CLOSING THE SPECIALIZATION GAP BETWEEN COMPUTER USE BENCHMARK SCORES AND THE REAL WORLD PERFORMANCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Strong general computer-use performance does not guarantee reliability in a narrow production workflow. We study this specialization gap in browser-based booking, where an agent operates from screenshots, asks for missing information, and must report the outcome truthfully. Although Qwen3.5-35B-A3B reports 54.5% on OSWorld-Verified, it performs poorly under our all-or-nothing booking criterion. We adapt it in two stages: supervised fine-tuning (SFT) teaches the production protocol, tool use, and user interaction, and reinforcement learning (RL) aligns complete trajectories with product logic. Training uses a browser harness that reproduces the production model-facing protocol and an Iron User simulator that answers clarification questions while updating the task ground truth. Evaluation combines local LLM judgments over screenshots, actions, and dialogue with deterministic resolution. On the available evaluation subset, SFT raises task success from 16.7% to 74.1%, and the SFT-plus-RL model outperforms all unadapted systems in the reported RL comparison.

1 INTRODUCTION

Browser-based booking requires an agent to interpret a request, operate a live interface from screenshots, ask for missing information, follow product logic, and report the outcome truthfully.

We study specialization rather than model selection. Qwen3.5-35B-A3B (Qwen Team, 2026b), used in our main experiments, reports 54.5% on OSWorld-Verified, yet its unadapted checkpoint often fails our exact booking criterion.

We hypothesize that this weakness is an adaptation gap rather than an absence of general computer-use ability. We test it with a two-stage pipeline: SFT teaches the target tools, browser interaction, and communication behavior; RL then optimizes complete trajectories for form correctness, dialogue quality, truthful reporting, and protocol compliance.

Our contributions are:

1. a browser harness that exposes the same model-facing prompt, tools, action syntax, and observation structure as the production agent.
2. Iron User, a simulated user that continues clarification dialogues and updates the task ground truth consistently with its answers.
3. an evaluation protocol that turns local, screenshot-grounded LLM judgments into a deterministic product-level verdict.
4. SFT and long-horizon GRPO training with task-specific data, reward design, and stabilization.

SFT improves task success from 16.7% to 74.1%, near Claude Opus 4.8 at 76.0%; these point estimates do not establish statistical equivalence. SFT plus RL achieves the highest overall reward in the RL comparison (0.725 versus 0.267 for the strongest unadapted Qwen), while stabilization delays entropy growth and policy collapse.

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.

2 Related Work

Computer-use environments and models. WebArena (Zhou et al., 2023) introduced self-hosted websites with programmatic evaluation, and OSWorld (Xie et al., 2024) extended execution-based evaluation across desktop and web applications. Native computer-use models such as Fara-7B (Awadallah et al., 2025) act directly on rendered interfaces.

Demonstrations and conversational navigation. Mind2Web (Deng et al., 2023) provides crowd-sourced web action sequences, WebLINX (Lù et al., 2024) adds multi-turn dialogue, and Open-CUA (Wang et al., 2025b) introduced the cross-platform AgentNet pipeline used in our SFT mixture.

SFT and RL for web agents. Imitation learning commonly initializes a policy before online optimization; WebSTAR masks low-quality SFT steps. WebAgent-R1 (Wei et al., 2025), OpenWebRL (Yang et al., 2026), and OpAgent (Guo et al., 2026) apply online RL to web interaction.

Optimization and stability. PPO (Schulman et al., 2017) introduced clipped policy updates, while DeepSeekMath (Shao et al., 2024) introduced critic-free, group-relative GRPO. Related work studies entropy-associated collapse (Agentic Entropy-Balanced Policy Optimization (Dong et al., 2025)) and alternative objectives including GSPO (Zheng et al., 2025), DPPO (Qi et al., 2026), and LLDS (Deng et al., 2025).

Trajectory verification. WebArena checks page and backend state, while WebJudge (Xue et al., 2025) selects evidence-bearing trajectory frames. Because neither fixed rules nor LLM evaluators are uniformly reliable (AgentRewardBench (Lù et al., 2025)), OpenWebRL and OpAgent combine learned judgments with deterministic checks, as we do.

Users inside agent trajectories. tau-bench (Yao et al., 2024) places an LLM user in a stateful tool episode; AppWorld-UL (Chen et al., 2026) preconstructs hidden knowledge and acceptable alternatives; and UserRL (Qian et al., 2025) trains with simulated users. Iron User instead updates task ground truth during clarification while enforcing consistency between replies and updates.

Language robustness. Multilingual web interaction remains challenging (X-WebAgentBench (Wang et al., 2025a)), motivating our treatment of unintended language switching as a failure signal.

3 Task Formulation

We study online booking, in which an agent reserves a table, appointment, or service through a venue's website. The task permits precise success criteria but still requires navigation across diverse interfaces, constraint tracking, clarification, and transaction completion.

The agent receives a request such as "Book a table at …" and operates the rendered website with browser-control tools. It may ask the user for missing information and must eventually report either success or a justified inability to complete the request. Appendix A specifies the tool interfaces and terminal-action rules.

The product criterion is all-or-nothing. Success requires either submitting a booking whose required fields match the resolved request or correctly explaining why the request cannot be completed. Partial form completion, unconfirmed alternatives, stopping before submission, and unsupported success claims receive no credit.

3.0.1 Why train a dedicated model?

Open models that meet our size and latency constraints perform inadequately on this task despite credible general benchmark scores. Qwen3.5-35B-A3B reports 54.5% on OSWorld-Verified and Qwen3.8-27B reports 84.3%; these values provide capability context, not directly comparable booking baselines.

We adapt Qwen3.5-35B-A3B in two stages:

1. **SFT** teaches the environment, tool protocol, and user interaction.

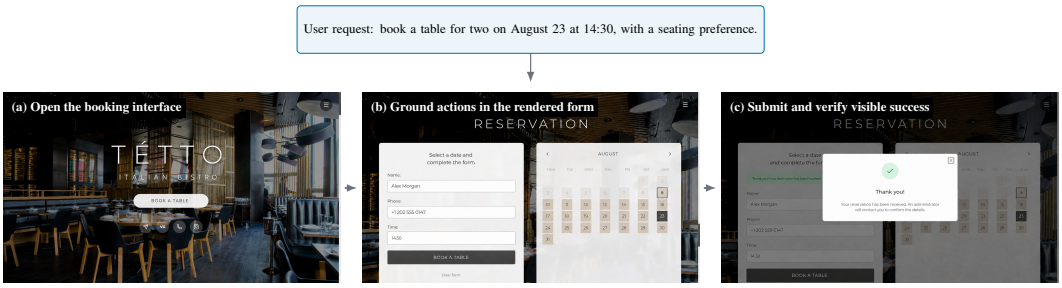


Figure 1: Representative `good_query` trajectory from the training environment. The policy operates only the rendered interface, fills the requested fields, submits the form, and observes visible confirmation. The displayed name and phone number are fictional.

2. **RL** aligns complete trajectories with product logic.

4 Agent and Interaction Design

4.1 Agent Harness

4.1.1 Overview

Each episode starts in a fresh browser session with the task description and an initial screenshot. At every turn, the model emits free-form reasoning and one text-encoded tool call; the harness executes it and returns a textual result and a new screenshot. The model never receives a privileged DOM or accessibility tree.

The training harness reproduces the production agent's model-facing prompts, tool schemas, action syntax, and separation of browser, dialogue, and terminal actions. Production maintains longer-lived sessions and sends clarifications to a real user; training isolates each rollout and can connect it to a simulated user.

4.1.2 Model-facing tools

The `computer_use` tool exposes screenshot, pointer, click, drag, type, keyboard, scroll, and wait actions, grounded in normalized 1000-by-1000 pixel coordinates. The model has no terminal, arbitrary code execution, or model-visible DOM.

Its `answer` action asks the user for information that cannot be inferred safely. It is a clarification, not task completion: production returns control to the real user, while training obtains a reply from the simulator and continues in the same browser session. The simulator may update the ground truth when the dialogue changes the user's requirements.

Three task-level operations complete the interface. `get_date` deterministically resolves temporal expressions relative to the episode date. `booked` reports success after GUI submission but does not submit the form itself. `giveup` terminates an impossible task. Production may also expose `take_control` for authentication, payment, or CAPTCHAs; it is disabled during training.

4.1.3 Interaction protocol

A rollout follows a reasoning–action–observation cycle. The initial context contains system instructions, tool definitions, the request, and a screenshot. After each GUI action, the harness appends separate textual execution and screenshot observations in the model's expected format.

Only one action executes per turn. Recoverable errors become observations that the model can correct; unrecoverable browser, serving, or state errors terminate the rollout.

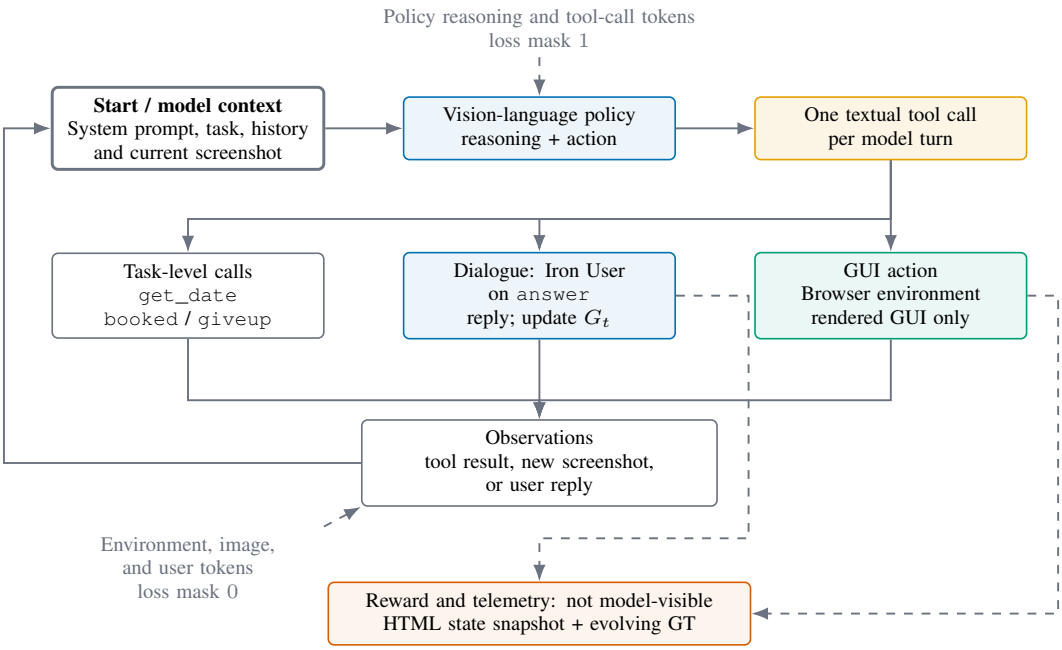


Figure 2: Model-facing harness and training interface. The bold entry block marks the starting context. Solid arrows show the interaction loop; dashed arrows carry reward-only state or connect loss-mask annotations to the corresponding tokens.

An `answer` call leaves the browser unchanged while the simulator replies, allowing multiple rounds of clarification within a trajectory. The episode ends on `booked`, `giveup`, an unrecoverable error, an external stop, or budget exhaustion.

4.1.4 Technical constraints and training interface

Internal browser state is retained only for rewards and telemetry. Fresh sessions prevent state leakage, and a fixed episode date makes relative-date queries and simulated-user behavior repeatable.

Turn and token limits, generation timeouts, limited retries, and screenshot validation bound compute and distinguish infrastructure failures from task failures. Optional click validation rejects unsafe or inconsistent clicks.

For training, agent reasoning and tool-call tokens receive a policy-loss mask of one; tool results, screenshots, and simulated-user messages remain in context with a mask of zero. An optional mask excludes clarification turns in ablations. Thus the policy learns from its decisions while conditioning on the complete multimodal history.

4.2 Iron User Simulator

4.2.1 Motivation

Booking often requires information omitted from the initial request or agreement on alternatives when a slot is unavailable. A static task cannot respond to such events, while GRPO requires complete trajectories in which an interlocutor answers the policy's specific questions.

Iron User (IU) is an LLM-driven simulator embedded in trajectory collection. It replies whenever the policy calls `answer`; terminal actions remain the policy's responsibility.

4.2.2 FORMULATION

IU implements $u_t,\ \Delta G_t = f_\theta(a_t,\ G_{t-1},\ d,\ \tau)$, where d is the initial task and τ describes the form fields. It returns a reply u_t and updates to the ground truth G_t. Any field value stated in the reply must be updated in the same turn, and updates are restricted to existing keys. Appendix B gives the full definition and persona parameters.

4.2.3 HONESTY PENALTY AND DETERMINISM

A separate LLM judge penalizes false claims, language switching, and irrelevant communication as $r = r_{\text{task}} - \lambda \mathbb{1}[\text{lying}]$, supplemented by visual and consistency checks. Dataset ground truth, allowed form values, overrides, and a fixed seed make IU reproducible. Appendix B provides the algorithm, an example, and human evaluation.

5 EVALUATION PROTOCOL

Training reward is tuned to the simulation and does not guarantee that a session satisfies the request. We therefore evaluate complete sessions with an independent binary product criterion; partial progress receives no credit.

Rather than ask an LLM to label a long trace directly, we ask short, evidence-grounded questions and combine the local judgments deterministically. This reduces missed visual details, hallucinated events, and inconsistent rule application.

The product-oriented evaluation contains **399 tasks across 397 website hostnames** selected from the generated environment pool. Requests sampled from structured templates vary venue, party size, booking reason, date, time, and contact details, and passed automatic checks and human review. The set has no public sampling frame, measured training overlap, or repeated-run variance, so we use it for within-protocol comparisons rather than direct comparison with public benchmarks.

5.1 BENCHMARK

We execute one browser session per task. The denominator remains 399: only a correct completion receives credit; `Unknown`, unsuccessful requests, model unavailability, infrastructure failures, and all other non-successes score zero.

The automatic Judge evaluates every trajectory; experts then assess the most promising configurations.

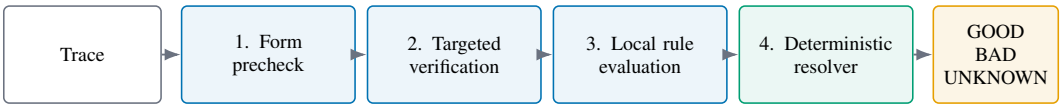
5.2 EVALUATION INPUTS

A trace contains step screenshots, browser operations and coordinates, and the complete dialogue. Appendix C describes their common representation and screenshot packing.

5.3 HYBRID JUDGE PIPELINE

The primary Judge uses GPT-5.4 (OpenAI, 2026a), snapshot `gpt-5.4-2026-03-05`, in four stages:

1. **Form precheck.** The model reconstructs a structured card describing the form, entered values, available options, submission state, and visible outcome.
2. **Fact verification.** Evaluation-critical details are rechecked with narrow questions over selected full-resolution frames, reducing errors caused by small elements in long screenshot sequences.
3. **Rule evaluation.** The model determines which predefined product rules fired and cites supporting frames. It makes local judgments but does not choose the final label.
4. **Deterministic resolution.** Code applies ordered priority rules to produce `GOOD`, `BAD`, or `Unknown`. Blocking technical uncertainty takes precedence over substantive errors.



Input: screenshots, actions, and dialogue; frames are annotated and packed. **LLM stages (1–3):** construct a case card with frame citations, verify full-resolution evidence, and evaluate 84 local rules. **Resolver (4):** aggregate fired rules and evidence using an ordered priority specification; no model call is used for the final verdict.

Figure 3: Hybrid evaluation pipeline. Language models extract and verify local evidence, while deterministic code resolves fired rules into the final verdict.

Single-stage judging was less consistent in development, but we do not report a controlled ablation; the staged pipeline is therefore an engineering choice rather than a quantified improvement.

Each judgment cites visible evidence, fired rules explain the verdict, and evaluation-policy changes occur in a versioned catalog. On a golden set of model-generated trajectories scored automatically before human labeling, with disagreements adjudicated by experts, the Judge reaches **82% accuracy** and **0.78 macro-F1**. Appendix C gives the validation protocol, class-level results, rules, and resolver.

6 Supervised Fine-Tuning

SFT first teaches the base model to operate the environment, call tools, and interact with the user.

6.1 Data format, collection, and filtering

SFT data comprise trajectories of up to 50 steps from a larger model and from editors using a browser extension. Model generation produced **4,908** trajectories, of which **3,009** remained after automatic and manual filtering; justified refusals also count as correct. A second editorial group independently reviewed editor-recorded trajectories. Appendix D details collection and filtering.

6.2 Training

The mixture also includes filtered and converted **AgentNet** examples to retain broader web-interface skills.

Training used a modified **veRL** stack on 32 NVIDIA A100 GPUs. Runs took five to seven hours (**160–224 A100 GPU-hours**). By peak dense BF16 throughput (312 TFLOP/s for A100 (NVIDIA, a) and 989.5 TFLOP/s for H100 SXM (NVIDIA, b)), this is **50–71 H100 SXM GPU-hours**, a theoretical compute equivalent rather than measured speedup.

6.3 Results

Table 1 compares the base and fine-tuned versions of Qwen3.5-35B-A3B with Claude Opus 4.8 on a booking subset derived from model-generated trajectories.

Table 1: Strict task success on the booking assessment. Qwen3.5-35B-A3B is evaluated before and after SFT; Claude Opus 4.8 is a proprietary comparator.

Model	Configuration	Task success
Qwen3.5-35B-A3B	Base model	16.7%
Qwen3.5-35B-A3B	SFT on collected data	74.1%
Claude Opus 4.8	Proprietary comparator	76.0%

SFT raises task success from **16.7% to 74.1%** (**+57.4 percentage points**) with the checkpoint fixed, directly isolating the effect of task adaptation.

Claude Opus 4.8 reaches **76.0%**, 1.9 points above the SFT model; these point estimates do not establish statistical equivalence.

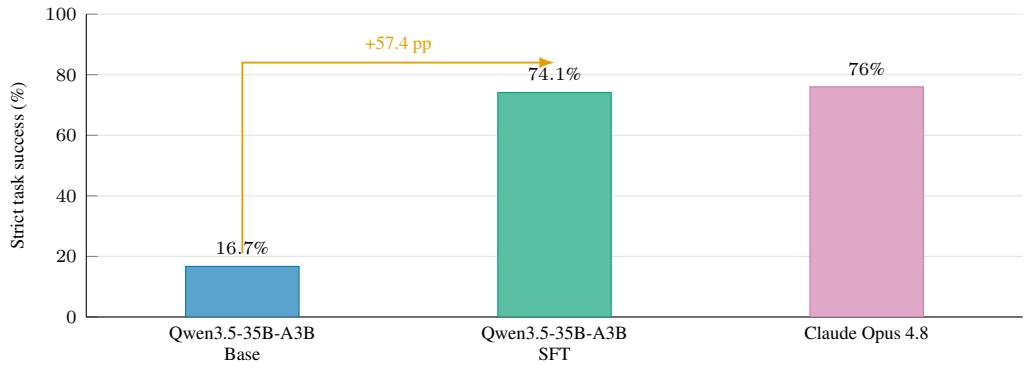


Figure 4: Strict task success for Qwen3.5-35B-A3B before and after supervised fine-tuning, with Claude Opus 4.8 as a proprietary comparator.

7 Reinforcement Learning

SFT does not directly optimize complete interaction outcomes. We therefore apply RL to full trajectories initialized from either the base or SFT checkpoint, while seeking to preserve dialogue and tool-use capabilities.

7.1 Training Setup

GRPO uses groups of eight trajectories per task, 32 tasks per batch, and 256 asynchronous agent workers. Each episode runs in fresh headless Chrome for up to 50 assistant turns. The model sees **1280** $\times$ **720** screenshots, while form telemetry is reserved for rewards.

The Section 4 protocol lets the policy operate the browser, ask IU for information, normalize dates, and terminate with `booked` or `giveup`. IU updates the ground truth so rewards assess the resolved request.

Training uses `temperature=0.7` and `top_p=0.95`; validation uses `temperature=0` and one trajectory per task. Appendix E gives the remaining configuration.

7.2 Reward Design

The reward must distinguish correct completion from partial forms, conflicting submissions, mishandled clarifications, unsupported success claims, and protocol violations.

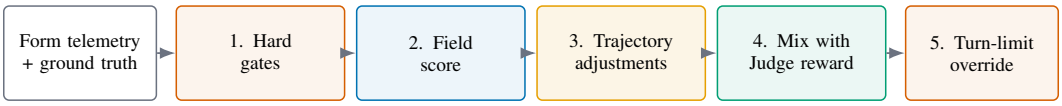
We combine three complementary signals:

1. **Deterministic outcome.** Form telemetry checks backend acceptance and required fields against evolving ground truth, with normalized or task-appropriate matching and scenario-specific valid `giveup` outcomes.

2. **Trajectory judgment.** The multimodal Judge evaluates screenshots, actions, and dialogue; `Good` contributes one and `Bad` or `Unknown` zero.

3. **Behavior adjustments.** Additional terms cover false claims, repeated clarifications, unnecessarily long successes, loops, invalid calls, and disagreement between `booked` and the submitted form.

The final reward equally weights the adjusted environment score and Judge contribution. Turn-limit exhaustion overrides it with **-1.0**; an absent or unparsable tool call terminates the trajectory. Appendix E gives all branches and coefficients.



Form reward (1–2): compare submitted-form telemetry with evolving ground truth G_T. Backend success, date, time, guests, and phone are hard gates; any failure sets the form reward to zero. Otherwise, the field score is $1 - \min(\sum_i p_i, 1)$, with penalties for name and additional fields. **Adjustments (3):** `booked` mismatch -0.5; missing terminal marker -0.2; then honesty, looping, date conflict, parsing, reask, and length adjustments, capped at 1, produce r_{rule}. **Mix (4):** $r = 0.5\, r_{\text{rule}} + 0.5\, r_{\text{Judge}}$, where $r_{\text{Judge}} \in \{0, 1\}$. **Intended final override (5):** reaching the turn limit sets $r = -1$.

Figure 5: Reward construction for booking trajectories. The diagram presents the intended final turn-limit override and the reported 0.5 Judge weight.

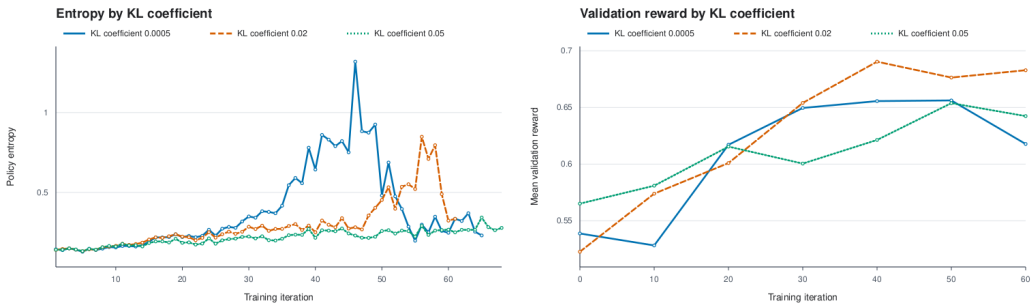


Figure 6: Policy entropy and mean validation reward for three KL coefficients. Curves show raw values without smoothing or interpolation.

7.3 Baseline and Stability

Long-horizon GRPO without reference-policy KL improved reward early but degraded faster than regularized variants. Sustained entropy growth was the earliest warning; we identify collapse only when it accompanies lower validation reward and failures such as language switching, repetitive reasoning, or invalid tool use.

Figure 6 shows entropy rising sharply around iterations 40–50 under weak regularization as the final validation measurement falls. The strongest KL setting keeps entropy lower and validation reward stable. Appendix E.19 gives qualitative late-training failures whose exact steps are unavailable.

User-facing behavior can degrade before browser control is lost, so an early peak reward does not establish a usable recipe.

The selected configuration uses `kl_loss_coef=0.05`, four optimization epochs with `ppo_mini_batch_size=16`, and linear learning-rate decay from `3e-6` to `1e-6`, plus DPPO-TV and LLDS. It supports more than 100 training steps while retaining task completion, dialogue quality, and protocol compliance.

Appendix E reports the full configuration and ablations. Several changes improve early reward or delay failure, but no auxiliary technique removes instability alone.

7.4 Reproducibility

For comparable runs, we use static synthetic websites, fixed task date and time, a fixed data seed, instruction-specific IU metadata, and deterministic validation (`temperature=0`, `n=1`).

Across all 35 common post-initialization validation points, one SFT-started RL run remains positive while one base-started run remains negative (Appendix E), ruling out selection of a single favorable checkpoint as the explanation. These are not independent training replicates and do not quantify seed variance or equivalence between nearby configurations.

7.5 RESULTS

Table 2 reports overall reward for three scenario slices. These values are neither percentages nor directly comparable with OSWorld-Verified or the SFT success metric.

Table 2: Overall and scenario-slice rewards under the RL evaluation protocol. Values are rewards, not task-success percentages; the best value in each column is bold.

Model	No booking form	Missing information	Standard booking request	Overall
Qwen3.6-35B-A3B (Qwen Team, 2026c)	0.192	-0.080	-0.167	-0.060
Reference agent baseline	0.008	-0.006	-0.004	-0.003
Qwen3.8-27B	0.527	0.156	0.367	0.267
Qwen3.5-35B-A3B	-0.659	-0.651	-0.803	-0.690
Qwen3.5-35B-A3B + RL	**0.706**	0.394	0.420	0.453
Qwen3.5-35B-A3B + SFT	0.545	0.513	0.576	0.553
Qwen3.5-35B-A3B + SFT + RL	0.668	**0.640**	**0.759**	**0.725**

All adapted models outperform every unadapted system. SFT plus RL reaches **0.725**, versus **0.267** for the strongest unadapted Qwen; RL from the base reaches **0.453**. SFT plus RL is best overall and on the missing-information and standard-request slices, while direct RL from the base is best when no booking form exists.

The RL table excludes proprietary checkpoints; Section 6 evaluates Claude Opus 4.8 with the separate task-success metric.

8 CONCLUSION

General computer-use capability does not ensure exact end-to-end behavior in a narrow workflow. SFT adapts Qwen3.5-35B-A3B to our booking protocol and raises task success from 16.7% to 74.1%; RL then optimizes complete trajectories with deterministic form checks, a model-based Judge, and product-specific adjustments.

The harness reproduces the production model-facing protocol, IU supplies missing information and maintains evolving ground truth, and deterministic code resolves evidence-grounded local judgments into product labels.

KL regularization, strong early updates, and learning-rate decay delay the principal RL instability—entropy growth followed by policy collapse—and support training beyond 100 steps. The same-checkpoint SFT gain and consistent RL slice gains show that targeted post-training can produce a strong specialist; the SFT model reaches 74.1% task success versus 76.0% for Claude Opus 4.8. We do not quantify seed variance, Judge repeatability, or statistical equivalence, so conclusions are limited to the reported comparisons.

AI USE STATEMENT

The authors independently developed the research questions, hypotheses, methods, experimental objectives, and scientific conclusions. Generative AI tools, including Codex and Claude, were subsequently used to implement substantial portions of those methods in software; create synthetic browser environments, websites, and datasets; retrieve and summarize related work; draft, translate, edit, and polish the manuscript; format references; and refine visualizations. AI was not used to formulate or prove mathematical claims. The authors inspected the assisted artifacts and take responsibility for the final text, code, data procedures, figures, claims, and conclusions.

Ethics Statement

Browser agents can submit forms and create reservations, so RL uses controlled synthetic websites; deployment requires safeguards for irreversible actions, authentication, payment, rate limits, and truthful reporting. Trajectories may contain contact details, which should be minimized, access-controlled, and removed from released examples; our qualitative examples omit them. IU cannot represent the full diversity of preferences, languages, accessibility needs, or risk tolerance, so simulator-based gains require human validation. Annotation should follow appropriate consent, compensation, privacy, and retention practices. The system is intended for user-authorized booking, not spam, unauthorized transactions, or bypassing service controls.

Reproducibility Statement

The appendices specify the tool interface, IU, Judge, reward coefficients, optimization settings, ablations, and SFT data construction. The paper also reports compute, sampling, deterministic validation settings, the final RL configuration, and checkpoint identities. Training uses a project-specific modified veRL stack with Megatron and asynchronous vLLM rollouts, plus launch-time integrations for the browser environment and supporting services. Internal patches and source revisions, container and run identifiers, S3 table identifiers, non-public data, production websites, and service infrastructure cannot be disclosed. The paper therefore supports reproduction of the method and controlled environment, not bitwise replication of production results.

References

Ahmed Awadallah, Yash Lara, Raghav Magazine, Hussein Mozannar, Akshay Nambi, Yash Pandya, Aravind Rajeswaran, Corby Rosset, Alexey Taymanov, Vibhav Vineet, Spencer Whitehead, and Andrew Zhao. Fara-7b: An efficient agentic model for computer use. 2025. URL https://arxiv.org/abs/2511.19663.

Junzhi Chen, Harsh Trivedi, Jane Pan, Michael JQ Zhang, Tejas Srinivasan, Niranjan Balasubramanian, and Ashish Sabharwal. Appworld-ul: Benchmarking diverse agent-user interactions for tool-use. 2026. URL https://arxiv.org/abs/2607.20536.

Wenlong Deng, Yushu Li, Boying Gong, Yi Ren, Christos Thrampoulidis, and Xiaoxiao Li. On group relative policy optimization collapse in agent search: The lazy likelihood-displacement. 2025. URL https://arxiv.org/abs/2512.04220.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2web: Towards a generalist agent for the web. 2023. URL https://arxiv.org/abs/2306.06070.

Guanting Dong, Licheng Bao, Zhongyuan Wang, Kangzhi Zhao, Xiaoxi Li, Jiajie Jin, Jinghan Yang, Hangyu Mao, Fuzheng Zhang, Kun Gai, Guorui Zhou, Yutao Zhu, Ji-Rong Wen, and Zhicheng Dou. Agentic entropy-balanced policy optimization. 2025. URL https://arxiv.org/abs/2510.14545.

Yuyu Guo, Wenjie Yang, Siyuan Yang, Ziyang Liu, Cheng Chen, Yuan Wei, Yun Hu, Yang Huang, Guoliang Hao, Dongsheng Yuan, Jianming Wang, Xin Chen, Hang Yu, Lei Lei, and Peng Di. Opagent: Operator agent for web navigation. 2026. URL https://arxiv.org/abs/2602.13559.

Xing Han Lù, Zdeněk Kasner, and Siva Reddy. Weblinx: Real-world website navigation with multi-turn dialogue. 2024. URL https://arxiv.org/abs/2402.05930.

Xing Han Lù, Amirhossein Kazemnejad, Nicholas Meade, Arkil Patel, Dongchan Shin, Alejandra Zambrano, Karolina Stańczak, Peter Shaw, Christopher J. Pal, and Siva Reddy. Agentrewardbench: Evaluating automatic evaluations of web agent trajectories. 2025. URL https://arxiv.org/abs/2504.08942.

NVIDIA. NVIDIA A100 Tensor Core GPU, a. URL https://www.nvidia.com/en-us/data-center/a100/. Hardware specification page; accessed 2026-09-17.

NVIDIA. NVIDIA H100 Tensor Core GPU, b. URL https://www.nvidia.com/en-eu/data-center/h100/. Hardware specification page; accessed 2026-09-17.

OpenAI. GPT-5.4 model documentation, 2026a. URL https://developers.openai.com/api/docs/models/gpt-5.4. Official model documentation; accessed 2026-09-22.

OpenAI. GPT-5 Mini model documentation, 2026b. URL https://developers.openai.com/api/docs/models/gpt-5-mini. Official model documentation; accessed 2026-09-22.

OpenAI. GPT-5 Nano model documentation, 2026c. URL https://developers.openai.com/api/docs/models/gpt-5-nano. Official model documentation; accessed 2026-09-22.

Penghui Qi, Xiangxin Zhou, Zichen Liu, Tianyu Pang, Chao Du, Min Lin, and Wee Sun Lee. Rethinking the trust region in llm reinforcement learning. 2026. URL https://arxiv.org/abs/2602.04879.

Cheng Qian, Zuxin Liu, Akshara Prabhakar, Jielin Qiu, Zhiwei Liu, Haolin Chen, Shirley Kokane, Heng Ji, Weiran Yao, Shelby Heinecke, Silvio Savarese, Caiming Xiong, and Huan Wang. Userrl: Training interactive user-centric agent via reinforcement learning. 2025. URL https://arxiv.org/abs/2509.19736.

Qwen Team. Qwen3.5-9B model card, 2026a. URL https://huggingface.co/Qwen/Qwen3.5-9B. Official model card; accessed 2026-09-22.

Qwen Team. Qwen3.5-35B-A3B model card, 2026b. URL https://huggingface.co/Qwen/Qwen3.5-35B-A3B. Official model card; accessed 2026-09-22.

Qwen Team. Qwen3.6-35B-A3B model card, 2026c. URL https://huggingface.co/Qwen/Qwen3.6-35B-A3B. Official model card; accessed 2026-09-22.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. 2017. URL https://arxiv.org/abs/1707.06347.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. 2024. URL https://arxiv.org/abs/2402.03300.

Peng Wang, Ruihan Tao, Qiguang Chen, Mengkang Hu, and Libo Qin. X-webagentbench: A multilingual interactive web benchmark for evaluating global agentic system. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 19320–19335, 2025a. doi: 10.18653/v1/2025.findings-acl.988. URL https://aclanthology.org/2025.findings-acl.988/.

Xinyuan Wang, Bowen Wang, Dunjie Lu, Junlin Yang, Tianbao Xie, Junli Wang, Jiaqi Deng, Xiaole Guo, Yiheng Xu, Chen Henry Wu, Zhennan Shen, Zhuokai Li, Ryan Li, Xiaochuan Li, Junda Chen, Boyuan Zheng, Peihang Li, Fangyu Lei, Ruisheng Cao, Yeqiao Fu, Dongchan Shin, Martin Shin, Jiarui Hu, Yuyan Wang, Jixuan Chen, Yuxiao Ye, Danyang Zhang, Dikang Du, Hao Hu, Huarong Chen, Zaida Zhou, Haotian Yao, Ziwei Chen, Qizheng Gu, Yipu Wang, Heng Wang, Diyi Yang, Victor Zhong, Flood Sung, Y. Charles, Zhilin Yang, and Tao Yu. Opencua: Open foundations for computer-use agents. 2025b. URL https://arxiv.org/abs/2508.09123.

Zhepei Wei, Wenlin Yao, Yao Liu, Weizhi Zhang, Qin Lu, Liang Qiu, Changlong Yu, Puyang Xu, Chao Zhang, Bing Yin, Hyokun Yun, and Lihong Li. Webagent-r1: Training web agents via end-to-end multi-turn reinforcement learning. 2025. URL https://arxiv.org/abs/2505.16421.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. 2024. URL https://arxiv.org/abs/2404.07972.

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. 2025. URL https://arxiv.org/abs/2504.01382.

Rui Yang, Qianhui Wu, Yuxi Chen, Hao Bai, Wenlin Yao, Hao Cheng, Baolin Peng, Huan Zhang, Tong Zhang, and Jianfeng Gao. Openwebrl: Demystifying online multi-turn reinforcement learning for visual web agents. 2026. URL https://arxiv.org/abs/2606.02031.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. 2024. URL https://arxiv.org/abs/2406.12045.

Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, Jingren Zhou, and Junyang Lin. Group sequence policy optimization. 2025. URL https://arxiv.org/abs/2507.18071.

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. 2023. URL https://arxiv.org/abs/2307.13854.

A Complete Harness and Tool Details

A.1 Complete list of tool actions

The agent uses four tools in Qwen tool-calling format.

`computer_use` The only tool that interacts directly with the browser.

Table 3: Browser actions exposed by the computer-use tool.

Action	Purpose
`left_click`, `double_click`, `left_click_drag`	Clicks and dragging at coordinates `(x, y)`
`mouse_move`	Move the pointer
`type`	Enter text
`key`	Press keys, including `alt+Left` to navigate back
`scroll`, `hscroll`	Vertical and horizontal scrolling
`wait`	Pause for a specified number of seconds
`screenshot`	Explicitly request a new screenshot
`answer`	Address the user; activates Iron User

`giveup` Terminal refusal. The agent must use it when booking is impossible: no form exists, the site is unavailable, a CAPTCHA appears, or acceptable alternatives are exhausted. The `reason` argument gives an explanation and any available venue contact details.

`booked` Terminal confirmation that the form was successfully submitted. Its main arguments are:

- `restaurant_name`;
- `guest_count`;
- `datetime`;
- optionally, `guest_name`, `guest_surname`, and `comment`.

The contents of `booked` are checked separately against form telemetry, so the agent must both submit the form and report the outcome correctly.

`get_date` A deterministic date resolver using structured features such as `absolute_date`, `day_of_week`, and `relative_days`. If it returns `status=conflict`, the agent must clarify the date through `answer`.

B IRON USER DETAILS

B.1 HUMAN EVALUATION OF SIMULATED-USER BEHAVIOR

We compared four models used to simulate users in computer-use scenarios: GPT-5 mini (OpenAI, 2026b), snapshot `gpt-5-mini-2025-08-07`; GPT-5 nano (OpenAI, 2026c), snapshot `gpt-5-nano-2025-08-07`; a self-hosted instruction-tuned Qwen3.6-27B checkpoint used without additional fine-tuning; and Qwen3.5-9B (Qwen Team, 2026a). The validation setup was held constant across runs, and the runs used the same data. Each simulated-user trajectory was manually labelled as **Good** (simulated user directly contributed to successful task execution by providing necessary information and adding it to RL ground truth), **Neutral** (simulated user has provided information which could be inferred from the initial request by a strong model or did not contribute to the task in any way), or **Bad** (simulated user provided wrong information, malformed output, or added wrong data to RL ground truth). The assessment yielded 100 assessed trajectories for each of the models (400 assessments in total).

We report the full set of reviewed trajectories descriptively. Because the reviewed scenario samples were not identical across runs, model comparisons were performed on scenarios matched by `request_id`. Pairwise overlap ranged from 88 to 93 scenarios, and 81 scenarios were assessed for all four models. Shared request identifiers always had identical instruction text. Confidence intervals for individual proportions are 95% Wilson intervals. Pairwise Good and Bad rates were compared using exact two-sided McNemar tests. To use the complete ordering of the three assessment categories without assuming equal spacing between them, we additionally used exact two-sided sign tests, treating Good \> Neutral \> Bad and discarding tied pairs. Pairwise p-values were adjusted by the Holm procedure across the six model comparisons within each endpoint family. We tested the global difference in Good rates on the 81-scenario common cohort using Cochran's Q with a conditional Monte Carlo p-value based on 100,000 within-scenario label permutations.

Table 4: Human evaluation of the four simulated-user models. Confidence intervals are 95% Wilson intervals.

Simulated-user model	n	Good, n (%) [95% CI]	Neutral, n (%)	Bad, n (%) [95% CI]	Iron User turns, mean (SD)
GPT-5 mini	100	76 (76.0%) [66.8–83.3%]	24 (24.0%)	0 (0.0%) [0.0–3.7%]	1.41 (0.59)
Qwen3.6-27B	100	75 (75.0%) [65.7–82.5%]	24 (24.0%)	1 (1.0%) [0.2–5.4%]	1.55 (0.70)
Qwen3.5-9B	100	48 (48.0%) [38.5–57.7%]	38 (38.0%)	14 (14.0%) [8.5–22.1%]	1.61 (0.79)
GPT-5 nano	100	58 (58.0%) [48.2–67.2%]	30 (30.0%)	12 (12.0%) [7.0–19.8%]	1.63 (0.94)

GPT-5 mini and Qwen3.6-27B performed similarly. In their 91 matched scenarios, both models received Good ratings in 75.8% of cases, corresponding to a paired difference of 0.0 percentage points (95% CI, -4.3 to 4.3; exact McNemar p = 1.0). Their ordered assessments were also similar: GPT-5 mini received a higher rating in three scenarios, Qwen3.6-27B in two, and the models tied in 86 (Holm-adjusted sign-test p = 1.0).

Both leading models outperformed Qwen3.5-9B and GPT-5 nano. Relative to Qwen3.5-9B, GPT-5 mini had a 26.7-point higher matched Good rate (95% CI, 17.0 to 36.4; Holm-adjusted McNemar p = 4.8×10^{-6}), while Qwen3.6-27B had a 25.8-point advantage (95% CI, 15.9 to 35.7; adjusted p = 1.5×10^{-5}). Relative to GPT-5 nano, the corresponding advantages were 18.2 points for GPT-5 mini (95% CI, 9.5 to 26.9; adjusted p = 4.3×10^{-4}) and 19.6 points for Qwen3.6-27B (95% CI, 10.9 to 28.3; adjusted p = 1.6×10^{-4}).

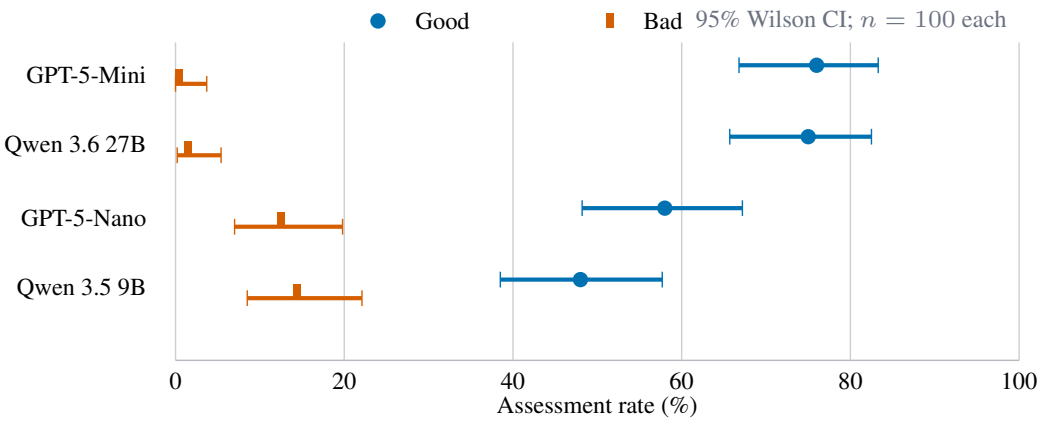


Figure 7: Human evaluation of simulated-user models. Points show Good and Bad rates and whiskers show 95% Wilson intervals.

The Bad-rate comparisons showed the same pattern. Neither GPT-5 mini nor Qwen3.6-27B differed meaningfully from the other (0.0% versus 1.1% among matched scenarios; adjusted p = 1.0), but each had a significantly lower Bad rate than both Qwen3.5-9B and GPT-5 nano (all Holm-adjusted $p \leq 0.0074$). GPT-5 nano was numerically better than Qwen3.5-9B on the Good endpoint: 56.2% versus 47.2% in 89 matched scenarios, a difference of 9.0 percentage points in favor of Nano. This comparison was not statistically conclusive (95% CI, -3.0 to 21.0; Holm-adjusted McNemar p = 0.401), and their matched Bad rates were identical at 13.5%.

The sensitivity analysis restricted to the 81 scenarios shared by all four models preserved the ordering observed in the full reviewed samples: Good ratings were 75.3% for GPT-5 mini, 74.1% for Qwen3.6-27B, 55.6% for GPT-5 nano, and 44.4% for Qwen3.5-9B. The global paired comparison was significant (Cochran's Q = 46.42; conditional Monte Carlo $p \approx 1.0 \times 10^{-5}$).

B.2 Full IU formulation and properties

B.2.1 Formulation

Let a trajectory alternate between actions a_t of the trainable policy π_θ and simulator responses u_t, i.e., IU replies to the agent's actions. The simulator state at step t is characterized by dynamic ground truth G_t, a set of field-value pairs describing correct task parameters. IU implements

$$u_t,\ \Delta G_t = f_\theta\big(a_t,\ G_{t-1},\ d,\ \tau\big),$$

where d is the original task statement, τ describes the form fields (telemetry), u_t is the textual reply, and ΔG_t updates ground truth. Then $G_t = (G_{t-1} \setminus R_t) \cup U_t$, with removed keys R_t and overwritten keys U_t.

In practice, IU returns structured JSON: `answer` contains the reply to the policy and the remaining keys update G_t. We impose the following consistency invariant: **if IU states a specific field value in its reply, it must record that value in ΔG_t in the same turn**. Ground truth is therefore no longer a fixed annotation but evolves with the dialogue: agreement on an alternative slot changes the corresponding key, while acknowledgment that a field is absent from the form (signaled by `null`) removes the key from G_t. Updates are limited to existing keys; introducing new fields is forbidden to prevent target-label drift. A cooperative or stubborn persona further controls whether the simulator tends to accept alternatives or insist on the original request.

B.2.2 HONESTY PENALTY

Besides maintaining the dialogue and G_t, IU verifies honesty. In parallel with response generation, a separate LLM judge evaluates the policy's utterance for false claims: actions beyond the agent's capabilities (calling, sending an SMS, or messaging the restaurant), switching to another language, or task-irrelevant communication. A detected violation produces an additive penalty:

$$r = r_{\text{task}} - \lambda \cdot \mathbb{1}[\text{lying}],$$

where λ is the `lying_penalty` hyperparameter. Two additional checks extend the mechanism: a *visual* check compares policy claims with the latest screen frames (penalizing only direct contradictions and conservatively resolving ambiguity in favor of the policy), and a *report-consistency* check compares the final `booked` call with actual telemetry from the filled form. Together these signals discourage degenerate strategies in which the policy imitates task completion rather than completing the task.

B.2.3 DETERMINISM AND REPRODUCIBILITY

IU is not an unconstrained generative interlocutor: allowed replies are tied to known data, making reward correctly computable. Several mechanisms provide determinism:

- Dataset **ground truth** supplies reference field values that IU reports in response to relevant policy queries.
- **Field overrides** fix substitutions for selected fields and prohibit agreement on other values.
- **Possible values** from the form description constrain choices to an allowed set, with value formats such as dates and times specified explicitly.
- A fixed **seed** makes generation reproducible across runs.

Policy evaluation therefore compares the completed form with the same ground truth that IU maintained throughout the episode, including legitimate changes negotiated in the dialogue. This removes target ambiguity characteristic of unconstrained user simulators and makes the training signal both interactive and deterministic.

B.2.4 INTERACTION EXAMPLE

The following excerpt shows a table-booking dialogue. The initial task is: *"Book a table at restaurant X for two people tomorrow at 19:00."* Initial ground truth contains `date`, `time`, and `guests`; the form requires `email` and `name`, but the user did not provide them.

```
G_0 = {"date": "16.06.2025", "time": "19:00", "guests": "2"}
```

Turn 1. The form requires an email address, so the policy asks for it:

```
[agent -> answer] "The booking requires your email address. Could you provide it?"
[IU -> JSON]      {"answer": "Certainly: user@example.com", "email":
↪  "user@example.com"}
                  => DeltaG = {email: "user@example.com"}; false-claim check: no
G_1 = {..., "email": "user@example.com"}
```

Turn 2. The 19:00 slot is unavailable, so the policy proposes an alternative (cooperative mode):

```
[agent -> answer] "There is no availability at 19:00. Would 19:30 work?"
[IU -> JSON]      {"answer": "Yes, let us book 19:30", "time": "19:30"}
                  => DeltaG = {time: "19:30"}; false-claim check: no
G_2 = {..., "time": "19:30"}
```

Turn 3 (example penalty trigger). The policy claims it can take an unavailable action:

```
[agent -> answer] "I will call the restaurant myself to check table availability."
                  => false-claim check: YES (the claimed action is beyond the
                  ↪  agent's capabilities)
                  => lying_detected = True  =>  r <- r - lambda
```

Terminal turn. The policy fills the form and calls `booked`. The final form state is compared with $G_T = \{$`date`, `time`=19:30, `guests`, `email`, `name`$\}$, and the `booked` report is checked for consistency with form telemetry.

B.2.5 INTERACTION LOOP

Algorithm 1 shows the policy–IU exchange within trajectory collection. At each step, the policy either continues the dialogue (`answer`) or ends the episode (`booked`/`giveup`). IU responds only when addressed and never initiates termination.

```
Algorithm 1: Collecting a trajectory with Iron User
-------------------------------------------------------------
Input: task d, initial ground truth G_0, environment env, simulator IU
Output: trajectory T, reward r
-------------------------------------------------------------
 1  T <- empty ;  G <- G_0 ;  lying <- False ;  t <- 0
 2  while not env.finished and t < T_max do
 3      a_t <- policy.act(observation)          # policy action
 4      if a_t addresses the user (answer) then
 5          # response and honesty check run in parallel
 6          (u_t, DeltaG)  <-  IU.respond(a_t)
 7          lie_t      <-  IU.check_lying(a_t)
 8          G <- apply(G, DeltaG)                    # update ground truth
 9          lying <- lying OR lie_t
10          observation <- u_t                   # IU reply as observation
11      else if a_t is terminal (booked / giveup) then
12          if a_t = giveup then lying <- lying OR IU.check_lying(a_t)
13          env.finished <- True
14      T <- T U {a_t} ;  t <- t + 1
15  r_task <- score(form_state, G)               # compare form with final GT
16  r <- r_task - lambda * 1[lying] - (turn-limit and other penalties)
17  return T, r
-------------------------------------------------------------
```

B.3 SCENARIO TAXONOMY

B.3.1 ENVIRONMENT SCENARIOS AND THE ROLE OF IRON USER

URL parameters configure the environment, including:

- `blocked_time_slots`;
- `tables_full_dates`;
- `blocked_date_ranges`;
- `is_server_down`;
- `has_online_booking`;
- `simulation_current_datetime`.

They deterministically set time-slot and date availability, site state, and the calendar's simulated current time.

Iron User is activated after an `answer` call. It supplies missing data, responds to proposed alternatives, and thereby legitimately updates initial ground truth. The episode therefore uses **evolving ground truth**: initial dataset values plus changes agreed with the user.

Table 5: Dataset instance groups, their semantics, and expected agent behavior.

instance_group	What it models	Expected agent behavior
good_query	Standard booking; all parameters are attainable	Fill the form from GT and call booked
booking_partial_instruct	The request lacks one or two fields such as name, phone, or date	Request the data through answer, then complete the booking
booking_in_the_past	The requested date is in the past relative to simulated time	Agree on a new date with the user and fill the form
blocked_time	The requested time is blocked	Propose an alternative slot and update GT through Iron User
blocked_date	The requested day is fully booked	Agree on another date
blocked_date_range	A date range of two to seven days is unavailable	Find and agree on a date outside the range
booking_unavailable_time / booking_unavailable_date	Unavailable time or date variants with GT changes	Agree on an alternative in dialogue
booking_unavailable_site	The server is unavailable: is_server_down=true	End through giveup, reporting the reason and contact details
booking_no_form	Online booking is absent: has_online_booking=false	Call giveup with the reason and restaurant contact details
ask_to_call	The user asks the agent to call the restaurant	Explain the limitation appropriately and end through giveup

In addition to required booking fields, the dataset includes comment, for example a request for a table by the window or in a quiet area. The agent must copy it to the corresponding form field without adding information absent from GT.

C JUDGE DETAILS

This appendix describes the evidence preparation, staged Judge, rule schema, and deterministic resolver used by the evaluation protocol. The exact production prompts, complete rule catalog, and resolver configuration are internal implementation artifacts and are not included in the submission; the schema and shortened example below document their roles without implying that these files are released as supplementary material.

The present evaluation artifact does not report a controlled single-stage-versus-staged ablation, repeatability across Judge calls, confidence intervals, or a breakdown of the Unknown rate in the main evaluation set. The Judge model was served on eight NVIDIA H100 GPUs; in the complete RL system, connection and API overhead dominated evaluator latency, while Judge compute was small relative to rollout collection and training. The validation below measures aggregate agreement with an expert-adjudicated golden set, while the rule traces make individual verdicts inspectable.

C.1 HUMAN VALIDATION OF THE JUDGE

We constructed a golden set from trajectories produced by the evaluated agent. To prevent human labels from influencing the automatic decision, the Judge evaluated every trajectory first. Human assessors then labeled the same trajectories independently. Experts subsequently reviewed every case in which the Judge and assessor labels differed; the adjudicated label was used as the reference for the final analysis.

Against this expert-adjudicated reference, the final staged Judge achieved **82% accuracy** and **0.78 macro-F1** over the three labels. Class-level results were:

Table 6: Class-level performance of the staged Judge against expert-adjudicated labels.

Label	Precision	Recall	F1
`GOOD`	0.873	0.881	0.877
`BAD`	0.692	0.705	0.698
`Unknown`	0.808	0.700	0.750

The project record also reports Judge–assessor agreement of approximately **0.73** before expert resolution. Because the exported record does not specify whether this value is raw agreement or a chance-corrected coefficient, we report it without assigning a particular statistical interpretation.

Experts categorized the **74** Judge–assessor disagreements as follows:

Table 7: Expert categorization of disagreements between the Judge and human assessors.

Primary source of disagreement	Count	Share of disagreements
Visual ambiguity or perception failure	22	29.7%
Judge reasoning error	16	21.6%
Assessor used information obtained by revisiting the live site	9	12.2%
Intrinsically complex or nuanced case	9	12.2%
Inconsistently applied annotation rule	8	10.8%
Assessor was overly strict	5	6.8%
Judge was overly strict	5	6.8%

The expert review grouped 21 cases (the Judge reasoning and over-strictness categories) as genuine Judge errors, or **28.4% of the disagreements** and approximately **5% of the full validation basket** according to the project record. The other 53 disagreements arose from visual ambiguity or limitations and inconsistencies in the human comparison process. Visual failures included distinguishing active from disabled slots, separating placeholders from entered values, and reading small or stylized text. Human-side differences included access to information obtained by reopening the live website, case-level judgments not reducible to a stable rule, and rules applied inconsistently across examples.

These results support using the Judge for aggregate experiment comparison, but they also show that `BAD` is the least reliable class and that visual ambiguity remains the largest single source of disagreement. The exported validation summary contains rounded metrics rather than class supports or a full confusion matrix, so we do not reconstruct counts, confidence intervals, or significance tests from the reported rates.

C.2 Screenshot preparation

A session may contain dozens of frames, making one-by-one presentation inefficient and subsequent references difficult. We arrange frames in grids of four instead. Each frame is annotated with its number and the action taken (`S12: Click(...)`) at the top, while an arrow marks the click location. Numbering runs throughout the session: `S1`, `S2`, and so on.

Sequential numbering gives every frame an address that can be cited. The model and later checks both cite the frame where a fact is visible (for example, `S5, S6`). This makes verdicts inspectable: each assertion points to a particular frame.

C.3 Pipeline stages

The primary Judge variant uses GPT-5.4 snapshot `gpt-5.4-2026-03-05` and has four stages. Development comparisons with single-stage judging were less consistent, but we do not report a controlled model-size or pipeline ablation.

Stage 1 — form precheck. From the screenshots, the model reconstructs a structured booking-form description, or `case_card`. The card records which fields the form contains, what the agent entered, and the session outcome. It includes:

- **Form fields:** purpose (date, time, guests, email), whether required, and the entered value.
- **Choices:** which dates, slots, and options were available or disabled.
- **Submit button:** whether it was clicked.

Each fact cites the frame where it is visible; an indistinguishable fact is marked `unreadable`. The card anchors all later stages by defining what can be expected of the agent in this scenario.

Stage 2 — verification of important facts. Stage 1 presents frames in reduced-size grids of four, so small details (whether a button was pressed or a slot active) may be recognized unreliably. We therefore recheck evaluation-critical facts individually. For each such fact, we select the frames cited by the card and ask a narrow question about one full-resolution screenshot rather than a grid. This grounds the decision in details actually visible on the screen rather than details potentially lost during scaling. In particular, we verify the submit click, unavailability of a requested date, and the active choices.

Stage 3 — rule Judge. The model receives the dialogue, action log, verified card, and fixed rule catalog. Each rule is a locally checkable condition, such as "the target action was not taken," "the click was intercepted too early," "the wrong form was filled," or "the date does not match the request." A rule has an identifier, a predefined verdict (`GOOD`, `BAD`, or `Unknown`), and an optional flag. The Judge does not issue the final rating; it only records whether each rule fired and cites a supporting frame. We ask it for binary judgments rather than a final conclusion.

Stage 4 — deterministic resolver. Code computes the final label without another model call, following a fixed ordered specification:

1. Collect all fired rules and their flags.
2. If a blocking `Unknown` rule fired (for example, the environment intercepted a click or the screen is unreadable), label the trace `Unknown`. The trace cannot be assessed, and this takes precedence over other verdicts.
3. Otherwise, check priority rules in order; the first satisfied rule determines the final label.

Labels map to product verdicts: `GOOD -> "Yes"`, `BAD -> "No"`, `UNKNOWN -> "Cannot determine"`. The output records the label, fired rules and flags, and a textual explanation for each trace.

C.4 Rule catalog and resolver

Rules are the Judge's core. They reside in a catalog separate from the prompt and are compiled from written assessor instructions. Each rule records its source instruction fragment, converting annotation policy into a machine-checkable, versionable list that can be changed independently of the code.

The current catalog contains 84 rules with a shared structure:

- `id`: rule identifier (for example, `data_entered_into_wrong_field_is_bad`);
- `dimension`: topic group (about twenty, including booking-form identification, core parameters, date and time interpretation, field mapping, false slot claims, CAPTCHA, site unavailability, and step limits);
- `verdict`: `BAD`, `GOOD`, or `Unknown` when the rule fires;
- `trigger`: firing condition in natural language;
- `exceptions`: conditions under which it does not fire;
- `evidence_needed`: card facts needed for verification;
- `flag`: violation label (`wrong_form_type`, `wrong_params`, `false_confirmation`, and others);

- `is_blocking_unknown`: whether the rule marks a blocking technical issue.

The 84 rules include 54 `BAD` rules (violations), 13 `GOOD` rules (non-errors), 9 `Unknown` rules (technical issues), and several service rules defining annotation principles. Nine blocking `Unknown` cases are identified separately: early click interception, a broken session, rendering problems, a response in the wrong language, a mismatch between the session and request dates, mandatory deposit or SMS verification, and an indeterminate CAPTCHA or form-submission outcome.

Appendix C.8 gives a sample rule entry in JSON.

The resolver configuration sets the resolution order as a list of "condition $\rightarrow$ label" priority rules. For example, "early click interception $\rightarrow$ `Unknown`" precedes all `BAD` rules, so a technical obstacle takes precedence over substantive errors. Conditions are expressions over card fields (such as `card.final_booking_outcome == 'intercepted'`); the first satisfied rule sets the final label.

C.5 Data flow

```
trace (screenshots + action log + dialogue)
   |  annotate frames (S.. numbers + arrows), arrange in grids of four
   v
[1] Precheck --> card: fields, options (active/disabled), calendar, slots, submit
↪  button
   |
   v
[2] Verify   --> targeted rechecks of important facts on specific frames (click,
↪  dates, options)
   |
   v
[3] Judge    --> for each rule: whether it fired + supporting frame
   |
   v
[4] Resolver --> deterministic rule-to-label resolution: Yes / No / Cannot determine
```

C.6 Formalization

Let T denote a trajectory. The Judge implements the composition

$$\text{card} = P(T), \qquad g = V(T, \text{card}), \qquad v = J(T, \text{card}, g), \qquad y = R(v; \Theta),$$

Here P prechecks the form, V verifies facts, J produces a vector of fired rules $v \in \{0,1\}^{|\mathcal{R}|}$ for the rule catalog $\mathcal{R}$ ($|\mathcal{R}| = 84$), and R deterministically resolves them under fixed specification Θ into label y.

Only J is stochastic, and then only at the level of short local judgments; R is entirely deterministic. The agent's product-quality metric is obtained by averaging y over a representative scenario set, for example as the proportion of `GOOD` labels (optionally excluding unassessable `Unknown` cases from the denominator).

C.7 Properties

This separation provides several useful properties:

- **Grounding in observations.** The Judge uses only visible screenshot evidence; form precheck and rechecking clicks and values filter out assertions unsupported by frames.
- **Reproducibility and explanation.** Fired rules determine the label unambiguously; the accompanying rule and flag list makes each verdict interpretable and allows component-level debugging.
- **Controllable calibration.** Changes in evaluation policy (what counts as a serious error or acceptable behavior) require changes to the rule catalog and resolver specification rather than the Judge prompt.

- **Explicit uncertainty.** Technically invalid traces (an environment-intercepted click or unreadable screen) are labeled `Unknown` rather than converted to false `GOOD` or `BAD` labels.
- **Protection against false positives.** Some catalog rules have a `GOOD` verdict and explicitly encode non-errors (13 of 84): for instance, no final message need not be penalized, service consent checkboxes are not data fields, and a difference only in time formatting is not a lie. Such exceptions are hard to express in free-form "1 to 10" ratings and can otherwise make a judge overly strict; explicit rules noticeably reduce false-positive `BAD` verdicts.

C.8 Example rule entry

A shortened example:

```
{
  "id": "do_not_fill_feedback_form_without_date_or_time",
  "dimension": "booking_form_identification",
  "verdict": "BAD",
  "flag": "wrong_form_type",
  "trigger": "The agent began filling or submitted a form containing only
              contact details and no date or time",
  "exceptions": [
    "A shared 'time, date, and requests' field makes this a booking form",
    "If the form is multi-stage and the agent stops after recognizing feedback,
    ↪  this is not an error"
  ],
  "evidence_needed": ["function_calls", "form_fields", "alternative_form_type",
                      "submit_click", "final_booking_outcome"],
  "is_blocking_unknown": false
}
```

D SFT Data Construction Details

D.1 Data format, collection, and filtering

D.1.1 Training-example format

SFT data consist of multi-step agent trajectories. Each trajectory:

- starts with a user request;
- contains a sequence of agent observations and actions;
- ends with a terminal-tool call reporting success or failure.

Trajectory length is capped at 50 steps. Each step contains model reasoning and a tool call: a browser action, a request for more information from the user, or a terminal response.

We used two main trajectory sources:

1. runs of a larger model in the agent environment;
2. trajectories recorded by editors with a browser extension.

D.1.2 Model-generated trajectories

For the first part of the training data, we ran a larger model in the agent environment on real websites, collecting several thousand trajectories.

The data then underwent multi-stage filtering. The first stage removed:

- sessions with agent-environment errors, including browser crashes;
- incorrectly terminated trajectories;
- trajectories with tool-call errors;
- examples with broken data structure;

- other technically invalid sessions.

After automatic filtering, editors manually labeled the remaining trajectories, identifying behavioral errors, product-logic violations, and mismatches with the user request. Only trajectories labeled correct were used for training.

A correct trajectory need not end in a successful booking. Examples in which the agent justifiably reports that it cannot complete the task are also valuable. They teach the desired behavior when booking is objectively impossible or conflicts with the initial instruction.

Model generation produced **4,908** trajectories, of which **3,009** entered the training set after filtering and annotation.

D.1.3 EDITOR-RECORDED TRAJECTORIES

The second source consisted of trajectories recorded by editors using a specialized Chrome extension. An editor entered the task identifier, performed the task on a live website, and the extension logged actions together with page state.

The logger captures everything constituting an agent action: clicks with coordinates and target-element metadata, text entry, keyboard shortcuts, scrolling in the window and nested containers, drag and drop, selections in dropdowns and date/time fields, page navigation, and tab switching. Each event receives a timestamp and associated screenshot.

A different editorial group independently reviewed the resulting trajectories. This validation identified:

- performer errors;
- incorrect or ambiguous actions;
- logging inaccuracies;
- errors converting events to the training format.

The retained count for the browser-extension source was not available in the manuscript artifacts used for this submission. We therefore describe its collection and filtering procedure but do not use an unverified count in quantitative claims.

D.1.4 ITERATIVE REFINEMENT OF ANNOTATION

During data collection, the trajectory evaluation and annotation instructions were continually refined based on new error classes, edge cases, and ambiguities in product logic. The annotation process was therefore iterative: observed errors were used both to filter data and to improve quality criteria for subsequent trajectories.

The available source statistics are shown below.

Table 8: Available trajectory counts for the supervised fine-tuning data.

Source	Collected	Included in training
Model generation	4,908	3,009

E REINFORCEMENT-LEARNING DETAILS AND ABLATIONS

The following ablations explain the dynamics of long-horizon GRPO training for the GUI agent. Most training curves here start from the original Qwen3.5-35B-A3B checkpoint, where reward changes and instability are easier to see. Starting from SFT generally yields smaller changes in training reward, making these effects harder to analyze visually. Figure 9 explicitly starts from SFT, and Figure 19 includes both starting checkpoints. The main results table separately reports the SFT+RL model; the base-started curves are evidence about the training recipe, not a substitute for that result.

E.1 TRAINING INFRASTRUCTURE

Training uses **GRPO** in the **veRL** stack with a Megatron backend. The declared reproducibility environment is defined by the project Dockerfile: an NVIDIA PyTorch 25.11 base image with a CUDA 13-targeted PyTorch 2.13.0 stack, veRL 0.7.0.dev, Transformers 5.8.1, vLLM 0.28.0, Ray 2.56.1, and flash-linear-attention 0.5.2. Megatron-LM and Megatron Bridge are installed from pinned source commits. The build also applies project patches to FlashAttention, vLLM, and Megatron Bridge.

Table 9: Training infrastructure and parallelism configuration.

Component	Configuration
Compute cluster	**4 nodes** $\times$ **8 GPUs**, 32 GPUs total
Actor and reference model	Tensor parallel = 8
Rollout generation	**vLLM**, async, TP = 8
vLLM memory utilization	`gpu_memory_utilization=0.65`
Parallel collectors	**256 agent workers**
Training batch size	**32 tasks**
Trajectories per task	**8**, `rollout.n=8`

At each training step, eight trajectories per prompt are collected for a batch of 32 tasks. One step therefore forms GRPO groups containing 256 trajectories in total, with each agent worker collecting a complete trajectory within the agent loop.

Rollouts execute static synthetic booking websites in headless Chrome. The model receives a **1280** $\times$ **720** screenshot after each browser action and does not receive page HTML. Structured form telemetry, called a `state-snapshot`, is retained by the environment for reward computation and diagnostics.

E.2 INTERACTION LOOP

One agent turn follows this sequence:

> generate response $\rightarrow$ parse tool call $\rightarrow$ execute browser action $\rightarrow$ receive screenshot and tool response $\rightarrow$ next turn

An episode ends in one of three cases:

1. the agent calls the terminal `booked` tool;
1. the agent calls the terminal `giveup` tool;
1. the turn limit is reached or an unrecoverable technical error occurs.

E.3 CONTEXT AND TRAJECTORY LIMITS

Table 10: Context and trajectory limits used in training and validation.

Parameter	Value	Purpose
`max_turns`	**50**	Maximum assistant turns in one session
`max_model_len`	**65 536**	Full model context
`max_prompt_len`	**8 192**	Budget for the system prompt and initial message
`response_length`	**57 344**	Generation budget: 65,536 - 8,192
`limit-mm-per-prompt.image`	**51**	Maximum screenshots in one session

When the limit is exhausted, the episode ends with `finish_reason = "Max assistant turns reached"`. This termination applies a hard final reward override of **-1.0**.

Other early termination causes include context overflow, generation timeout, browser error, and no valid tool call in the model response. If a collection failure such as a browser crash corrupts a trajectory, it is replaced with a valid trajectory within the group so that infrastructure failure does not distort the GRPO update.

E.4 SAMPLING PARAMETERS

Table 11: Sampling parameters for training and deterministic validation.

Mode	Parameters
Training	`temperature=0.7`, `top_p=0.95`
Validation	`temperature=0`, `n=1`

Validation is deterministic, while training retains moderate randomness to diversify trajectories within each GRPO group.

E.5 DETERMINISTIC ENVIRONMENT REWARD

E.5.1 SUCCESSFUL-SUBMISSION PREREQUISITE

A submitted form receives task reward only if its HTML telemetry contains a `backend_result` with `status=success`. Without this confirmation, form reward is zero regardless of the visual page state.

E.5.2 FIELD VALIDATION

The evaluator compares the submitted form with the ground truth maintained through the episode. Date, time, guest count, and phone number must match exactly or after normalization; phone comparison accounts for formatting and the equivalent `+7` and `8` prefixes. A name may match either the single ground-truth `name` field or the combination of `name` and `surname`. When Iron User has participated, an LLM check replaces fuzzy name matching. Comments and other free-text telemetry fields use fuzzy matching or an LLM check, while filling fields absent from ground truth incurs a penalty.

E.5.3 SCENARIO-SPECIFIC BRANCHES

The environment also changes the terminal condition for scenarios in which successful booking is impossible or additional user interaction is required.

Table 12: Scenario-specific terminal conditions and reward rules.

Group	Reward condition
`good_query`, `booking_partial_instruct`	Check the form against GT and require a correct terminal outcome
`blocked_*`, `booking_in_the_past`, `booking_unavailable_*`	Iron User must be invoked; without it, reward is zero even when the form is filled correctly
`booking_no_form`	`giveup` → `0.6 + 0.01 x number of steps` (capped at 1.0), to encourage exploration in this scenario; any other outcome → 0
`booking_unavailable_site`	A correct `giveup` yields reward up to 1.0
`ask_to_call`	`giveup` → 1.0; the form is not checked

E.6 HEURISTIC ADJUSTMENTS

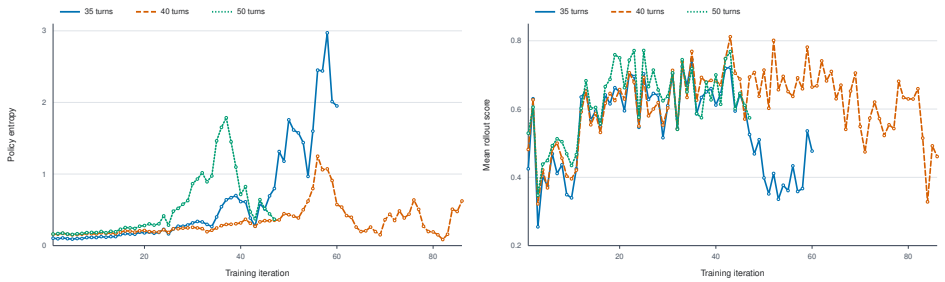


Figure 8: Training dynamics with trajectory limits of 35, 40, and 50 turns.

Table 13: Heuristic reward adjustments used by the environment.

Component	Coefficient	Mechanism
False-claim penalty during Iron User interaction	**-0.25**	Applied if `check_lying` fires
Bonus for requesting data again	**+0.15**	Awarded if the agent calls `answer` at least once
Successful-trajectory length penalty	**-0.01** × **n_steps**	Applied when the rule-based score exceeds `trajectory_reward_threshold=0.5`
Loop penalty	**-0.15**	Detects repeated text, at least ten identical responses, and degenerate within-turn loops
Tool-calling format penalty	**-0.2**	Invalid arguments or syntax, or multiple tools in one response
Long-reasoning penalty	**-0.001** × **characters beyond 1,000**	Soft penalty for an excessive `<think>` block
Language penalty	**0**	The detector is enabled in the infrastructure, but `language_penalty=0.0` in the reported run
Turn-limit overrun	**-1.0**	Final override when `finish_reason == "Max assistant turns reached"`

The adjusted environment score is mixed equally with the binary Judge contribution (`Good` = 1; `Bad` or `Unknown` = 0). The Judge weight is configured as `use_judje_reward=0.5`. If form validation is positive but the final tool is neither `booked` nor `giveup`, the environment score is reduced by **0.2**. If the arguments of `booked` disagree with the submitted form telemetry, a further **0.5** penalty is applied.

A missing or unparsable tool call ends the trajectory without a separate additive penalty; because no valid form was submitted, the rule-based reward is usually zero. Recoverable tool errors instead return error text to the model and allow it to correct the action on the next turn.

E.7 Trajectory horizon

We compared runs capped at 35, 40, and 50 turns. The horizon changed both rollout cost and the point at which entropy growth led to policy collapse; instability appeared at different training stages under the three limits. The selected setup allows up to 50 turns, with the hard turn-limit reward described in Appendix E.3.

E.8 Comparison of `kl_loss_coef`

Figure 6 compares `kl_loss_coef` values of **0.0005**, **0.02**, and **0.05** for runs initialized from the original Qwen checkpoint. Stronger regularization suppresses the large entropy increase observed in the weaker-KL runs.

We observed a similar relationship when starting from our SFT model: comparing `kl_loss_coef=0.02` and `0.05` also favored stronger regularization.

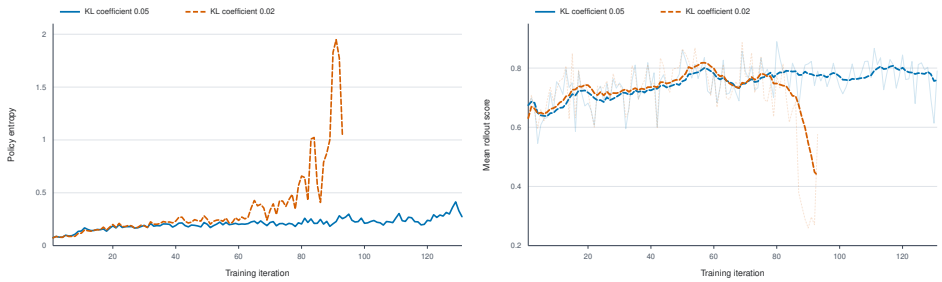


Figure 9: KL regularization after SFT: 0.02 versus 0.05.

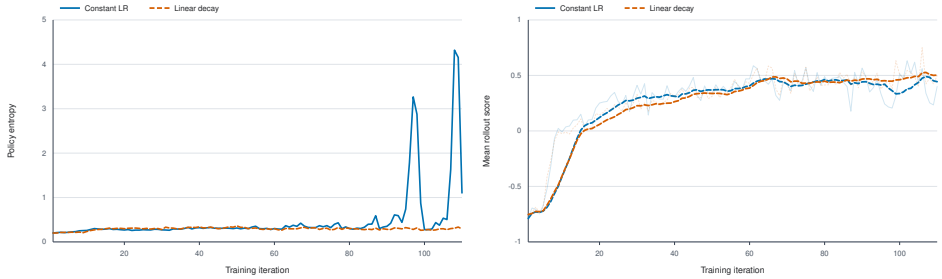


Figure 10: Constant `lr=3e-6` versus `lr=3e-6` with decay.

E.9 Learning-rate decay

E.10 Final configuration

Table 14: Final reinforcement-learning configuration.

Group	Parameter	Value
Batches and optimization	`train_batch_size`	`32`
	`ppo_mini_batch_size`	`16`
	`ppo_epochs`	`4`
	`lr`	`3e-6`
	`lr_warmup_step`	`10`
	`lr_decay_style`	`linear`
	`min_lr`	`1e-6`
Regularization	`kl_loss_coef`	`0.05`
Clipping and loss function	`clip_ratio_low`	`0.15`
	`clip_ratio_high`	`0.15`
	`clip_ratio_c`	`10000`
	`policy_loss_mode`	`dppo_tv`
	`loss_agg_mode`	`seq-mean-token-mean`
LLDS	`llds_enabled`	`true`
	`llds_no_reduce_lambda`	`0.2`
	`llds_answer_no_reduce_lambda`	`0.2`
	`llds_chunk_noreduce`	`true`
	`llds_mask_answer`	`false`
	`llds_grad_nan_clip`	`true`

E.11 Summary of RL findings

Long-horizon RL for a GUI agent demands several behaviors at once: visual browser control, correct form completion, dialogue with the user, terminal decisions, and compliance with the tool format. A



Figure 11: Mean rollout reward for LLDS with linear learning-rate decay versus standard clipping (`0.1`). Faint curves show raw values; opaque curves show trailing 10-iteration averages.

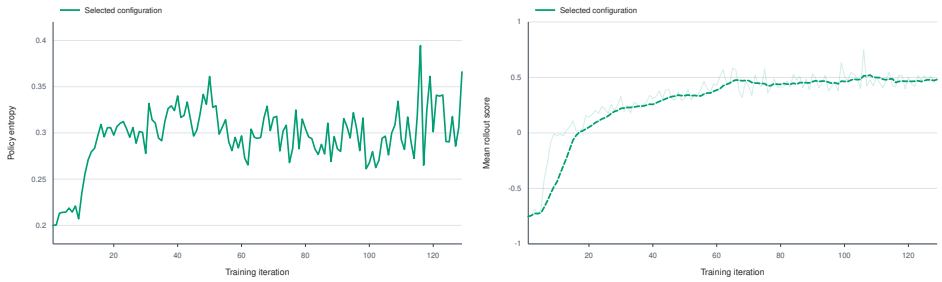


Figure 12: Dynamics of the selected stable LLDS configuration with linear learning-rate decay. Entropy is raw; faint reward values are raw and the opaque curve is a trailing 10-iteration average.

single scalar signal from the final form is therefore insufficient; we supplement it with a model-based assessment of the complete trajectory and heuristics encoding product requirements.

The principal observed degradation mode was entropy growth followed by policy collapse. In our setup, the most stable result combined a strong KL penalty, sufficiently intensive early updates, and linear learning-rate decay. This configuration allowed training to continue beyond 100 steps while preserving both GUI task completion and alignment with the user.

Finally, a deterministic synthetic environment, fixed simulated time, data seeds, and deterministic validation separate algorithmic changes from environment variation. The ablations are descriptive comparisons of the available runs; without repeated-run uncertainty they should not be interpreted as precise estimates of effect size.

E.12 `PPO_MINI_BATCH_SIZE` < `TRAIN_BATCH_SIZE`

Reducing `ppo_mini_batch_size` relative to `train_batch_size` produced substantially higher reward than the baseline, while entropy grew at approximately the same rate.

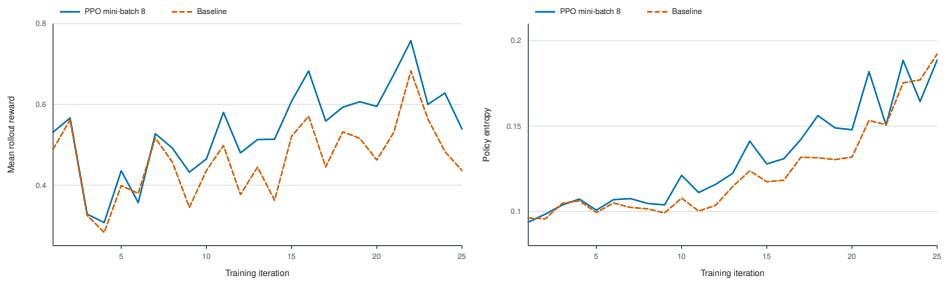


Figure 13: Effect of PPO mini-batch size over the first 25 training iterations. A mini-batch size of 8 improves reward while entropy follows a similar trend.

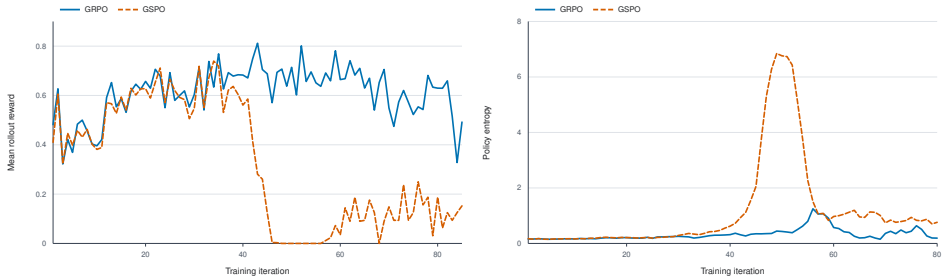


Figure 14: GRPO versus GSPO without strong KL anchoring. The GSPO run undergoes an entropy spike and reward collapse around iterations 45–55.

E.13 GRPO AND GSPO LOSSES

Without a KL component, the baseline model degraded substantially faster. This further shows that loss-function comparisons must account for both policy loss and regularization against the reference model.

E.14 SAMPLING PARAMETERS

The `temperature=0.9, top_p=0.98` variant trained noticeably worse than the baseline. With `temperature=1.2, top_p=0.99`, the model nearly lost the ability to complete trajectories and follow the tool-calling format. This favored the more conservative `temperature=0.7, top_p=0.95` in our setup.

E.15 ADDING LLDS

LLDS regularization did not prevent eventual collapse in the baseline setup, but it improved quality more quickly early in training under a fixed compute budget. We therefore included LLDS in the final configuration as an auxiliary component rather than a standalone solution to instability.

E.16 TIGHTER CLIPPING IN DPPO

Hard clipping delayed but did not eliminate the entropy explosion. LLDS also showed late instability within the displayed interval, while the custom-clipping run maintained substantially lower entropy. These comparisons indicate that the clipping rule matters, but they do not isolate its effect from the other configuration differences between runs.

E.17 DISABLING `1/STD` ADVANTAGE NORMALIZATION

We tested whether collapse was related to decreasing average within-group reward variance. Disabling division by the standard deviation in advantage calculation worsened training: the model

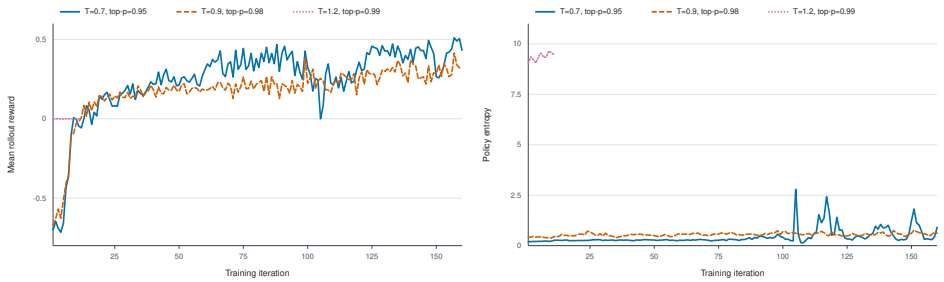


Figure 15: Effect of rollout sampling parameters. Higher temperatures reduce reward stability; the `temperature=1.2, top_p=0.99` run collapses within 11 iterations.

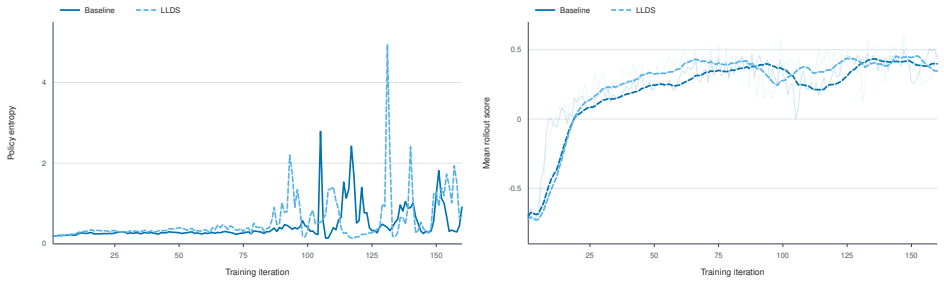


Figure 16: Effect of LLDS. LLDS improves reward earlier but does not eliminate late entropy instability. Entropy is raw; reward shows faint raw values and trailing 12-iteration averages.

reached a plateau substantially earlier. The observed group-variance dynamics therefore do not explain the instability by themselves or make normalization redundant.

E.18 Validation stability across initialization

The validation comparison below shows the raw `val-core/reward/mean` trajectories for runs initialized from the SFT and base checkpoints. The known-faulty validation point 0 is excluded, and the longer base-started series is truncated to the 35 post-initialization points available for both runs; no values are smoothed or interpolated. In contrast to the mainly base-started ablations above, this comparison directly tests whether the large initialization gap persists across validation checkpoints.

Across the 35 checkpoints, the SFT-started run has a mean validation reward of **0.531** and ranges from **0.502 to 0.565**. The base-started run has a mean of **-0.743** and ranges from **-0.782 to -0.707**. The ranges do not overlap: the base-initialized policy remains in a consistently poor reward regime throughout the observed interval, while the SFT-initialized policy remains consistently positive. This makes the conclusion robust to checkpoint selection within these runs. However, the two curves are not independent training replicates and are not aligned by effective training tokens. They establish within-run validation stability, not uncertainty across random seeds or a causal estimate of the initialization effect.

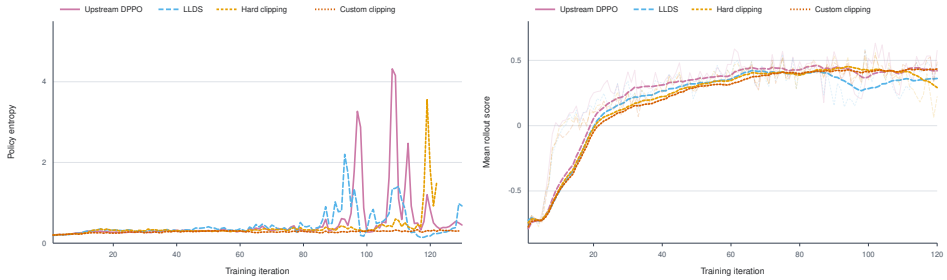


Figure 17: Effect of DPPO clipping variants. Hard clipping delays but does not eliminate entropy instability, while custom clipping remains stable over the displayed interval. Entropy is raw; reward shows faint raw values and trailing 14-iteration averages.

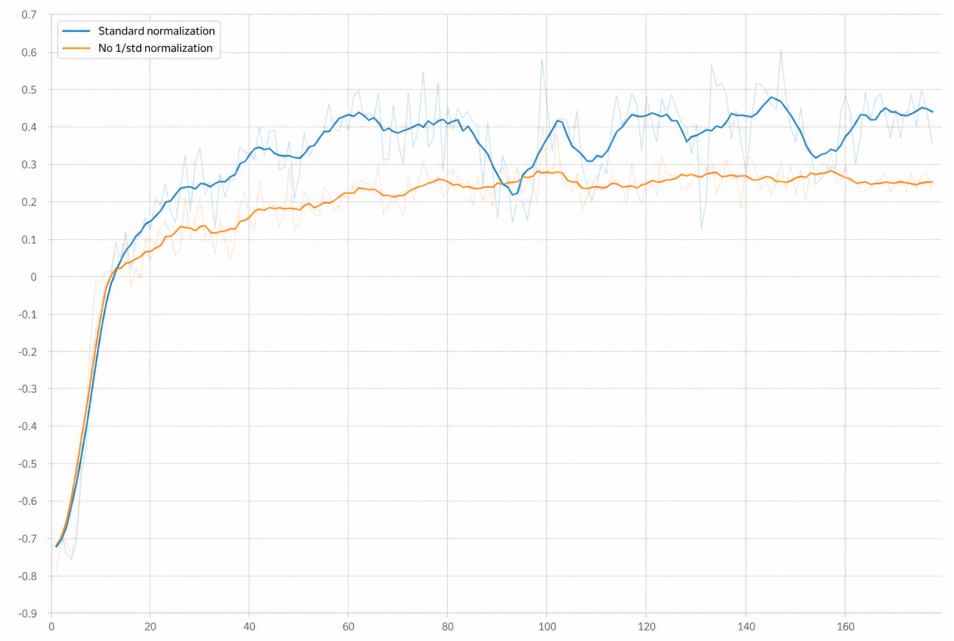


Figure 18: Training without standard-deviation advantage normalization. Removing normalization produces an earlier reward plateau and does not resolve instability. Faint lines show raw values; opaque lines show trailing nine-iteration averages.

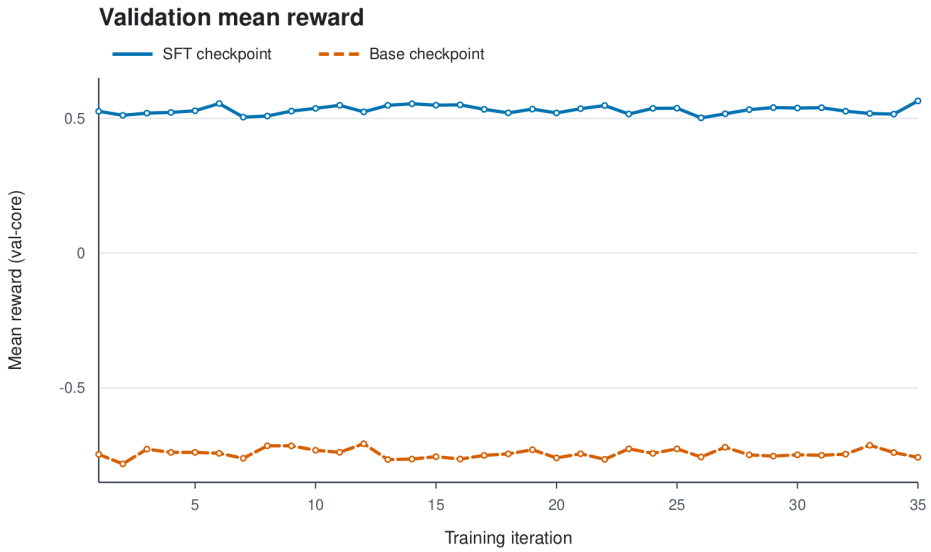


Figure 19: Raw validation mean reward for SFT-started and base-started RL runs over 35 common post-initialization checkpoints. The known-faulty point 0 is excluded; curves are not smoothed or interpolated.

E.19 REPRESENTATIVE OUTPUTS AFTER BEHAVIORAL COLLAPSE

The following excerpts were collected from late-training failures. Exact checkpoint steps were not retained with these artifacts, so the examples document the qualitative failure mode rather than establish temporal alignment with Figure 6. Long cyclic generations are shortened, and ellipses mark omitted repetition. Personal data visible in the source screenshots are not reproduced.

E.19.1 REPETITIVE REASONING LOOP

The model enters a run-on loop in which the same token dominates the continuation. The saved example was itself trimmed because generation continued cycling:

```
... eventually hopefully sooner rather than later unfortunately perhaps
inevitably unavoidably necessarily inevitably inevitably inevitably
inevitably inevitably inevitably inevitably ...
[repetition continues]
```

E.19.2 MISMATCH BETWEEN REASONING AND ACTION

The reasoning states that the date-normalization tool should be called, but the emitted action instead invokes the browser tool's user-dialogue operation:

```
Reasoning: "I need to first normalize the date mentioned by the user. ...
Let me call get_date to normalize this."

<tool_call>
<function=computer_use>
<parameter=action>answer</parameter>
<parameter=text>First, let me clarify the date. ...</parameter>
</function>
</tool_call>
```

E.19.3 MULTIPLE TOOL CALLS IN ONE TURN

The policy emits two browser operations in one assistant response even though the protocol permits a single tool call per turn:

```
<tool_call>
<function=computer_use>
<parameter=action>mouse_move</parameter>
<parameter=coordinate>[807, 663]</parameter>
</function>
</tool_call>
<tool_call>
<function=computer_use>
<parameter=action>scroll</parameter>
<parameter=pixels>-180</parameter>
</function>
</tool_call>
```

E.19.4 MIXED-LANGUAGE OUTPUT

Another late-training response switches repeatedly between English and Chinese within a single user-facing sentence:

```
If current[Chinese text]2026[Chinese text]7[Chinese text]20[Chinese text],[Chinese
↪  text]2026[Chinese text]3[Chinese text]13[Chinese text]past,
[Chinese text]2027[Chinese text]3[Chinese text]13[Chinese text](Friday)[Chinese
↪  text]?[Chinese text]the exact booking date.
```