, a teacher proposes a short repair, and CFR keeps the repair only if execution shows that it helps this policy: rerun from the same state, the agent succeeds far more often with the repair than without it. The agent then learns the verified action with reasoning it writes itself. On WebArena-Lite, CFR improves a Qwen3.5-4B agent on held-out tasks by 6.7 points across three training seeds while keeping the tasks it already solves. On the same training tasks it achieves the highest gain among on-policy expert correction, step-level teacher correction and self-training, with several times less teacher input than the teacher-based alternatives. CFR also does better from the base model's own error states on unseen tasks, and the same pipeline improves agents on WebShop and MiniWoB++.

1 INTRODUCTION

A web agent that fails a task often fails at one decisive step: it clicks the wrong element, commits to the wrong search or sends the wrong answer, and from then on the goal is out of reach. For a small open model such failures are the common case: on WebArena-Lite (Zhou et al., 2024; Liu et al., 2025) the 4B-parameter model we study solves 22% of the 165 tasks and about one in ten of the 121 harder *held-out* tasks, which we exclude from training. The decisive step often comes late: in the unseen-task test set of §5.3, 46 of the 56 decisive errors of the base model come at step 3 or later.

The two standard ways of improving such an agent pay for far more than that one step. Supervised fine-tuning on teacher demonstrations (Xu et al., 2025; Ou et al., 2024; Gandhi & Neubig, 2026) pays the teacher to drive every task from the start, and it trains the student on states the teacher visits rather than on the states where the student goes wrong. Reinforcement learning (Qi et al., 2025) trains on the student's own states, but needs many rollouts and a reward signal for each. On-policy correction methods sit in between: a teacher takes over the student's rollout at a random step and finishes it (Lauffer et al., 2026; Ross et al., 2011), or judges every student step and corrects the first mistake (Chen et al., 2025). Both still pay the teacher for many steps that were not the problem, and neither compares the corrected run with what the student would have done on its own.

We start from a simpler observation: the most informative supervision is at the step where the agent itself failed, given at the last moment the failure can still be averted. *CounterFactual Repair* (CFR, Figure 1) turns this into a data pipeline that teaches the agent the right action at that step. (i) In the agent's own failed trajectories, a teacher that never sees the reference answer locates s_w, the last state from which the task can still be completed. (ii) At s_w the teacher writes a *minimal repair*, instructed to use at most three actions, that puts the agent on a course it can finish on its own. (iii) The repair is admitted only if it passes a counterfactual execution test: from the same state, rebuilt from fresh containers for every rollout, $K{=}4$ rollouts with the repair must succeed at least 75% of the time, and at least 0.5 more often than $K{=}4$ rollouts in which the agent re-decides on its own. (iv) The reasoning before each repair action is sampled from the policy itself, if needed with the verified action as a hint (a form of the rationalisation in STaR (Zelikman et al., 2022)). (v) Repair

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

rows, mixed with the agent's own successful steps, train the agent by supervised fine-tuning once five mechanical checks confirm that training prompts equal inference prompts.

Why this design. Each stage answers one failure mode of learning from a teacher. A repair that looks right to the teacher need not help this policy, so execution against the policy's own success rate at the same state decides admission, and the self arm removes apparent failures the policy would recover from anyway. The repair stops as soon as the policy can finish, so the teacher supplies actions only at the decisive step. Reasoning comes from the policy because a model trained on teacher-written reasoning learned the corrective action after text it never writes, and rarely produced it at inference (App. A.2). WebArena's sites keep orders, posts and repositories across actions, so we rebuild every rollout from fresh containers: both arms then start from the same state, and the collected data reflect what the policy actually does there.

Results. On WebArena-Lite with Qwen3.5-4B, CFR raises held-out success from 9.9% to 16.6% ($+6.7$ pp), a significant gain on every training seed, and it passes a non-inferiority test on tasks the base model already solves (§5.1). On 100 shared training tasks we reproduce OEC (Lauffer et al., 2026), STeP (Chen et al., 2025) and RFT (Yuan et al., 2023) from their original recipes with the same base model, training recipe and evaluation. CFR has the highest held-out gain of all four methods, ahead of OEC by 1.10 pp, RFT by 2.20 and STeP by 2.75 (Tables 3 and 7). It is the only method whose gain over the base model is clear of zero on both the held-out and the full task set. It needs the least teacher input of all teacher-based methods: $4.7\times$ less than STeP, 3–$14\times$ less than OEC and an estimated $24\times$ less than full demonstrations (§5.2). From the exact points where the *base* model went wrong on unseen tasks, CFR succeeds $+7.6$ pp more often, with the largest gain on late errors (§5.3). Re-run on WebShop (Yao et al., 2022), the pipeline raises reward by $+12.2$; on MiniWoB++ it raises success on its training templates by $+23.7$ pp over eight training seeds in a supplementary analysis (§5.4). CFR keeps what the base model can do: it keeps the tasks the base model solves, keeps its success on untrained MiniWoB++ templates where self-training on the same number of successful rows loses ground, and shows no measurable loss zero-shot. CFR learns where the agent acts: on its own errors, with its own reasoning, in its own environment.

Contributions.

- *Counterfactual admission* against the policy's own success rate at the same state (§3): training on the 172 repair rows it admits, together with the policy's own successful steps, raises held-out success by $+6.70$ pp over three training seeds (§5.1).
- *Teacher actions, policy reasoning*: training targets whose reasoning the policy itself produces, which fit the base model far better than teacher-written reasoning (App. A.1, A.2).
- *Rebuilt states on stateful websites*: every rollout starts from fresh containers, and replay refuses any action whose target element does not match (App. A).
- *Evidence*: three training seeds on WebArena-Lite; a controlled comparison with OEC, STeP and RFT under one teacher-cost accounting (§5.1–5.3); replications on WebShop and MiniWoB++ (§5.4).

2 Related work

Where the supervision comes from. Most web-agent training data are complete successful trajectories: demonstrations replayed from tutorials (Xu et al., 2025), synthesised from indirect knowledge (Ou et al., 2024), or collected by exploration (Gandhi & Neubig, 2026; Murty et al., 2025), and self-training that keeps only the policy's own successes (Yuan et al., 2023). None of these contains data from the states at which the policy fails. STaR (Zelikman et al., 2022) adds *rationalisation*: for a problem the model got wrong, it regenerates a rationale with the correct answer given as a hint. CFR's reasoning step applies the same idea to a single verified action (§3). Online reinforcement learning such as WebRL (Qi et al., 2025) trains on the policy's own states and turns failures into new curriculum tasks, not into corrections. CFR is orthogonal to how demonstrations are obtained.

Teacher corrections on the student's states. DAgger (Ross et al., 2011) queries an expert on the states the learner visits. LEAP (Choudhury & Sodhi, 2025) fine-tunes an agent on feedback from an

Table 1: Where supervision comes from. "Verified" = the supervised correction itself is checked by execution (not only the final outcome of a whole trajectory). Teacher cost is qualitative; measured numbers for the methods we reproduce are in Table 3.

Method	Supervised state	Correction	Verified by execution	Reasoning from	Teacher cost
Demonstrations	teacher's own states	full trajectory	LLM judge / none	teacher	every step
RFT	policy's successful runs	none	final outcome	policy	none
OEC	policy state at a random step	expert suffix to the end	final outcome	expert	suffix
STeP	first step a per-step judge flags	reflection + one action	final outcome	teacher	every step judged
SRC	earliest harmful action (rollback)	one action	final outcome	none (action only)	review per branch
CSO	PRM-flagged failed step	one expert action	one rollout succeeds	expert	PRM + 5 actions
Early Experience	expert states, one-step alternatives	none (expert action)	consequences only	policy	demonstrations
Agent-R	MCTS failures	spliced good branch	final reward	templates	none
CFR (ours)	**last completable state of the policy's own failure**	**1–3 actions, then hand back**	**repair vs. policy's own rollouts, $K{=}4$ each**	**policy**	**one localisation + repair per error state**

AI teacher given privileged state; CFR's teacher never sees the reference answer, and execution decides what is kept. OEC (Lauffer et al., 2026) applies this to multi-turn software-engineering agents: the student's rollout is handed to an expert at a uniformly random turn and the expert finishes the episode; only resolved episodes are kept. STeP (Chen et al., 2025) lets a teacher judge every student step and, at the first judged error, insert a first-person reflection and one corrective action, after which the student continues; kept trajectories must succeed. A Few Teacher Steps (Ye et al., 2026) argues, outside the web domain, that short bounded teacher continuations from random student contexts are more cost-effective than full ones. The two closest works are SRC and CSO. SRC (Hao et al., 2026), concurrent with ours, lets the student run speculatively, has a teacher locate the earliest harmful action, rolls the environment back, and inserts one corrective action; it archives trajectories that pass a verifier and uses them for next-action SFT. CSO (Li et al., 2026) starts from failed trajectories on text deep-search tasks: a PRM flags critical steps, an expert proposes alternatives, one successful policy continuation verifies an alternative, and DPO trains on (alternative, original) pairs. CFR differs from CSO in four ways: (1) a *counterfactual* test against the policy's own success rate from the same state ($K{=}4$ each) admits each repair; (2) the policy writes the reasoning in each training row, and CFR trains with SFT; (3) CFR works on stateful websites and rebuilds their state from fresh containers for every rollout; (4) we compare CFR with adapted reproductions of OEC, STeP and RFT on the same tasks under one teacher-cost accounting (§5.2). The first two points also separate CFR from SRC, which keeps trajectories that pass a verifier and trains on the action alone. In CFR the repair is a short sequence written to hand control back, so the policy itself completes the task in most admitted states (85 of 117). Table 1 summarises the comparison.

Verifying a step by rolling out from it. Math-Shepherd (Wang et al., 2024) labels a reasoning step by decoding several completions from it and checking their answers; OmegaPRM (Luo et al., 2024) cuts the cost of this Monte Carlo estimate with a tree search that finds the first error by binary search, and GroundedPRM (Zhang et al., 2025b) checks MCTS-derived step labels with an external tool. All three train process reward models, as does WebArbiter (Zhang et al., 2026b) for web agents, judging each action with principle-guided reasoning. CFR's admission test is closest to the Monte Carlo estimates, but it compares continuations with and without a proposed repair from the same rebuilt state, measuring what the repair adds over the policy's own continuation, and it gates which rows enter supervised training rather than training a judge. Agent Q (Putta et al., 2024) and WebPilot (Zhang et al., 2025a) search web tasks with MCTS; Agent Q also learns from both branches with off-policy DPO.

Learning from failures and self-reflection. ETO (Song et al., 2024) contrasts failed and expert trajectories with a preference loss; Agent-R (Yuan et al., 2025) splices good MCTS branches onto failed prefixes with templated reflections; Early Experience (Zhang et al., 2026a) executes one-step alternatives at expert states and trains on their consequences and on self-generated reflections; RMSWeb (Liu et al., 2026) mines the last aligned state between successful and failed runs for offline

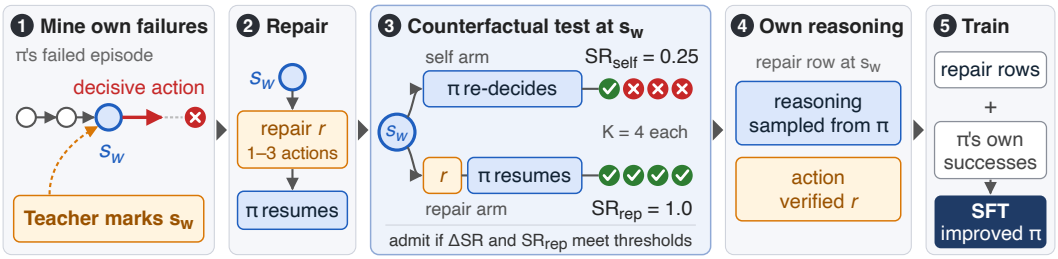

Figure 1: CFR spends teacher supervision (orange) only at error states the agent π (blue) produced itself. (1) A teacher that never sees the reference answer marks the decisive action of a failed episode; s_w is the last completable state, just before it. (2) Its repair r (at most three actions) hands control back to π. (3) On sites rebuilt to s_w, we compare $K{=}4$ rollouts with r against $K{=}4$ in which π re-decides ($\Delta\mathrm{SR} = \mathrm{SR}_{\mathrm{rep}} - \mathrm{SR}_{\mathrm{self}}$, Eq. 1). (4) Admitted repairs yield rows with reasoning sampled from π and the verified action, (5) mixed with π's successful steps for SFT.

RL; RoTS (Bu et al., 2026) synthesises recovery trajectories from other GUI agents' errors, judged by an LLM; Reflexion (Shinn et al., 2023) reflects at inference time. In CFR the error states come from the policy's own rollouts, and each correction is checked by repeated execution against a no-correction control. (STeP is unrelated to the prompting framework SteP (Sodhi et al., 2024).)

3 CounterFactual Repair

Setting. A policy π acts in a web environment. At each step it receives the task statement, the current page as an accessibility tree whose interactive elements carry a numeric identifier, and its own previous actions (the chat template drops earlier reasoning; App. A.1); it writes a reasoning segment and then an action such as `click('497')`. The environment's evaluator scores the episode. CFR produces training rows for π in five stages (Figure 1; full protocol in App. A).

1. Mining error states. We collect failed episodes of π on a training pool (App. A.1). A teacher marks the first action that makes the goal unreachable; the error state s_w is the last state from which the task can still be completed (the state before that action), and every replay stops at s_w, also when that action ended the episode. We place s_w at the last completable state because it is the latest point at which a correction can still change the outcome: before it the task is still open, and after it the task is lost. The agent may already have strayed before s_w; in about one in ten to one in five admitted states the repair first undoes a correctable deviation, such as a misparsed start address, and otherwise it replaces the decisive action (App. A). The teacher may instead judge that no local repair exists; we call such an error *locally unrepairable*. It never sees the reference answer: it localises from the episode's actions and URLs, and writes the repair from the task statement, the action history, the observation at s_w and any earlier observation the localiser points to. When a repair fails verification, the teacher may retry once with the failed attempt and its verification outcome as feedback (App. A).

2. Minimal repair. At s_w the teacher writes a repair $r = (a_1, \ldots, a_m)$, using only the same information and instructed to keep $m \leq 3$. The instruction is to correct the deviation and stop, returning control once the policy can plausibly finish on its own; the repair is not a solution of the task. The repair completes the task only when the deviation is the last decision, for example sending an answer already on the page (App. A counts these cases). We call this the *enabling* principle: the teacher's job is to move the policy to a state from which its own behaviour succeeds. Short repairs keep the teacher cost per admitted state low, and handing control back keeps the rest of the trajectory in the policy's own distribution.

3. Counterfactual admission. Whether a repair helps *this* policy is decided by execution. For every rollout we rebuild the websites from their images (a full reset), replay the recorded prefix up to s_w, and then run two arms. In the *self* arm the policy re-decides at s_w; in the *repair* arm r is executed first and then the policy continues. Each arm runs $K{=}4$ times with independent seeds, giving $\mathrm{SR}_{\mathrm{self}}$ and $\mathrm{SR}_{\mathrm{rep}}$. Replay resolves every recorded action by the target element's role and

accessible name, and refuses to act on a mismatch. A candidate is admitted when

$$\text{every rollout replayed} \quad \wedge \quad \mathrm{SR}_{\text{rep}} \geq 0.75 \quad \wedge \quad \mathrm{SR}_{\text{rep}} - \mathrm{SR}_{\text{self}} \geq 0.5. \tag{1}$$

The second condition asks that the repair work reliably; the third that the success not have been there already, which removes apparent failures the policy would avoid on its own anyway. The two-arm design turns a teacher's judgment of plausibility into a measured effect on this policy at this state. Because admission is decided by execution, the teacher's quality mainly affects how many candidates are admitted; it can also shift which states are admitted (App. A). Every repair candidate is carried to one of three terminal verdicts (tasks judged locally unrepairable or without a legitimate repair have no candidate and are recorded separately; App. A.1): *admitted*, *low efficacy* (the repair executes but fails Eq. 1), or *high baseline* (the self arm already succeeds). Replay refusals and infrastructure faults say nothing about the policy; such candidates are re-queued until they reach a terminal verdict.

4. Reasoning from the policy. An admitted repair fixes *what* to do, not *why*. For each repair action we rebuild the live state, render the exact prompt the policy sees at inference, and sample the policy's own reasoning: first unconditionally (8 samples), keeping one whose action equals the verified action; failing that, with the verified action given as a hint that appears only in the generation prompt (samples that mention the hint are discarded); and, as a last resort, with the action fixed. The training target is this reasoning followed by the verified action. All reasoning comes from the policy, most of it generated with the verified action in view, as in the rationalisation of STaR (Zelikman et al., 2022) (App. A gives the breakdown). We chose this because the model must produce the reasoning itself at inference; a model trained on teacher-written reasoning learned the action conditioned on text it never writes, and rarely produced the action from its own reasoning; with the current pipeline, swapping in teacher-written reasoning costs 4.9 pp on all tasks and 18 pp on the retention set (App. A.2).

5. Training. Each repair step becomes one *off-track* row: the inference prompt at that state and the target above. *On-track* rows are the steps of the policy's own successful episodes on pool tasks, whose targets are the policy's recorded reasoning and action; they keep the model anchored on what it already does well and make up most of the rows. We fine-tune all parameters at a low learning rate, with the loss on the final assistant turn of each row, so that the few repair rows change the policy where it fails without moving it elsewhere. The training job refuses to start unless five mechanical gates pass: every training prompt is byte-identical to the inference prompt for that state; on- and off-track rows share one renderer; every target acts on a visible element and uses no unseen literal; no row exceeds the context length; and no target quotes harness control messages. At a few hundred rows, a handful of mismatched or contaminated rows shows up in the trained model's behaviour (§6), so the gates check every row before training starts.

4 Experimental setup

Policy and environment. The policy is Qwen3.5-4B with reasoning enabled, driven by a single-turn ReAct-style controller whose prompt holds the task, the current accessibility tree and the policy's previous actions; temperature 1.0, 30 steps per episode. We use the self-hosted WebArena sites (Zhou et al., 2024) through BrowserGym (de Chezelles et al., 2025). Every episode, in mining, verification and evaluation alike, runs on containers created from the published images for that episode alone, so no state leaks between rollouts.

Choice of base model. We selected the base model by one principle, which we also fixed in advance on WebShop (one recorded deviation, App. D.5). The model must already solve part of the tasks, because CFR learns from the policy's own successful steps and needs a policy that can finish a task once repaired. It must also fail often enough to leave repairable failures and room to improve. We compared model sizes of one family against this principle. Qwen3.5-4B falls in this range on all three benchmarks. It solves 22.0% of all WebArena-Lite tasks and 9.9% of the held-out ones (Table 2), 13.0% of WebShop tasks at selection time, and 52.3% and 43.6% of MiniWoB++ episodes on the templates we train on and the others (Table 5). The smaller Qwen3.5-2B reaches 2.0% on WebShop, too few successful steps to learn from (App. D.5).

Table 2: WebArena-Lite (165 tasks), Qwen3.5-4B. CFR is trained with three seeds; each arm is evaluated with $K=3$ evaluation seeds under the same harness as the base model. The pooled Δ averages per-task differences over training seeds and bootstraps over tasks ($B=5000$). SR in %, Δ in pp with 95% CI; bold: CI excludes 0. Retention is judged against a non-inferiority margin of -5 pp, frozen after seed 1 and before seeds 2 and 3 were evaluated.

	Three training seeds pooled			Per training seed, Δ vs. base		
Subset	Base	CFR	Δ	seed 1	seed 2	seed 3
Held-out (121)	9.9	16.6	**+6.70** [+3.30, +10.28]	**+9.92** [+5.23, +14.33]	**+5.51** [+1.10, +10.19]	**+4.68** [+1.10, +8.26]
Full (165)	22.0	27.8	**+5.79** [+2.22, +9.36]	**+10.10** [+5.86, +14.34]	+3.84 [−0.61, +8.48]	+3.43 [−0.61, +7.68]
Non-held-out (44)	55.3	58.6	+3.28 [−5.56, +11.87]	**+10.61** [+0.76, +20.45]	−0.76 [−10.61, +9.09]	+0.00 [−12.12, +11.36]
Retention (39)	60.7	66.4	+5.70 [−2.85, +14.25]	+9.40 [−0.85, +18.80]	+5.13 [−5.98, +16.24]	+2.56 [−9.40, +13.68]

Data. The mining pool is the 647 WebArena tasks outside WebArena-Lite. The policy produced a clean failure on 580 of them, and every one of these tasks reached a terminal verdict (App. A.1). The main CFR training set holds 635 rows from 219 tasks: 172 repair rows from 117 admitted error states and 463 on-track rows. Localisation and repairs were written by a frontier LLM (App. A). Training is full-parameter SFT (learning rate 2×10^{-6}, 2 epochs, effective batch 8, fp32 master weights, 160 optimizer steps); the main arm is trained with three seeds.

Evaluation. We evaluate on WebArena-Lite (Liu et al., 2025), the 165-task human-verified subset. The final training data contain none of its tasks, but 44 of them were in training pools of earlier iterations of this project; we report those separately ("non-held-out") and use the other 121, never trained on in any iteration and fixed before any arm in this paper was trained, as the primary *held-out* endpoint. WebArena-Lite served as our development evaluation set; held-out means absent from every training pool. To detect forgetting we use a *retention* set of 39 WebArena-Lite tasks that the base model solved in at least two of three rounds of an earlier base evaluation, independent of the base run we compare against. It was fixed before any arm here was trained and is disjoint from all training data. Each arm is evaluated with $K=3$ evaluation seeds under one scheduling harness, and every comparison pairs it with a base run from the same harness. All absolute rates in this paper come from this harness.

Statistics. We average success per task over evaluation seeds (and, when pooling, the per-task difference over training seeds) and report the mean difference with a 95% task-level paired bootstrap interval ($B=5000$); App. D.2 adds a two-level bootstrap over training seeds. Bold table entries exclude zero. Criteria fixed in advance were frozen in dated documents before the data they judge existed, except the retention margin and the three-seed pooled estimator, frozen after one training seed's evaluation and before the other two.

5 Results

5.1 CFR improves held-out success and keeps existing ability

CFR raises held-out success from 9.9% to 16.6% (+6.7 pp, significant on every training seed; Table 2) and success on all 165 tasks from 22.0% to 27.8%. The held-out gain also holds when training seeds are resampled together with tasks, and App. D.2 breaks it down by seed and template.

Retention. CFR keeps what the base model can do. On the 39 retention tasks it passes the non-inferiority rule fixed before seeds 2 and 3 were evaluated (pooled lower bound above -5 pp and every seed's point estimate above -6 pp), and its point estimate is above the base model's on every seed (Table 2).

5.2 CFR has the highest gain at the lowest teacher cost

Same tasks, adapted original recipes. We drew 100 of the 219 CFR training tasks at random and trained every method on those tasks only, from the same base model, with the same optimiser settings and evaluation; CFR restricted to these tasks (268 rows, 82 of them repair rows) is the

Table 3: CFR has the highest held-out gain on the same 100 tasks, ahead of OEC (original) by 1.10 pp with 13.9× less teacher input. All arms share the tasks, base model, recipe and seed value, and are evaluated identically (Lite165, $K=3$, paired with one base run). Teacher input: all teacher calls that produce the training rows, upstream calls included, as a multiple of CFR's 0.48M input tokens (App. C). Δ in pp, 95% task-level paired bootstrap CI; bold: CI excludes 0. Table 7 (App. B) adds the full task set, row composition, calls and tokens.

		Teacher	Held-out (121)		Retention (39)
Method	Rows	input	vs. base	vs. CFR	vs. CFR
CFR (ours)	268	1×	**+6.06** [+2.20, +9.92]	—	—
OEC, original	338	13.9×	**+4.96** [+0.83, +9.37]	−1.10 [−5.79, +3.58]	**−9.40** [−17.95, −0.85]
OEC, row-matched	268	12.4×	+4.13 [−0.01, +8.26]	−1.93 [−5.79, +1.93]	−7.69 [−18.80, +3.42]
OEC + on-track	524	13.9×	+1.65 [−2.20, +5.52]	−4.41 [−9.37, +0.28]	−8.55 [−19.66, +2.56]
STeP	293	4.7×	+3.31 [−0.28, +6.89]	−2.75 [−6.34, +0.83]	−6.84 [−17.95, +3.42]
RFT	1,087	0	+3.86 [−0.55, +7.99]	−2.20 [−5.79, +1.38]	−3.42 [−16.24, +8.55]
Full demos (est.)	—	24.3×	not trained		

anchor. We reproduce OEC (Lauffer et al., 2026) from its released code, transferred from software engineering to the web: the rollout switches to an expert at a uniformly random step in $[0, 30]$, the expert drives to the end, and only expert turns of resolved episodes are trained; since this yields more rows than CFR, we add a row-matched variant (CFR's 186 on-track rows plus 82 expert rows) and a variant with CFR's on-track rows plus all expert rows. STeP (Chen et al., 2025) follows the paper: a teacher judges every student step, writes a first-person reflection and correction at the first error, the student continues, successful trajectories are kept and the error step is masked. RFT (Yuan et al., 2023) samples the base policy four times per task and keeps de-duplicated successes. App. B lists every deviation.

Effect. CFR has the highest held-out gain of all methods on the same 100 tasks (Table 3, Figure 2), within the range of its three full-data seeds. It is ahead of OEC (original) by 1.10 pp, OEC (row-matched) by 1.93, RFT by 2.20 and STeP by 2.75, and it is ahead of every baseline on the retention set. The strongest baseline, OEC (original), uses 13.9× more teacher input. Pooling two training seeds per arm, CFR leads OEC (row-matched) by 5.79 pp and STeP by 2.89 (Table 11, App. D.3).

Cost. For every method we count all teacher calls needed to produce its training rows if every call succeeds the first time, including upstream calls (CFR's localisation, STeP's per-step judgments), re-tokenised by one tokenizer, with visible tokens only; CFR needs one localisation and one repair call per admitted state. CFR has the lowest teacher input of all teacher-based methods: STeP needs 4.7× more, OEC 3–14× more depending on the variant and where it stops, and full demonstrations an estimated 24× more (Figure 2, App. C). If we also charge CFR for its rejected repair candidates and the baselines for all of their calls, CFR needs 2.3–3.0× less teacher input than OEC and STeP (App. C). CFR buys this saving with self-hosted environment rollouts, which cost compute but no teacher tokens; App. C counts both.

5.3 CFR RECOVERS MORE OFTEN FROM THE BASE MODEL'S ERRORS ON UNSEEN TASKS

To test decisions at the exact points where an agent goes wrong on tasks it has never seen, we built a test set that uses no CFR data: on 60 held-out tasks, the teacher localised s_w in one failing episode of the *base* model per task. Because the states are chosen where the base model failed, we also run both models from the initial state of the same tasks (Δ_{init}) as a reference. Both models then continue from each s_w (rebuilt as in §3), $K=4$ times. The criterion, fixed in advance, was a positive lower bound for the difference from s_w.

CFR succeeds from these error states in 21.4% of rollouts against 13.8% for the base model (+7.6 pp; Table 4), which meets the criterion fixed in advance; it is better on 12 tasks and worse on 4. The result holds under every treatment of capped rollouts and of the one unresolved task (Table 4, App. D.1), and the gain is largest for errors after the second step. From the initial state of

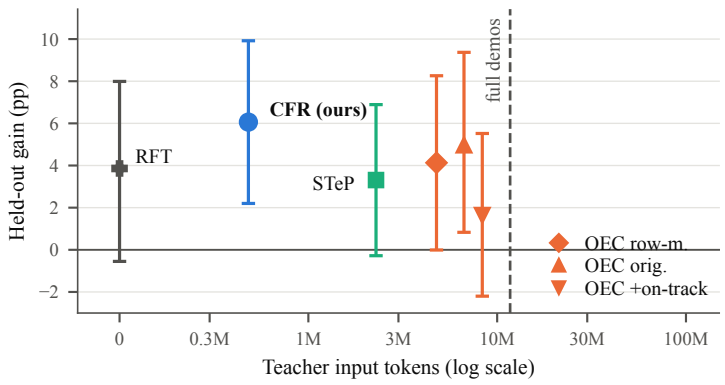

Figure 2: CFR reaches the highest held-out gain with the lowest teacher input of all teacher-based methods. Held-out gain over the base model (pp, 95% CI; values in Table 3) against teacher input tokens (unified idealised count, log scale) on the same 100 training tasks. RFT uses no teacher. The dashed line marks the estimated input of full teacher demonstrations (not trained). OEC variants: original, row-matched (plotted as run; 1.65M if stopped at 82 rows) and with on-track rows added; their points are offset horizontally for readability.

Table 4: Success from the base model's error states on unseen (held-out) tasks. Each error state s_w is localised in a failing *base* episode (test-set construction only); both models continue from s_w, $K{=}4$ each. Rates in %, Δ in pp with 95% CI (task bootstrap); bold: CI excludes 0. W/T/L: tasks where CFR is better/tied/worse. Per-site results are in Figure 3 (App. D.1).

Readout	n	CFR	Base	Δ	W/T/L
Success from s_w (primary)	56	21.4	13.8	**+7.59** [+1.34, +14.73]	12/40/4
wall-clock cap hit counted as failure	56	21.4	12.9	**+8.48** [+2.23, +15.62]	12/41/3
unresolved task, base at its observed 1/4, CFR 0/4	57	—	—	**+7.02** [+0.87, +14.04]	—
CFR 1/4	57	—	—	**+7.46** [+1.32, +14.47]	—
CFR 4/4	57	—	—	**+8.77** [+2.19, +15.79]	—
error at step ≤ 2	10	10.0	7.5	+2.50 [+0.00, +7.50]	1/9/0
error at step ≥ 3	46	23.9	15.2	**+8.70** [+1.09, +16.85]	11/31/4
From the initial state, same tasks (Δ_{init})	60	12.9	7.9	+5.00 [+0.00, +10.00]	15/39/6
$D = \Delta_{s_w} - \Delta_{\text{init}}$ (all four cells)	56	—	—	+2.68 [−4.02, +8.93]	11/36/9

the same tasks CFR's point estimate is also higher. CFR carries its gain to the states where the base model fails on tasks it never trained on.

5.4 The pipeline carries over to WebShop and MiniWoB++

WebShop. We re-ran the entire pipeline on WebShop (Yao et al., 2022) in the AgentGym implementation (Xi et al., 2024), with the same 4B model and the same training recipe; it admitted 91 repairs from 600 training episodes and trained on their 142 repair steps and 385 on-track rows. On the 200 test tasks, reward rises from 19.97 to 32.15 (+12.2, significant; Table 5) and success from 12.8% to 22.0%, meeting the criterion fixed in advance.

MiniWoB++. On MiniWoB++ (Liu et al., 2018) we trained on the eight templates with base success between 0.2 and 0.8 for which the pipeline admitted at least five verified repairs. Success on new environment seeds of these templates rises from 52.3% to 74.2% over three training seeds (Table 5); in a supplementary analysis over eight training seeds the gain is +23.7 pp, ahead of RFT trained on the same number of rows. On the 42 untrained templates CFR keeps the base model's success, while RFT loses ground (§6, App. D.4). Without template selection, on a fixed 25/25 split, CFR gains on its training templates on every training seed (App. D.4).

Table 5: Beyond WebArena. Top: the pipeline re-run on WebShop (task bootstrap) and MiniWoB++ (three training seeds pooled, template-cluster bootstrap). Bottom: the WebArena model zero-shot. Δ with 95% CI (bold: excludes 0).

Benchmark	Split / metric	Base	CFR	Δ
WebShop, 200 test $\times$ 3 seeds	reward ($\times$100)	19.97	32.15	**+12.18** [+8.45, +16.04]
	success rate (%)	12.83	22.00	**+9.17** [+5.33, +13.00]
MiniWoB++, new seeds	8 training templates $\times$ 32	52.3	74.2	**+21.88** [+6.12, +37.11]
	42 other templates $\times$ 16	43.6	44.6	+1.04 [$-$0.65, +3.03]
WorkArena L1, 33 templates, K=1	SR (%), zero-shot	9.1	15.2	+6.06 [$-$9.09, +21.21]
AssistantBench dev, 32 tasks, K=3	accuracy (%), zero-shot	0.9	2.4	+1.47 [$-$0.39, +3.90]

Zero-shot transfer. Evaluated without further training on WorkArena L1 (Drouin et al., 2024) and AssistantBench (Yoran et al., 2024), the WebArena-trained model shows no measurable loss (Table 5).

6 Analysis

CFR teaches the agent to finish. On WebShop, CFR converts episodes that stall on a product page into orders (Figure 5, App. D.5). Over 600 test episodes, CFR orders in 230 against 145 for the base model, with unchanged reward per order, and episodes that end without an order after product-page wandering fall from 181 to 79. Unparsable steps fall from 3.9% to 0.1%. On the 68 tasks where the base model makes no invalid or unparsable step, the reward difference shrinks to $+1.95$, so the gain concentrates on tasks where the base model makes such errors. On WebArena, gains from error states are largest on GitLab and map (Fig. 3).

Repair rows keep the generalisation that extra successful rows lose. MiniWoB++ gives the most direct comparison: its RFT control trains on CFR's 187 on-track rows plus 80 further successful rows of the policy in place of CFR's 80 repair rows, so the two arms differ only in the content of those 80 rows. Over eight training seeds, CFR is ahead of this control by 4.33 pp on the 42 templates neither arm trains on, where CFR keeps the base model's success (App. D.4). On WebArena, RFT trains on 1,087 rows of the policy's own successes on the same 100 tasks, with no teacher and no error states, and CFR is ahead of it on held-out tasks with 268 rows (Table 3). From the base model's error states on unseen tasks, the arm trained without the repair rows stays at the base model's level, while CFR succeeds 4.5 pp more often than that arm (App. D.3).

Data hygiene matters at this scale. Seven rows of an earlier training set quoted harness control messages in their reasoning. The model trained on it emitted pseudo-control text in 23 of 117 retention-set episodes and gave up in 16; without those rows (the data used above) the counts fall to 0 and 2, which motivated the fifth gate.

7 Limitations and conclusion

Limitations and future work. CFR's admission test replays error states from rebuilt environments, so it applies where the state of a website can be reset; live websites with bot protection and changing pages do not yet offer this (App. D.7). Extending the pipeline to screenshot-based agents and to tasks that span several sites is a natural next step, and an adaptive counterfactual test that stops once its outcome is clear would cut the environment rollouts it spends (App. C).

Conclusion. CFR teaches a web agent the right action at the last state from which a failed task can still be completed, and admits a repair only when execution shows that it helps this policy. The agent improves significantly on every training seed, keeps what the base model can do, and leads OEC, STeP and RFT on held-out tasks with 3–14$\times$ less teacher input than OEC.

ETHICS STATEMENT

This work trains web agents in sandboxed, self-hosted replicas of websites (WebArena, WebShop, MiniWoB++) and evaluates zero-shot on WorkArena and AssistantBench; no personal data is collected and no human subjects are involved. The live-web feasibility study (App. D.7) and AssistantBench browse public pages; tasks whose start page was blocked by a CAPTCHA or bot check were excluded rather than circumvented. More capable web agents can also be misused; the methods here make an agent better at tasks it can already partly do.

REPRODUCIBILITY STATEMENT

Section 3 and App. A specify the pipeline, including the admission rule, the replay procedure and the five pre-training gates; App. B lists every deviation of the baseline reproductions from the original recipes; App. C defines the cost accounting. The code needed to reproduce the results and the repair data used for training are available at `https://anonymous.4open.science/r/CFR-73EF`.

AI USE STATEMENT

Large language models are part of the method: frontier LLMs wrote the localisations and repairs in the training data and are the expert or teacher in the baseline reproductions (§3, App. B). We also used AI assistants in three further ways: for research execution, to write experiment and analysis code and to operate experiment runs; for retrieval and discovery, to search for and check related work and citations; and to polish the writing of this paper. The authors made all research decisions and take full responsibility for the content; every reported number is traced to recorded experiment outputs.

REFERENCES

Tianpeng Bu, Xin Liu, Qihua Chen, et al. Recovering policy-induced errors: Benchmarking and trajectory synthesis for robust GUI agents. In *International Conference on Machine Learning (ICML)*, 2026. arXiv:2605.29447.

Yihan Chen, Benfeng Xu, Xiaorui Wang, Yongdong Zhang, and Zhendong Mao. Training LLM-based agents with synthetic self-reflected trajectories and partial masking. *arXiv preprint arXiv:2505.20023*, 2025.

Sanjiban Choudhury and Paloma Sodhi. Better than your teacher: LLM agents that learn from privileged AI feedback. In *The Thirteenth International Conference on Learning Representations*, 2025. URL `https://openreview.net/forum?id=st7XqFgbAH`.

Thibault Le Sellier de Chezelles, Maxime Gasse, Alexandre Lacoste, Massimo Caccia, Alexandre Drouin, Léo Boisvert, Megh Thakkar, Tom Marty, Rim Assouel, Sahar Omidi Shayegan, Lawrence Keunho Jang, Xing Han Lù, Ori Yoran, Dehan Kong, Frank F. Xu, Siva Reddy, Graham Neubig, Quentin Cappart, Russ Salakhutdinov, and Nicolas Chapados. The browsergym ecosystem for web agent research. *Trans. Mach. Learn. Res.*, 2025, 2025. URL `https://openreview.net/forum?id=5298fKGmv3`.

Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H. Laradji, Manuel Del Verme, Tom Marty, Léo Boisvert, Megh Thakkar, Quentin Cappart, David Vazquez, Nicolas Chapados, and Alexandre Lacoste. WorkArena: How capable are web agents at solving common knowledge work tasks? In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, 2024. arXiv:2403.07718.

Apurva Gandhi and Graham Neubig. Go-Browse: Training web agents with structured exploration. In *International Conference on Learning Representations (ICLR)*, 2026. arXiv:2506.03533.

Longkun Hao, Hongyu Lin, Hao Li, Zhichao Yang, Haojie Hao, Dongshuo Huang, Haitao Yang, Hongyu Ge, Ming jie Xie, Yanjun Wu, Zi Hao Yin, Yan Bai, and Yihang Lou. Speculative

rollback correction for quality-diverse web agent imitation, 2026. URL https://arxiv.org/abs/2606.12485.

Niklas Lauffer, Xiang Deng, Srivatsa Kundurthy, Brad Kenstler, and Jeff Da. Imitation learning for multi-turn LM agents via on-policy expert corrections. In *International Conference on Machine Learning (ICML)*, 2026. arXiv:2512.14895.

Mukai Li, Qingcheng Zeng, Tianqing Fang, Zhenwen Liang, Linfeng Song, Qi Liu, Haitao Mi, and Dong Yu. Verified critical step optimization for LLM agents. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 39627–39639, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-395-1. doi: 10.18653/v1/2026.findings-acl.1974. URL https://aclanthology.org/2026.findings-acl.1974/.

Chengbo Liu, Lifang Zhou, Ruijie Yan, Pei Tan, Ao Sun, Haojun Huang, Guichun Hua, Sining Wei, Yining Chen, Yingying He, and Yutao Xie. RMSWeb: Reflection, failure-mode mining, and salvage-DS for web agent reinforcement learning. *arXiv preprint arXiv:2608.00335*, 2026.

Evan Zheran Liu, Kelvin Guu, Panupong Pasupat, Tianlin Shi, and Percy Liang. Reinforcement learning on web interfaces using workflow-guided exploration. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=ryTp3f-0-.

Xiao Liu et al. VisualAgentBench: Towards large multimodal models as visual foundation agents. In *International Conference on Learning Representations (ICLR)*, 2025. arXiv:2408.06327.

Liangchen Luo, Yinxiao Liu, Rosanne Liu, Samrat Phatale, Meiqi Guo, Harsh Lara, Yunxuan Li, Lei Shu, Yun Zhu, Lei Meng, Jiao Sun, and Abhinav Rastogi. Improve mathematical reasoning in language models by automated process supervision, 2024. URL https://arxiv.org/abs/2406.06592.

Shikhar Murty, Hao Zhu, Dzmitry Bahdanau, and Christopher D. Manning. Nnetnav: Unsupervised learning of browser agents through environment interaction in the wild, 2025. URL https://arxiv.org/abs/2410.02907.

Tianyue Ou, Frank F. Xu, Aman Madaan, Jiarui Liu, Robert Lo, Abishek Sridhar, Sudipta Sengupta, Dan Roth, Graham Neubig, and Shuyan Zhou. Synatra: Turning indirect knowledge into direct demonstrations for digital agents at scale. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/a6a6891cf1dfc64d664f086cf5976e93-Abstract-Conference.html.

Pranav Putta, Edmund Mills, Naman Garg, Sumeet Ramesh Motwani, Chelsea Finn, Divyansh Garg, and Rafael Rafailov. Agent Q: advanced reasoning and learning for autonomous AI agents. *CoRR*, abs/2408.07199, 2024. doi: 10.48550/ARXIV.2408.07199. URL https://doi.org/10.48550/arXiv.2408.07199.

Zehan Qi, Xiao Liu, Iat Long Iong, Hanyu Lai, Xueqiao Sun, Jiadai Sun, Xinyue Yang, Yu Yang, Shuntian Yao, Wei Xu, Jie Tang, and Yuxiao Dong. Webrl: Training LLM web agents via self-evolving online curriculum reinforcement learning. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=oVKEAFjEqv.

Stéphane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2011. arXiv:1011.0686.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances*

in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

Paloma Sodhi, S. R. K. Branavan, Yoav Artzi, and Ryan McDonald. Step: Stacked llm policies for web actions, 2024. URL https://arxiv.org/abs/2310.03720.

Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. Trial and error: Exploration-based trajectory optimization for LLM agents. *CoRR*, abs/2403.02502, 2024. doi: 10.48550/ARXIV.2403.02502. URL https://doi.org/10.48550/arXiv.2403.02502.

Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. Math-shepherd: Verify and reinforce LLMs step-by-step without human annotations. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9426–9439, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.510. URL https://aclanthology.org/2024.acl-long.510/.

Zhiheng Xi, Yiwen Ding, Wenxiang Chen, Boyang Hong, Honglin Guo, Junzhe Wang, Dingwen Yang, Chenyang Liao, Xin Guo, Wei He, Songyang Gao, Lu Chen, Rui Zheng, Yicheng Zou, Tao Gui, Qi Zhang, Xipeng Qiu, Xuanjing Huang, Zuxuan Wu, and Yu-Gang Jiang. Agentgym: Evolving large language model-based agents across diverse environments, 2024. URL https://arxiv.org/abs/2406.04151.

Yiheng Xu, Dunjie Lu, Zhennan Shen, Junli Wang, Zekun Wang, Yuchen Mao, Caiming Xiong, and Tao Yu. Agenttrek: Agent trajectory synthesis via guiding replay with web tutorials. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=EEgYUccwsV.

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. *CoRR*, abs/2504.01382, 2025. doi: 10.48550/ARXIV.2504.01382. URL https://doi.org/10.48550/arXiv.2504.01382.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/82ad13ec01f9fe44c01cb91814fd7b8c-Abstract-Conference.html.

Junze Ye, Jiayi Cheng, Miao Lu, Michal Mankowski, Jose Blanchet, and Mohsen Bayati. A few teacher steps go a long way: Cost-efficient on-policy data augmentation for agent post-training. *arXiv preprint arXiv:2607.04574*, 2026.

Ori Yoran, Samuel Joseph Amouyal, Chaitanya Malaviya, Ben Bogin, Ofir Press, and Jonathan Berant. AssistantBench: Can web agents solve realistic and time-consuming tasks? In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2024. arXiv:2407.15711.

Siyu Yuan, Zehui Chen, Zhiheng Xi, Junjie Ye, Zhengyin Du, and Jiecao Chen. Agent-r: Training language model agents to reflect via iterative self-training. *CoRR*, abs/2501.11425, 2025. doi: 10.48550/ARXIV.2501.11425. URL https://doi.org/10.48550/arXiv.2501.11425.

Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Chuanqi Tan, and Chang Zhou. Scaling relationship on learning mathematical reasoning with large language models. *CoRR*, abs/2308.01825, 2023. doi: 10.48550/ARXIV.2308.01825. URL https://doi.org/10.48550/arXiv.2308.01825.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. Star: Bootstrapping reasoning with reasoning. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/639a9a172c044fbb64175b5fad42e9a5-Abstract-Conference.html.

Kai Zhang, Xiangchao Chen, Bo Liu, Tianci Xue, et al. Agent learning via early experience. In *International Conference on Machine Learning (ICML)*, 2026a. arXiv:2510.08558.

Yao Zhang, Zijian Ma, Yunpu Ma, Zhen Han, Yu Wu, and Volker Tresp. WebPilot: A versatile and autonomous multi-agent system for web task execution with strategic exploration. In *AAAI Conference on Artificial Intelligence*, 2025a. arXiv:2408.15978.

Yao Zhang, Yu Wu, Haowei Zhang, Weiguo Li, Haokun Chen, Jingpei Wu, Guohao Li, Zhen Han, and Volker Tresp. GroundedPRM: Tree-guided and fidelity-aware process reward modeling for step-level reasoning. In *NeurIPS Workshop on Language, Agent, and World Models (LAW)*, 2025b. arXiv:2510.14942.

Yao Zhang, Shijie Tang, Zeyu Li, Zhen Han, and Volker Tresp. WebArbiter: A principle-guided reasoning process reward model for web agents. In *International Conference on Learning Representations (ICLR)*, 2026b. arXiv:2601.21872.

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 *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=oKn9c6ytLx.

A PROTOCOL DETAILS

Teachers. Localisation and repairs were written by a frontier LLM under one written protocol. The localisation call sees the actions, URLs and reasoning tails of the whole failed episode, but no page content after s_w. In the first annotation pass the repair call sees only the task statement, the action history up to s_w (plus the policy's action at s_w), the observation at s_w and any earlier observation the localiser points to (1.5 pages on average). The teacher never sees the reference answer or the evaluator configuration; dynamic accessible names are left as rendered. Verification feedback: for states whose first repairs failed verification, a retry round gave the teacher the failed verification rollouts after s_w and their outcomes; 7 of the 117 training states (4 of the 57 in the 100-task subset) come from that round.

Teacher quality mainly affects yield. Every repair is executed $K{=}4$ times against the policy's own continuation from the same state, and only repairs after which the student itself succeeds reliably (Eq. 1) enter the training set. A weaker teacher proposes more repairs that fail this test, so it needs more candidates per admitted repair. It can also change the composition of the admitted set, in two ways. The test is a threshold at $K{=}4$, so a teacher that proposes more marginal repairs also gets more of them admitted (see *Verification* below for the pass probabilities), and a teacher that repairs only some kinds of error yields training data concentrated on those states. The choice of teacher therefore changes the admission rate and the cost per admitted state, and can change which states the method trains on.

Repairs. The teacher is instructed to use at most three actions; two admitted states in the main data carry a four-action sequence, which we keep because admission is decided on the executed sequence. In 32 of the 117 admitted states (36 of 172 repair rows) the repair itself ends the episode, 29 with a final answer and 3 by declaring the task infeasible; in the other 85 every successful repaired rollout continues under the policy. On WebShop, 56 of the 91 admitted repairs end with the purchase. The repair is executed action by action, each action re-grounded against the live page by the recorded element identity, and control returns to the policy after the last one.

Verification. Each rollout starts from containers created from the published images, re-applies the environment patches of the benchmark's setup scripts, replays the recorded prefix, and checks a login probe before handing control to the policy. Replay resolves each action by the target element's role and accessible name and refuses to act when the recorded identifier points to a different element; a candidate with any refused replay is not admitted. Both arms stop early once the verdict is determined (for example, three self-arm successes make the attribution condition unreachable), so $\mathrm{SR}_{\mathrm{rep}}$ of rejected candidates is right-censored. Wall-clock caps protect against stuck environments; a rollout that hits the cap is re-drawn, and we report cap sensitivity wherever caps were hit. Each training state was judged over 1.62 candidate versions on average (median 1, maximum 5). These counts differ in scope. The feedback retry of §3 happens at most once per annotation round, but a later mining round can annotate a state again, and each newly verified repair is a new candidate version; the 1.62 counts the versions on the training state itself, one admitted version of which enters training. The 9.9 candidates per admitted state in App. C instead divide all candidates judged in the pool (counted up to each state's first admission, most of them on tasks that never admitted a repair) by the 147 admitted states. At $K{=}4$ the repair-arm threshold alone ($\mathrm{SR}_{\mathrm{rep}} \geq 0.75$) is met with probability 0.31 by a repair whose true success probability is 0.5 and missed with probability 0.05 by one at 0.9; full admission also depends on the self arm and on every rollout replaying.

Policy-generated reasoning. For each repair step we replay to the live state, render the inference prompt, and sample the policy (temperature 0.9): 8 unconditional samples, keeping one whose action equals the verified action (*self*); otherwise up to 6 samples with the verified action given as a hint that is not part of the training prompt, discarding samples that reveal the hint (*rationalised*); otherwise a sample with the action fixed (*forced*). Every WebArena repair row has weight 1. In the main data the 172 repair rows are 53 self, 112 rationalised and 7 forced; in the 100-task subset, 22, 56 and 4. Forced rows receive an additional check that the reasoning does not refer to anything absent from the prompt; one such row was withheld for inventing a "previous action".

Training. Full-parameter SFT of Qwen3.5-4B with Adafactor, learning rate 2×10^{-6}, 2 epochs, batch 1 with 8 gradient-accumulation steps, fp32 master weights (bf16 parameters silently absorb updates at this learning rate), maximum length 38,912 tokens with truncation treated as an error, no KL term. The loss covers the final assistant turn of each row (reasoning and action). The three training seeds of the main arm differ only in the seed.

Evaluation harness. All WebArena-Lite arms, including the base model, are evaluated with one scheduler that runs each episode on its own containers, with evaluation seeds 40, 41 and 42; the model is served with vLLM in float16 with an 81,920-token context. The same harness evaluates every comparison in §5.1 and §5.2. An episode without a clean verdict (1,800 s cap, serving outage or browser error) is re-run, at most three times; this affects 0 to 6 of the 495 episodes per arm, and scoring all of them as failures changes no Δ by more than 0.2 pp. An earlier per-task harness gives a different absolute base rate (24.9% vs. 22.0% on all 165 tasks), which is why we never compare absolute rates across harnesses.

A.1 Population accounting

The mining pool is the 647 WebArena tasks outside WebArena-Lite. The base policy produced a clean failure (one that survives a check of the harness error field and is not an infrastructure fault) on 580 of them. These failures come from two controller configurations of the same base model: 229 from the evaluation controller and 351 from an earlier one with temperature 0, 512 output tokens and 12 steps (66 of the 117 training states are localised on the latter). For those 66 states the actions before s_w come from the earlier controller; verification and every training row use the evaluation controller's prompt template. The chat template drops the reasoning of all earlier assistant turns, in training and at inference alike, so a row conditions only on the earlier actions: we checked that the rendered training prompts are token-identical whichever controller recorded the history (83 of 83 affected rows), and that the serving endpoint renders the same prompt. The 580 failures split as $580 = 106 + 68 + 4 + 402$: 106 admitted and 68 high-baseline verdicts carried over from a first campaign, 4 tasks judged locally unrepairable (§3) in a review pass, and 402 tasks annotated and verified in a second campaign. All 402 tasks reached a terminal state: 41 admitted, 37 high baseline, 322 low efficacy, and two tasks with no legitimate repair under the protocol. No task was left with a replay, grounding or infrastructure outcome. Errors at the very first action were kept and repaired from the initial state. On-track rows are steps of the base policy's own successful episodes in collection runs on pool tasks, never continuations from verification: 245 rows come from 37 successful episodes of the mining run and 218 rows from 119 successful episodes of six earlier collection runs of the same base policy (102 tasks in total). No task contributes both on-track and repair rows. Verdicts and training admission are separate. Of the 147 admitted states (106 + 41), 30 did not enter training because no valid training row could be assembled: the legal state could not be recaptured on replay (7), the repair target was absent from the recaptured observation (4), the policy produced no matching reasoning within the budget (3), the annotation source could not be resolved (2), the exact inference prompt could not be rebuilt for a historical batch (5), a historical zero-budget row (3), a six-step historical macro (2), an annotation that used information from after s_w (1), a repair that encoded a withheld reference value (1), reasoning that leaked the hint (1), and reasoning that failed semantic review (1). The main training set uses 117 admitted error states (172 repair rows) and 463 on-track rows over 219 tasks. Eight tasks that scored zero because of incorrect reference values in the evaluator were corrected against a recorded twelve-field diff and returned to verification rather than dropped; map tasks depend on a public geocoding back-end and their state similarity on replay is structurally low (map tiles are unnamed images).

How close the training reasoning is to the policy. On the 82 repair rows (57 error states) of the 100-task subset we scored the reasoning of each training row under the base model with the training prompt. Reasoning sampled from the policy has a mean token NLL of 0.62; teacher-written reasoning for the same rows, prompts and actions has 2.31 (paired difference $+1.69$ $[+1.59, +1.79]$, higher in all 82 rows). Among the policy's own rows, reasoning written after seeing the verified action (0.65, 56 rows) stays close to the unprompted reasoning (0.49, 22 rows), and both lie far below the teacher's reasoning for the same rows (2.38 and 2.06).

A.2 Why the policy writes the reasoning

During development, an early version of the pipeline let the teacher write the reasoning of each repair row as well as the action. A diagnostic on that version is what led us to have the policy write its own reasoning; we report it as the rationale for this design choice. On the early data, the base model's per-token loss showed that the teacher's reasoning carried 59.5% of the training loss and the corrective actions 1.8%. After training, the model predicted the corrective action almost perfectly under teacher forcing (action NLL 0.036) but produced it from its own reasoning at 23 of 97 repair states (base model: 18 of 97): it had learned the action conditioned on a prefix it does not itself produce. Replacing only the reasoning with policy-generated reasoning, with prompts and actions unchanged, lowered the loss on the reasoning from 2.61 to 0.87 (close to the 0.67 of the policy's own on-track reasoning) and raised the actions' share of the loss to 2.9%; the model trained on it reached the verified action at 33 of 96 states (one row exceeded the probe's length limit once its reasoning grew; the base model misses it too).

Controlled comparison with the current pipeline. We repeated the comparison with the current pipeline: the same 268 rows of the 100-task CFR data, in the same order, with the same recipe and training seed, where only the reasoning of the 82 repair rows is replaced by teacher-written reasoning. Over three evaluation seeds, policy-written reasoning is ahead by 4.85 pp $[+0.40, +9.29]$ on all 165 tasks, 3.86 $[-0.55, +8.26]$ on held-out tasks and 17.95 $[+6.84, +29.06]$ on the retention set, where the teacher-reasoning arm also falls below the base model (-8.55).

Relation to verified-critical-step DPO. An early version of our pipeline trained with the verified-critical-step DPO objective of CSO (Li et al., 2026), preferring the verified repair over the original erroneous action (β=0.5), on 60 pairs with templated reasoning, at four times our final learning rate and with bf16 parameters. In a WebArena pilot (one training seed, one evaluation pass on an earlier harness, 139 WebArena-Lite tasks with a valid score, task-level paired bootstrap) it reached 18.0% against 23.7% for the base policy (-5.8 pp $[-12.9, +1.4]$); on MiniWoB++, with an earlier Qwen3-4B base, DPO gave no gain over SFT on the same verified pairs. We then moved to SFT on policy-sampled reasoning with on/off-track mixing and kept counterfactual verification for admission.

B Baseline reproductions

All baselines are trained on the same 100 tasks, drawn at random from the 219 CFR training tasks (disjoint from WebArena-Lite and from the retention set), from the same base snapshot, with the CFR training recipe (App. A) and training seed; they differ only in how the training rows are produced. Rows are rendered with the inference prompt of the state they were recorded at and pass the same five gates. The student samples at the evaluation temperature (1.0) and has the evaluation step budget (30). OEC and STeP use a frontier LLM as expert or teacher; it sees the student's request (task, current observation, history) and nothing else. Table 6 lists what differs from each original.

Remaining results. Table 7 completes Table 3 with the full task set, retention against the base model, the on/off-track split of the rows, and the teacher calls and tokens; with both tables, all 15 pairwise differences against CFR (five baselines, three subsets) are reported.

Yield. OEC: 200 clean episodes, 63 resolved with the expert on, 56 of which yield 338 rows over 45 tasks (rows over 32K tokens dropped). STeP: 100 episodes, 22 kept (54 stopped at the error limit), each with one error step and one correction; the row gates drop 5 over-length corrections and 11 error steps (10 act on invalid elements, 1 is unparsable text), leaving 293 rows with 17 corrections and 11 masked error steps. RFT: 400 clean episodes, 147 successes, 145 kept after de-duplication over 65 tasks, 1,087 rows.

C Cost accounting

Unified idealised teacher cost. For every method we count all teacher calls needed to produce its training rows if every call succeeds the first time, including upstream calls, and exclude

Table 6: Deviations of the reproductions from the original recipes.

Method	Kept as in the original	Changed (and why)
OEC	switch point uniform in $[0, 30]$ counted in model queries, drawn once per episode; expert drives to the end without resetting history; only resolved episodes; only expert turns are trained (student turns are context); expert text copied verbatim	software engineering $\rightarrow$ web; switch range falls in a 30-step budget rather than a 75-call budget and the student samples at $T{=}1$ (the original uses $T{=}0$), so fewer students reach a late switch point (take-over rate 43%); two episodes per task; per-step rows instead of whole-episode masking; turns that needed a parse retry are dropped; one fabricated-success rewrite in the original converter is not applied
OEC variants	row-matched: CFR's 186 on-track rows plus 82 expert rows sampled per task (same rows and updates as CFR); + on-track: CFR's 186 on-track rows plus all 338 expert rows	added by us so that the comparison is not confounded by the absence of on-track rows or by row count
STeP	teacher judges every student step with the full history and the post-action observation; first-person reflection plus corrective action at the first error; student continues; at most one error step (the original's WebShop setting); keep successful trajectories only; erroneous step masked from the loss	no human demonstrations exist, so the initial SFT split is replaced by CFR's on-track rows and the student is the base model; a frontier LLM teacher instead of Qwen1.5-110B; our training recipe instead of the original hyper-parameters
Teacher (all)	a frontier LLM as expert or teacher	a frontier LLM as expert or teacher, called through the same action space and observation interface as the student
RFT	sample the policy on the training tasks, keep successes, de-duplicate by action signature with the original farthest-edit-distance rule, train on all kept trajectories	$k{=}4$ samples per task instead of 100; the sampler is the base model (no prior SFT stage exists); temperature 1.0 (the original release also includes $T{=}1$ samples)

Table 7: Controlled comparison on the same 100 tasks: full task set, retention against the base model, on/off-track rows, teacher calls and input tokens (same arms, evaluation and cost count as Table 3). Δ in pp with 95% task-level paired bootstrap CI; bold: CI excludes 0.

	Rows	Teacher		Full (165)		Retention (39)
Method	(on/off)	calls	input tok.	vs. base	vs. CFR	vs. base
CFR (ours)	186/82	114	0.48M	**+4.44** [+0.61, +8.28]	—	+9.40 [+0.00, +17.97]
OEC, original	0/338	409	6.67M	+3.64 [−0.20, +7.68]	−0.81 [−5.05, +3.23]	+0.00 [−8.55, +8.55]
OEC, row-matched	186/82	360	5.96M	+2.02 [−2.22, +6.26]	−2.42 [−6.26, +1.41]	+1.71 [−8.57, +11.97]
OEC + on-track	186/338	409	6.67M	+3.03 [−0.81, +7.07]	−1.41 [−5.86, +2.83]	+0.85 [−7.69, +10.26]
STeP	276/17	94	2.28M	+2.83 [−1.01, +6.67]	−1.62 [−5.26, +2.02]	+2.56 [−6.84, +11.97]
RFT	1,087/0	0	0	+3.64 [−0.20, +7.47]	−0.81 [−4.85, +3.03]	+5.98 [−4.27, +16.24]
Full demos (est.)	—	916	11.7M	not trained		

failed episodes, rejected candidates and hidden reasoning tokens; inputs and visible outputs are retokenised with one tokenizer (o200k). CFR: one localisation call and one repair call per admitted state. The localisation input is the localisation instruction plus the episode's actions, URLs and reasoning tails; the repair input is the repair instruction plus the packet the teacher saw (task, prefix actions, the policy's action and reasoning at s_w, the observation at s_w and any earlier observation the localiser points to). The policy-generated reasoning is not a teacher cost. OEC: all expert calls of every episode that yields rows. STeP: all judge calls in its kept episodes; each correction is written by the judge call at the error step. Full demonstrations: one successful demonstration per task. RFT has no teacher. Per call, CFR averages 4.2K input tokens (localisation 2.1K, repair 6.3K), STeP 24.3K, OEC 16.3K and a demonstration step 12.8K. Table 8 gives the full numbers.

Full-demonstration estimate. No model was called. For each task we take all available successful trajectories with recorded observations (from any policy run on that task in our logs), substitute each step's observation into the exact expert prompt of our OEC reproduction, count tokens, and take the per-task median number of calls and input tokens; tasks with only step counts use the per-step median of the same site. As for the other methods, every demonstration is assumed to succeed ($p{=}1$); a demonstration step's prompt is long because it carries recent observations.

Per teacher row. Per training row that carries teacher output, CFR spends 5.9K input tokens (localisation and repair), OEC (original) 19.7K and STeP 134K.

Table 8: Teacher cost on the 100-task pool, unified idealised count (o200k, visible tokens only). The last rows are the main arm (219 tasks), for reference.

Method	Calls	Input tok.	Output tok.	Input × CFR
CFR (localisation + repair)	114	481,332	3,475	1.0
localisation / repair	57 / 57	121,806 / 359,526	2,223 / 1,252	—
STeP	94	2,282,227	1,706	4.7
OEC, row-matched	360	5,964,677	18,399	12.4
OEC, original and + on-track	409	6,672,476	20,767	13.9
RFT	0	0	0	0
Full demonstrations (estimate)	916	11,711,050	44,860	24.3
CFR, main arm (219 tasks)	234	966,926	7,035	1.0
Full demonstrations, main arm (estimate)	1,893	22,803,964	92,757	23.6

Counting rejected candidates. The idealised count charges one successful teacher call per training row. We also charge CFR for every verified repair candidate its teacher wrote on a state up to the first admitted one, including all candidates on states that never admitted any, and charge the baselines for all of their calls, including episodes that produced no training rows; each candidate is charged as one repair call of average size. With the teacher we used, this is 9.9 candidates per admitted state (a pool-wide count; App. A relates it to the 1.62 versions per training state). On the 100 tasks CFR then needs about 565 repair calls and 3.56M input tokens, or 789 calls and 4.09M with one localisation per failed episode, against 547 calls and 9.46M for OEC and 461 calls and 9.34M for STeP. Under this count CFR needs about 2.7× less teacher input than OEC and 2.6× less than STeP (2.3× for both with localisation; 2.7–3.0× across ways of counting duplicate annotations), at a similar number of calls. This cost scales with the number of candidates per admitted state, so a teacher whose repairs pass verification more often lowers it. Review calls are not counted in any of our cost figures.

Environment rollouts and training compute. Table 9 counts the completed environment rollouts each method used on the 100 tasks and the GPU time of its training run. CFR's count includes mining, the collection episodes behind its on-track rows (an upper bound) and $K{=}4$ verification; the rollouts run on self-hosted sites, so they cost compute but no teacher tokens. Training runs used one GPU; H100 and A100 hours are listed separately because runs were scheduled on both. Serving time for data collection was not logged per method.

Table 9: Environment rollouts (completed) and training GPU time on the 100-task pool. Row-matched OEC and STeP also train on CFR's on-track rows, whose collection episodes are counted in the CFR row. Each CFR candidate needs at most $2K{=}8$ verification rollouts. The amortised CFR row charges the 100 tasks their share (57 of 117 training states) of all verification and mining rollouts in the pool, counting each candidate once or as run.

Method	Rollouts	Policy steps	Training GPU time
CFR (on these tasks)	1,519	15,534	3.19 H100-h
CFR (amortised per training state)	4.2K–6.2K	—	—
OEC, original	232	2,202	10.41 A100-h
OEC, row-matched	232	2,202	4.40 H100-h
STeP	111	571	4.49 H100-h
RFT	401	5,143	14.76 A100-h

D Additional results

D.1 Success from error states on unseen tasks, by site

The test uses the CFR model trained with the main recipe (Table 2). The test set starts from the 121 held-out tasks. Five were solved by the base model in all three source episodes and have no failing episode; for each of the other 116 we sampled one failing episode. The localiser judged 24 of them

locally unrepairable (§3; mostly a premature final answer, a browser error page after leaving the site, or an unreachable target), placed the error at the initial state in 6 (excluded here because the error state and the initial state would coincide), and 6 prefixes could not be rebuilt; 60 of the remaining 80 were sampled. No failure was left without a localised error state. The 24 are first-pass verdicts; the 4 such tasks in App. A.1 went through an additional review pass. Three tasks were removed because the prefix did not replay on fresh containers, and one task remains unresolved because both models repeatedly hit the wall-clock cap on it. On that task the base model completed its cell with 1 success in 4 rollouts; CFR never completed four rollouts within the 5,400 s cap, and its partial rollouts contain 1 success in 4; from the initial state both models score 0 of 4. The extreme bound that assigns the task entirely to the base model, $+5.70$ $[-1.75, +13.16]$, scores it 4/4 for the base model and 0/4 for CFR, which contradicts the base model's observed 1/4. With the base model at its observed 1/4, the lower bound stays above zero whether CFR is scored 0/4, its observed 1/4, or 4/4 ($+7.02$ $[+0.87, +14.04]$, $+7.46$ $[+1.32, +14.47]$ and $+8.77$ $[+2.19, +15.79]$). Rollouts that hit the cap were at most 3.2% per cell. The replayed state matches the recorded one exactly in the median (Jaccard overlap of the multisets of (role, accessible name) pairs of the page elements, median 1.0); values below 0.8 occur almost only on map pages, whose tiles are unnamed images. Both models continue from identical prefixes, so these effects do not differ between arms. Figure 3 breaks the difference down by error depth and site.

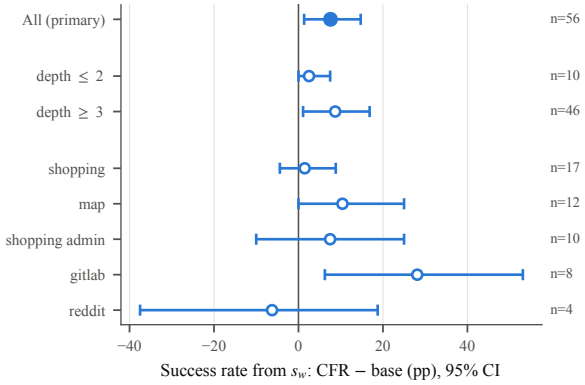

Figure 3: Success from base-generated error states on unseen tasks, CFR minus base, by error depth and site (sites with at least 4 tasks). Filled: primary readout.

D.2 Per-seed results and sensitivity of the main result

Figure 4 shows the per-seed held-out and full-set differences behind Table 2. The pooled interval in Table 2 resamples tasks with the training seeds fixed. Resampling training seeds and then tasks (two-level bootstrap, B=5000) widens it to $[+2.48, +11.66]$ on the 121 held-out tasks and $[+0.67, +11.11]$ on all 165; both still exclude zero. A t-interval over the three per-seed differences (two degrees of freedom, task set held fixed) is $[-0.29, +13.69]$ on held-out tasks. The pre-specified analysis is the task-level interval; these are sensitivity analyses.

Held-out tasks by template. Table 10 splits the held-out tasks by whether a training task shares their WebArena template (`intent_template_id`). The main arm's held-out gain comes mainly from the 53 tasks that share one ($+11.53$ pp $[+5.66, +17.61]$); on the other 68 it is $+2.94$ pp $[-0.82, +6.86]$. This pattern is not a floor effect: among held-out tasks the base model never solved in an independent evaluation, the same split gives $+12.25$ $[+5.41, +19.37]$ (39 tasks) against $+3.07$ $[-0.77, +7.28]$ (58 tasks). The 100-task CFR arm gains $+16.67$ pp $[+1.67, +30.00]$ on 20 tasks whose templates belong to the 219-task pool but not to its own 100 tasks, which suggests that the split tracks task families the pool covers rather than exposure to a template as such.

D.3 Ablations

A second training seed for the controlled comparison. We trained CFR, OEC (row-matched), STeP and the on-track-only arm below on the same 100 tasks with a second training seed and pooled

Table 10: Held-out gain by template overlap with the training tasks. Δ vs. base in pp, 95% task-level paired bootstrap; bold: CI excludes 0. Groups within a split are disjoint; the difference row resamples each group independently.

Arm	Group of held-out tasks	n	Δ
CFR, three seeds pooled	template shared with a training task	53	**+11.53** [+5.66, +17.61]
	no shared template	68	+2.94 [−0.82, +6.86]
	difference	—	**+8.59** [+1.41, +15.61]
CFR, 100 tasks (Table 3)	template shared with its own 100 tasks	33	**+8.08** [+1.01, +15.15]
	no shared template	88	**+5.30** [+0.76, +9.85]
	template in the 219-task pool, not in its 100	20	**+16.67** [+1.67, +30.00]

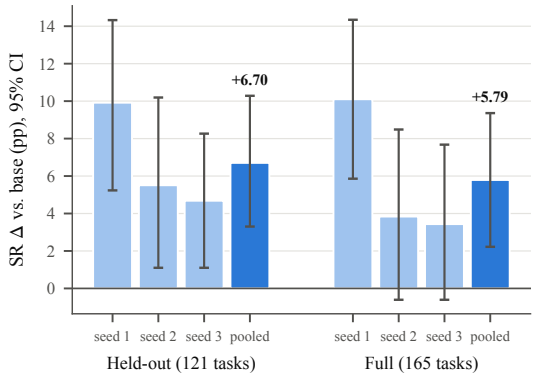

Figure 4: CFR minus base on WebArena-Lite for each training seed and pooled (95% CI).

both seeds (Table 11). CFR keeps the highest gain on every subset: +7.02 pp [+1.65, +12.81] on held-out tasks, ahead of OEC (row-matched) by 5.79 [+1.24, +10.33] and of STeP by 2.89 [−0.83, +7.02], and on the retention set ahead of OEC (row-matched) by 14.1 [+3.9, +24.4].

On-track rows alone. From the 268 rows of the 100-task CFR data we removed the 82 repair rows and trained on the remaining 186 on-track rows with the same recipe, two training seeds for each arm, evaluated on the same Lite165 seed. Against this on-track-only arm, CFR is ahead by +4.13 pp [+0.41, +8.26] on held-out tasks, +2.42 [−2.12, +7.27] on all 165, and +12.8 [+1.3, +24.4] on the retention set; the on-track-only arm itself is +2.89 [−1.65, +7.44] above the base model on held-out tasks. The on-track-only arm trains on 186 rows, CFR on 268. The same two arms were also run from the base model's error states on the unseen tasks of §5.3 (56 tasks, K=4 each). CFR succeeds in 17.0% of rollouts and the on-track-only arm in 12.5%, +4.46 pp [−1.81, +11.61], with CFR better on 7 tasks and worse on 4. Against the base model's rollouts from the same states (Table 4), the on-track-only arm stays at the base model's level (−1.34), while CFR is above it (+3.12), so the gain at the points where the agent fails comes with the repair rows.

D.4 MiniWoB++

4B, current pipeline, selected templates (§5.4). Training templates were chosen by a rule that uses training-side information only and was pre-specified in a dated document before any data of this run was collected: among templates with base success between 0.2 and 0.8 on the mining seeds, those for which the pipeline admitted at least five verified repairs (eight templates: choose-list, click-checkboxes-large, click-collapsible, click-collapsible-2, copy-paste, search-engine, text-transform, use-autocomplete). With at most ten repairs per template, CFR trains on 267 rows (80 repair rows from 78 admitted states, 187 on-track rows); the RFT control uses the same 187 on-track rows plus 80 further successful rows. Each arm was trained with three seeds and evaluated on all 50 templates, with 32 new environment seeds on the eight training templates and 16 on the other 42 (template-cluster bootstrap, 10,000 resamples). On the training templates the three arms reach 52.3% (base), 74.2% (CFR) and 62.0% (RFT); per training seed, CFR gains +38.7, +25.4 and +1.6 pp. CFR

Table 11: The controlled comparison with two training seeds per arm. Every entry pools both training seeds of its method (the seed of Table 3 and a second one). Each training seed is evaluated once on Lite165 with evaluation seed 40 ($K{=}1$) and paired with the base model's run on the same evaluation seed. Δ in pp, 95% task-level paired bootstrap CI; bold: CI excludes 0.

	Held-out (121)		Full (165)		Retention (39)	
Method (two seeds)	vs. base	vs. CFR	vs. base	vs. CFR	vs. base	vs. CFR
CFR (ours)	**+7.02** [+1.65, +12.81]	—	**+6.36** [+0.61, +12.12]	—	**+16.67** [+2.56, +32.05]	—
OEC, row-matched	+1.24 [−3.72, +5.79]	**−5.79** [−10.33, −1.24]	+1.52 [−4.55, +7.58]	**−4.85** [−9.39, −0.30]	+2.56 [−14.10, +19.23]	**−14.10** [−24.36, −3.85]
STeP	+4.13 [−1.65, +9.92]	−2.89 [−7.02, +0.83]	+5.76 [−0.30, +11.82]	−0.61 [−4.55, +3.33]	+7.69 [−8.97, +24.36]	−8.97 [−19.23, +0.00]
CFR, on-track rows only	+2.89 [−1.65, +7.44]	**−4.13** [−8.26, −0.41]	+3.94 [−2.42, +10.00]	−2.42 [−7.27, +2.12]	+3.85 [−14.10, +23.08]	**−12.82** [−24.36, −1.28]

produced no malformed actions in any seed. Results are reported separately for the eight selected templates and the other 42. After the three-seed readout we trained and evaluated five more seeds per arm as a supplementary readout outside the pre-specified criteria (same template-cluster bootstrap). Over all eight seeds, CFR minus base is $+23.73$ pp $[+9.08, +38.57]$ on the eight selected templates and -0.24 $[-2.08, +1.75]$ on the other 42. On those 42 untrained templates, RFT with the same number of rows changes success by -4.58 $[-9.64, -0.37]$ relative to the base model, and CFR is ahead of it by $+4.33$ $[+0.69, +8.72]$; the repair rows cost no generalisation, and replacing them with successful rows does. On the selected templates CFR minus RFT is $+7.91$ $[-7.81, +23.98]$.

4B, early prototype. Ten task templates on which the 4B base model fails at a moderate rate were used for data; error states were placed by rule at the deepest recorded state of 61 failed episodes, a teacher wrote repairs of up to four actions together with the reasoning for them (24 were declined because the task payload was not in the observation), and $K{=}4$ two-arm verification admitted 31 of 37. The trained model and the base model were evaluated on all 50 templates with 16 environment seeds not used anywhere in training (800 paired episodes, reasoning mode off, temperature 0.3, 15 steps). Success rises from 347 to 398 of 800; the gain is on the training templates. On WebArena, the corresponding recipe (verified repairs with teacher-written reasoning, same base model, an earlier evaluation harness) gave held-out point estimates of $+2$ to $+3$ pp with intervals containing zero, and multi-step repairs lowered success on the retention set; App. A.2 gives the diagnostic that explained it and led to policy-written reasoning.

4B, current pipeline, fixed split. We also ran the current pipeline on Qwen3.5-4B with the same 25/25 template split as the 2B run below, without selecting templates: localisation and 1–3 action repairs, $K{=}4$ verification (95 admitted, 51 low-efficacy, 3 high-baseline), and 343 training rows (103 repair, 240 on-track), with three training seeds. On new seeds of the training templates, CFR raises success over the base model by $+8.17$ pp $[+1.67, +15.42]$ averaged over the three seeds, and every seed gains ($+6.25$, $+11.25$, $+7.00$). An RFT control with the same 240 on-track rows plus 103 further successful rows trails CFR by 5.17 pp $[-3.00, +15.00]$ on these templates, and on the untrained templates it loses 11.75 pp to the base model $[-24.75, -0.75]$ while CFR does not ($+2.33$ $[-6.75, +11.33]$).

2B, current pipeline. With Qwen3.5-2B, 50 templates were split by base success rate into 25 training and 25 held-out templates. Localisation and 1–3 action repairs by the teacher and $K{=}4$ verification gave 70 admitted, 48 low-efficacy and 8 high-baseline verdicts; CFR trains on 270 rows (81 repair, 189 on-track), and an RFT control on the same 189 on-track rows plus 81 additional successful rows. Table 12 gives the result: neither pre-specified criterion (an interval for CFR minus base above zero on the training templates, and on the held-out templates) passed, and RFT obtains the same gain on training templates.

Table 12: MiniWoB++ with Qwen3.5-2B, one training seed per arm, 50 templates × 16 new seeds, template-cluster bootstrap. SR in %.

Split	Base	CFR	RFT	CFR−base	RFT−base	CFR−RFT
Training templates (25)	41.3	48.8	49.0	+7.50 [−2.25, +18.25]	+7.75 [+0.00, +16.50]	−0.25 [−7.00, +5.75]
Held-out templates (25)	39.3	36.3	33.8	−3.00 [−11.75, +4.75]	−5.50 [−15.75, +2.50]	+2.50 [−1.75, +6.75]
All (50)	40.3	42.5	41.4	+2.25 [−4.50, +9.00]	+1.13 [−5.63, +7.38]	+1.13 [−2.75, +4.75]

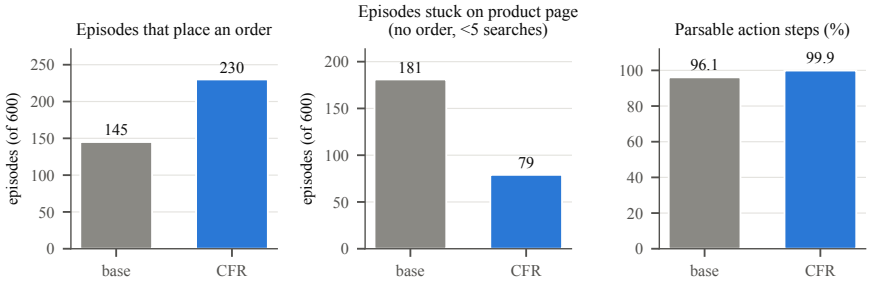

Figure 5: WebShop, 600 test episodes per model (200 tasks × 3 seeds): CFR mostly converts episodes that stall on a product page into orders.

D.5 WebShop details

The environment is the AgentGym WebShop (small product set) with a fixed seed for goal generation; the 200 test tasks are AgentEval's split, disjoint from the 600 training tasks drawn from the AgentGym training split (products are shared between the two, as in the original setup). The policy sees the available actions, writes reasoning and an action, and has 15 turns. Model selection followed a rule fixed in advance (success rate in $[0.08, 0.60]$ and reward ≥ 0.20): the 2B model reached 2.0% success and the 4B model 13.0% success with reward 0.194, 0.006 below the reward threshold. Neither size met the rule as written; we admitted the 4B model on the success-rate criterion before any repair or training data existed, and record this as a deviation from the rule. Of the 91 admitted repairs, 56 complete the task (typically choosing a product option and buying) and 35 hand control back. Published WebShop numbers use other models, prompts and turn limits.

D.6 Zero-shot transfer details

WorkArena L1: 33 templates, one seed each ($K=1$), same controller; the interval uses a template bootstrap. AssistantBench: 32 development tasks (one task crashed deterministically in both arms), $K=3$, live web with a search-engine start page; 32–36% of episodes met a blocking page.

D.7 Live websites: Online-Mind2Web feasibility

We also tried the pipeline on the live web with Online-Mind2Web (Xue et al., 2025). Live sites block automated clients and change between visits: 81 of the 300 tasks had a start page blocked by a CAPTCHA or bot check, 79.5% of rollouts replayed their prefix, and about one third of recorded element identifiers were still valid on replay. Such sites do not provide the reproducible states that a counterfactual test, or any replay-based training, requires. The base model solved 20 of 59 training-seed tasks (33.9% under the benchmark's LLM judge); of the 22 teacher repairs that reached verification, 6 failed to replay and 4 to find their target element on the drifted page, 5 were cut off by a judge-service outage and 7 had low efficacy, so none was admitted. We therefore train only on self-hosted benchmarks, where we rebuild each state by a controlled reset from fresh containers and still check every replay for a complete prefix and matching elements (App. A).