# ASYNCWEBRL: EFFICIENT ASYNCHRONOUS REINFORCEMENT LEARNING FOR MULTI-STEP VISUAL WEB AGENTS

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Training vision-language web agents with multi-step RL is compute-intensive, with two dominant forms of inefficiency: idle GPUs in synchronous RL, and trajectories that use more steps and tokens than necessary. We present **AsyncWebRL**, which addresses both. On the system side, an asynchronous design overlaps rollout, gradient update, and policy refresh across iterations, paired with two web-agent-specific adaptations, namely an everlasting rollout pool and lightweight screenshot handling, that together deliver up to a $\mathbf{2.9}\times$ end-to-end training-throughput speedup over the previously fastest open synchronous pipeline (WebGym). On the algorithmic side, we identify the per-trajectory normalizer $1/|\tau_i|$ in multi-step GRPO as the root cause of trajectory-level and token-level inefficiency: because failures are systematically longer than successes, it down-weights the negative gradient on failed tokens, so the policy keeps producing verbose memory schemas. Replacing $1/|\tau_i|$ with a constant $1/k$ breaks this coupling, contracting trajectories while preserving aggregate success. Together, these contributions set a new open-source state of the art on the WebGym out-of-distribution test split ($+\mathbf{5.8}\%$ relative over the $42.9\%$ prior best), with the largest gains on the harder slices ($+\mathbf{42}\%$ relative on Medium, $+\mathbf{48}\%$ relative on Hard).

## 1 INTRODUCTION

Vision-language web agents trained with multi-step reinforcement learning have rapidly become a leading recipe for autonomous browsers that complete real-world tasks across the long tail of the web (Bai et al., 2026; Wang et al., 2025; Wei et al., 2025; Yang et al., 2026b), part of a broader shift from single-turn preference tuning to long-horizon, environment-grounded agent training. The defining feature of this regime is compute: each training run consumes hundreds of GPU-hours against hundreds of concurrent browser sessions, and progress is largely a function of how many trajectories the trainer can profitably consume per wall-clock hour. Against this backdrop, any compute inefficiency translates directly into a lower-performing agent at fixed budget. This paper targets both forms of inefficiency within a single framework, combining (1) a fully asynchronous system design tailored to per-step visual rollouts with (2) a one-line algorithmic fix that contracts trajectory and token usage without harming ending success.

The natural systems solution for GPU idleness is *async RL*, where rollout, gradient update, and policy refresh proceed concurrently across iteration boundaries (Mnih et al., 2016; Espeholt et al., 2018; Hilton et al., 2021). On the surface this appears to be a solved problem: single-step LLM-RL is already async, and WebGym (Bai et al., 2026) already makes within-iteration rollouts async. Combining the three properties *visual, multi-step, and fully async*, however, requires a coordinated design that none of the existing open-source systems carry together (Section 2). The core difficulty has two parts. First, the natural payload of a visual multi-step rollout, tens of high-resolution screenshots per trajectory shared across workers every step, swamps the standard inter-worker data store and pushes the framework into a slow disk-spill path that erases any async benefit. Second, an iteration-synchronous rollout pool of the kind WebGym uses pays a warm-up cost on hundreds of browser sessions every round.



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.

We address both with two web-agent-specific designs: lightweight screenshot handling and an everlasting rollout pool. The former keeps image tensors out of the shared data store entirely, routing only lightweight references between rollout workers and the trainer. The latter keeps workers continuously alive across iteration boundaries so episodes hand off immediately and parameter updates of $\pi_\theta$ never stall collection (Section 3.1). Closing the resulting off-policy gap also requires an algorithmic correction. We adopt the decoupled-PPO factorization of Hilton et al. (2021), splitting the standard $\pi_\theta/\pi_{\text{behave}}$ ratio into a rollout-staleness term and a current-update term and centering PPO-style clipping (Schulman et al., 2017) on a proximal policy $\pi_{\text{prox}} \approx \pi_\theta$. As we will show in Section 3.2 and Section 4.3, this roughly halves the clip-trigger rate and substantially speeds reward improvement. The resulting system is, to our knowledge, the fastest reported open multi-step RL framework for visual web agents, at a $\mathbf{2.4}$ to $\mathbf{2.9}\times$ end-to-end speedup over prior open rollout pools (Bai et al., 2026).

Beyond this form of off-policy correction, we identify that per-trajectory step-number based normalization[1], i.e., $\mathbf{1/|\tau_i|}$ in multi-step GRPO (Shao et al., 2024) as the root cause of trajectory- and token-level inefficiency, and replace it with a constant $\mathbf{1/k}$ (Section 5). The intuition follows Dr. GRPO (Liu et al., 2025b) but at a step-level granularity rather than token granularity: in our setting failures average $12.5$ steps against $5.1$ for successes, so $1/|\tau_i|$ attenuates the gradient on failed tokens by $\approx 2.4\times$, and the policy responds by producing verbose memory schemas. Combined with the off-policy correction, this sets a new state of the art on the WebGym OOD test split (Section 4.2).

**Our contributions. (1) System.** The first open multi-step RL framework for visual web agents that is fully async end-to-end, via an everlasting rollout pool, lightweight screenshot handling, and a decoupled importance-sampling ratio (Section 3.1); this delivers a $2.4$ to $2.9\times$ end-to-end speedup over the previously fastest open sync pipeline and a new open-source state of the art on the WebGym OOD test split ($+5.8\%$ relative over the $42.9\%$ prior best). **(2) Algorithm.** A diagnosis of the per-trajectory step-number normalizer $1/|\tau_i|$ in multi-step GRPO as the root cause of trajectory- and token-level inefficiency, traced through length-coupled memory drift (Section 5), and a one-line fix replacing $1/|\tau_i|$ with a constant $1/k$ that drives the largest gains on the harder OOD slices ($+42\%$ relative on Medium, $+48\%$ relative on Hard over the prior WebGym SOTA).

## 2 Related Work

**System: visual, multi-step, and async RL.** Multi-step RL on VLM policies has been studied for web browsing (Bai et al., 2026; Wei et al., 2025; Yang et al., 2026b), computer use (Bai et al., 2024; 2025; Wang et al., 2025), embodied learning (Kim et al., 2025; Liu et al., 2025a), and sequential decision-making (Zhai et al., 2024). Open-source frameworks split into two disjoint subsets that AsyncWebRL bridges. *Async LLM-RL on text* and *async single-turn VLM-RL* (Fu et al., 2025; Lu et al., 2025; Sheng et al., 2025; Wu et al., 2025; Han et al., 2025; Zhong et al., 2025; Zhou et al., 2025; Noukhovitch et al., 2024; Sheng et al., 2024) ship at most one image per training example; pushing multi-turn web-agent rollouts (tens of high-resolution screenshots per trajectory, hundreds concurrent) through their shared data store exhausts its budget and triggers a disk-spill path that erases the async benefit. *Sync multi-step VLM-RL* (Bai et al., 2026; Wei et al., 2025; Bai et al., 2024; 2025; Yang et al., 2026a) carries the right workload but resynchronizes at every iteration boundary, leaving GPU bubbles. DART-GUI (Li et al., 2025) also decouples rollout and training for multi-turn visual agents, but targets a substantially more I/O-bound OS-level regime, where slow environment interaction and adaptive data curation are the primary bottlenecks; in contrast, our high-cadence web setting is limited by sustained screenshot traffic and iteration-level synchronization. Switching synchronous multi-step VLM-RL to fully async execution opens an off-policy gap that sync frameworks never see: training batches contain tokens drawn from several policy versions at once, and the trust region must be re-centered on a proximal policy (Hilton et al., 2021) so that clipping reflects current-update movement rather than rollout staleness. The closed-source UI-TARS-2 (Wang et al., 2025) reportedly satisfies all three properties but releases only model weights. AsyncWebRL is, to our knowledge, the first open framework to combine all three in the high-throughput visual web-agent setting.

[1]Throughout the paper, "step-number normalization" refers to the multi-step GRPO factor $1/|\tau_i|$ that normalizes the loss by the number of *steps* (not tokens) in a trajectory. This is distinct from Dr. GRPO's removal of the single-turn factor $1/|y_i|$, which normalizes by the number of tokens in a response. We call the latter "token-number normalization."

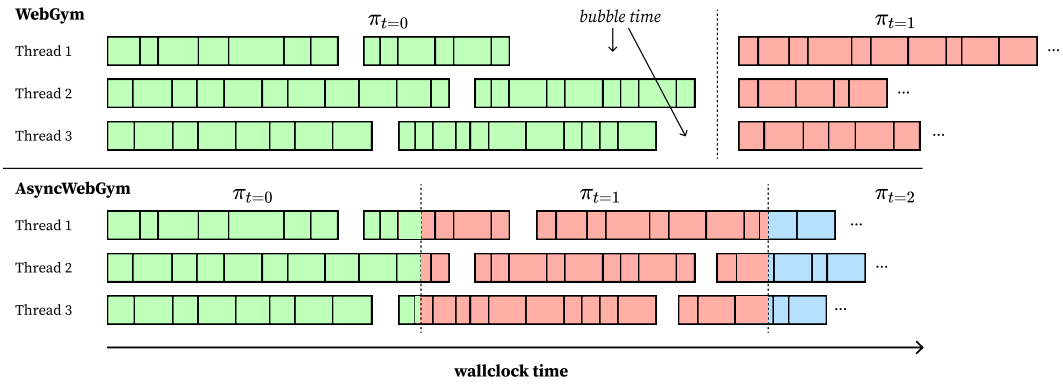



Figure 1: **Multi-step Asynchronous Management.** Compared to WebGym, AsyncWebRL eliminates the *inter-iteration bubble time* caused by reconstructing the rollout pool at every iteration boundary and waiting for the policy refresh of $\pi_t$ to complete. Colored blocks denote concurrent rollout workers producing trajectories, gradient updates on $\pi_t$, and policy refreshes that broadcast new weights to the rollout workers. White gaps under synchronous RL (top) are bubble time. AsyncWebRL (bottom) eliminates these gaps by maintaining an everlasting rollout pool so that rollout, gradient update, and policy refresh overlap continuously.

**Algorithm: loss shape under length asymmetry.** On the algorithmic side, Dr. GRPO (Liu et al., 2025b) showed that the single-turn GRPO normalizer $1/|y_i|$ reweights per-token gradients between long and short responses, and removing it (replacing $1/|y_i|$ with a constant) corrects an unintended length bias inside a single response. We extend the same diagnosis one level up. In multi-step GRPO (Shao et al., 2024; Zeng et al., 2025), the analogous factor is the per-trajectory step-number normalizer $1/|\tau_i|$, which couples per-token gradient scale to trajectory length. What is new at the step granularity that the single-turn analysis does not see is that the multi-step web-agent setting has two structural features which turn Dr. GRPO's step-level prediction into a self-reinforcing failure mode: failed trajectories are dominated by horizon exhaustion rather than clearly wrong actions, and an additive memory prompt against a multi-step visual environment mechanically lengthens per-step responses as step count grows. We trace these features in Section 5 and show that the multi-step counterpart of Dr. GRPO's fix breaks the loop at its source.

# 3 AsyncWebRL

## 3.1 System

Our goals are to keep GPUs from waiting on each other, and to keep the shared inter-worker store from saturating under per-step image traffic. On top of the synchronous multi-step rollout pool of WebGym (Bai et al., 2026), we make two changes: *fully asynchronous execution with an everlasting rollout pool*, and *lightweight screenshot handling*. The first keeps rollout workers continuously alive across iteration boundaries and overlaps rollout, gradient update, and policy refresh end to end: when an episode ends, the next begins immediately on the same worker without waiting for the rest of the batch or for the next iteration. The parameter update of $\pi_\theta$ can happen at any time while rollout continues; new weights are broadcast in place to the inference workers and the next rollout segment is sampled under the updated policy (Figure 1). The second (Section B) keeps per-step image tensors out of the shared store and routes only lightweight references between rollout workers and the trainer, which prevents the disk-spill path that hundreds of concurrent high-resolution rollouts otherwise induce. Together these eliminate the GPU idle time spent waiting for the slowest trajectory in a batch and the warm-up cost paid on hundreds of browser sessions when the rollout pool is rebuilt each round.

## 3.2 Algorithm

**Decoupled off-policy correction.** Asynchronous execution shifts the sampling distribution: the policy that generated a given token ($\pi_{\text{behave}}$) is several updates behind the policy $\pi_\theta$ the trainer is now updating, and a single trajectory in a training batch can be stitched from different policy snapshots.

$$\mathcal{J}(\theta) = \mathbb{E}_{\tau\sim\pi_{\text{behave}}}\left[\frac{1}{G\cdot k}\sum_{i=1}^{G}\sum_{j=1}^{|\tau_i|}\sum_{t=1}^{|\tau_{i,j}|}\min\left(\frac{\pi_\theta}{\pi_{\text{behave}}}\hat{A}_i,\ \frac{\pi_{\text{prox}}}{\pi_{\text{behave}}}\,\text{clip}\left(\frac{\pi_\theta}{\pi_{\text{prox}}},\,1{-}\epsilon,\,1{+}\epsilon\right)\hat{A}_i\right)\right]. \quad (1)$$

The natural correction is importance sampling: weight every token by $\pi_\theta/\pi_{\text{behave}}$ for an unbiased off-policy gradient, and clip this ratio to $[1-\epsilon, 1+\epsilon]$ as in PPO. Under async RL the same ratio must capture two quantities at once: how much the policy has moved since the rollout was sampled (*rollout staleness*), and how much the optimizer has moved the policy during the current gradient update. Clipping a single coupled ratio confounds the two, so rollout staleness alone triggers clip events at many token positions and slows training substantially. We adopt the decoupled-PPO factorization of Hilton et al. (2021) (also used by AReaL (Fu et al., 2025) for text RL) and split the ratio into a rollout-staleness factor $\pi_{\text{prox}}/\pi_{\text{behave}}$ and a current-update factor $\pi_\theta/\pi_{\text{prox}}$, where $\pi_{\text{prox}}$ is the policy snapshot at the start of the current update. The PPO clip is then centered around $\pi_{\text{prox}}$ only, so that clip events reflect how much the optimizer has moved the policy during this update rather than staleness inherited from the rollout. Empirically this roughly halves the clip rate relative to the coupled formulation (Section 4.3), and enters the final loss as the inner $\min$ of Equation (1) (a negative-advantage dual-clip extension is given in Equation (2)).

**Removing trajectory-length normalization.** Separately from the off-policy correction, we change the loss aggregation. The standard multi-step GRPO (Shao et al., 2024) loss normalizes each rollout by its own step count $|\tau_i|$, which under our setting introduces a length-coupled gradient attenuation we trace in Section 5. The fix is one line: replace $1/|\tau_i|$ with a constant $1/k$, where $k$ is the Easy-difficulty horizon (10 throughout this paper). That is, instead of every rollout entering the loss with total weight 1 regardless of length, each rollout enters with weight $|\tau_i|/k$, restoring full per-token gradient weight on the long failures the policy must learn to avoid. This change appears as the constant $1/k$ outer factor in Equation (1).

We treat each trajectory as a bandit and share a single trajectory-level advantage $\hat{A}_i = (r_i - \text{mean}(\mathbf{r}))/\text{std}(\mathbf{r})$ across all tokens in trajectory $i$. With $G$ rollouts per task, $|\tau_i|$ steps per rollout, and $|\tau_{i,j}|$ tokens per step, the AsyncWebRL loss combines the two changes above with the standard GRPO surrogate (Equation (1)).

The red-colored terms in Equation (1) mark these two changes: the constant $1/k$ in place of $1/|\tau_i|$, and the decoupled clip centered around $\pi_{\text{prox}}$ with $\pi_{\text{prox}}/\pi_{\text{behave}}$ as an unclipped weight. Following DAPO (Yu et al., 2025), we also drop the reference-KL term and apply dynamic sampling, skipping all-success or all-failure groups and gathering 128 mixed trajectories (16 groups) before launching training of that step.

**RAFT++ as a baseline.** As a contrasting off-policy baseline we also consider RAFT++ (Xiong et al., 2025), which can be viewed as vanilla multi-step GRPO with the same per-trajectory normalizer $1/|\tau_i|$, but with group normalization disabled and the group-relative advantage replaced by a success filter ($r > 0$). Only successful trajectories contribute gradient, so RAFT++ effectively performs behavior cloning on a rolling buffer of positives and provides no contrastive signal on below-average trajectories. Decoupled importance sampling is still applied to keep the off-policy gradient unbiased.

# 4 Experiments

## 4.1 Setup

**Environment.** We train and test under the protocol from WebGym (Bai et al., 2026). We picked WebGym because it is the largest open multi-step visual web-agent training environment to date: roughly 290k training tasks across 128k real-world websites in three difficulty levels (Easy, Medium, Hard), evaluated on a 1,167 task OOD test split whose websites do not appear in training. The scale of the OOD test split (1,167 tasks across held-out websites) is what makes single-seed comparisons informative here. WebGym's own analysis reports that two runs differing in early-training trajectory dynamics nonetheless converge to overlapping final test performance, with error bars across runs small enough that single-seed comparisons are informative (Bai et al., 2026, Figure 8). We inherit this property and report single-seed results throughout, focusing reporting bud-

Table 1: Peak test set success rate (%) on the WebGym (Bai et al., 2026) OOD test split. Best per column within each variant in bold.

| Algorithm | Easy | Med. | Hard | Avg |
|---|---|---|---|---|
| `Model: Qwen3-VL-8B-Instruct` | | | | |
| Base (no RL) | 32.5 | 11.2 | 0.0 | 26.2 |
| WebGym (sync REINFORCE) | 50.9 | 24.1 | 4.8 | 42.9 |
| AsyncWebRL-RAFT++ | 46.6 | 27.8 | 5.5 | 39.3 |
| **AsyncWebRL (full)** | **52.4** | **34.3** | **7.1** | **45.4** |
| `Model: Qwen3-VL-8B-Thinking` | | | | |
| Base (no RL) | 37.4 | 24.3 | 1.2 | 32.0 |
| AsyncWebRL-RAFT++ | 47.3 | 30.0 | 5.2 | 40.5 |
| **AsyncWebRL (full)** | **51.8** | **35.1** | **11.3** | **44.4** |

get on algorithm-level ablations rather than seed sweeps. Rewards are binary, produced by WebGym's GPT-4o rubric evaluator. Following WebGym, we use the coordinate-based action space {`click`, `type`, `scroll`, `go_back`, `navigate`, `ANSWER`} on raw screenshots, with per-difficulty horizons of $(10, 20, 30)$ steps. The Easy-difficulty horizon $k{=}10$ is the WebGym recommended trade-off between truncation and rollout cost, and our constant $1/k$ fix uses $k{=}10$ throughout. All hyperparameter details are in Section C.5.

**Model Variants.** We use two Qwen3-VL-8B variants: Instruct (Qwen3-VL-8B-Instruct) and Thinking (Qwen3-VL-8B-Thinking), both with the additive memory prompt template shown in Section C.4. At each step the policy sees only the previous screenshot and its previous response. For the Thinking variant the previous step's thinking tokens are hidden from history, following standard practice (Bai et al., 2026).

**Comparisons.** REINFORCE style on-policy algorithms present a type mismatch for an async framework that is not on-policy by design. We therefore compare to the closest off-policy baseline, RAFT++ (defined in Section 3.2). We run three configurations:

- **WebGym**: prior synchronous RL pipeline with the Filtered BC objective of Bai et al. (2026), the published WebGym numbers.
- **AsyncWebRL-RAFT++**: our async framework with RAFT++ as the loss, the async compatible substitute for WebGym's sync REINFORCE.
- **AsyncWebRL (full)**: our async framework with the full method from Section 3, namely multi-step GRPO with decoupled importance sampling and the constant $1/k$ step-number normalizer.

### 4.2 Main Results

**AsyncWebRL (full) sets a new SOTA over WebGym, with the largest relative gains on harder slices.** On Instruct, AsyncWebRL (full) reaches $45.4\%$ Avg against WebGym's $42.9\%$, a $\mathbf{+5.8\%}$ **relative** improvement (Table 1), but Avg understates where the gain lives: **Easy** barely moves ($50.9 \rightarrow 52.4$, $+2.9\%$ relative), **Medium** widens to $\mathbf{+42\%}$ **relative** ($24.1 \rightarrow 34.3$), and **Hard** reaches $\mathbf{+48\%}$ **relative** ($4.8 \rightarrow 7.1$). The same harder-slice pattern shows up on Thinking in Figure 2 (bottom row), where WebGym's reported numbers are not available at our training horizons. Medium and Hard are the slices on which prior REINFORCE-style baselines have the most room to grow, because successful trajectories there are rarer and a behavior-cloning loss on a slowly changing positives buffer cannot push down the dominant failure modes. GRPO's group-normalized advantage does (Section A.2, Section A.1).

**Reproducing the WebGym sync REINFORCE pipeline under async.** Before comparing algorithms, we use AsyncWebRL-RAFT++ as a like-for-like check against the published WebGym sync REINFORCE numbers (same loss family, async framework substituted for sync). AsyncWebRL-RAFT++ on Instruct reaches 39.3%, against 42.9% for the prior WebGym pipeline (Table 1). The $3.6\%$ gap is consistent with the importance sampling overhead any async framework has to pay, and the two methods land in the same neighborhood on the same loss family, with the gain reported above

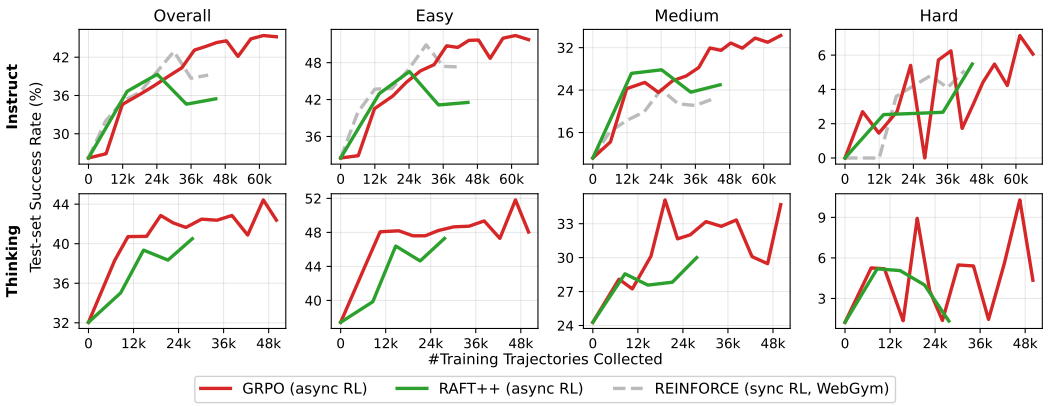



Figure 2: Test success rate vs. training trajectories collected on the WebGym OOD test split. Solid colored curves are our runs under AsyncWebRL: AsyncWebRL (full) and AsyncWebRL-RAFT++. The gray dashed curve is the prior WebGym sync REINFORCE baseline (values from Bai et al. (2026)). *Top:* Instruct. *Bottom:* Thinking. No WebGym curve is shown on Thinking because its baseline was not trained under our $(10, 20, 30)$ per difficulty horizons. AsyncWebRL (full) achieves both higher final test reward and higher sample efficiency per training trajectory collected.

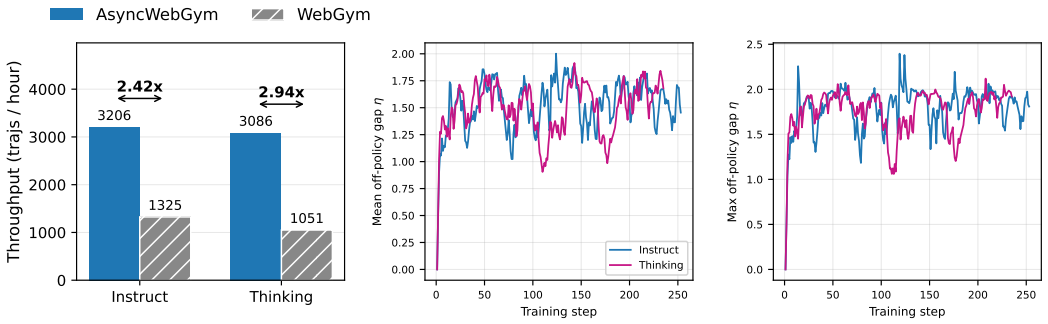



Figure 3: **Left:** End to end training trajectory throughput. **Right:** Off-policyness during GRPO training for the Instruct (blue) and Thinking (red) Qwen3-VL-8B runs: mean and max of the per-token off-policy gap $g$ across a training batch.

being net of this cost. Wall-clock cost moves the other way: AsyncWebRL produces about 3,100 training trajectories per hour on both variants against approximately 1,300 and 1,050 trajectories per hour for the synchronous WebGym baseline, a **2.4 to 2.9× end-to-end speedup** (Figure 3, left). The speedup also incorporates smaller engineering improvements over WebGym, including the distributed inference server and in-memory weight broadcast. Since WebGym (Bai et al., 2026) itself reports the fastest open-source rollout system for multi-step visual web agents to date, the 2.4 to 2.9× end-to-end speedup on top of it makes AsyncWebRL, to our knowledge, *the fastest reported open multi-step RL framework for visual web agents*. We qualify with "reported" because among open multi-step VLM RL frameworks in this category, only WebGym publishes throughput numbers.

**Off-policyness stays small.** A natural worry under fully async RL is that the off-policy gap might be large. AReaL (Fu et al., 2025), the closest async text reasoning analog, reports running with a max staleness of $\eta$=4 for coding and $\eta$=8 for math. In our setting, with the max staleness set to $\eta = 2$, the mean per-token off-policy gap $\eta$ stays near 1.5 and the max near 2.0 throughout training (Figure 3, right two panels), well below the cap, and GPUs stay busy throughout. Off-policyness is naturally smaller than in coding and math because web-agent responses are much shorter (Shen et al., 2025), giving a higher rollout/train speed ratio and leaving few stale trajectories in the training batch.

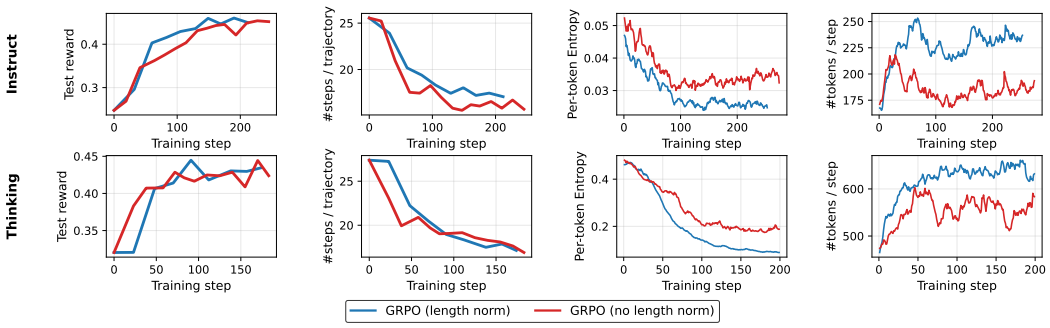



Figure 4: Effect of the $1/|\tau_i|$ normalizer on GRPO training dynamics. Rows are the two Qwen3-VL-8B variants (top: Instruct, bottom: Thinking), columns are, from left to right, test reward, #steps per trajectory, per-token entropy, and #tokens per step. Test reward is essentially tied between the two losses, but the $1/|\tau_i|$ run produces longer trajectories, longer per-step responses, and lower per-token entropy.

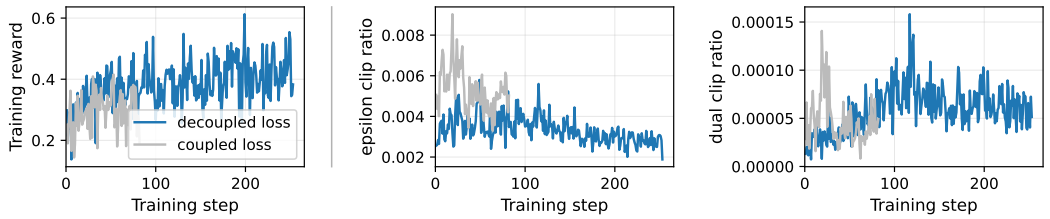



Figure 5: Coupled vs. decoupled importance sampling under the async RL GRPO loss. From left to right: per update mean of training reward, fraction of tokens hit by the $\epsilon$-clip, and fraction of tokens hit by the dual clip.

### 4.3 Efficiency

**Removing $1/|\tau_i|$ preserves performance and shortens trajectories.** The constant $1/k$ replacement does not change what the policy can solve: test reward is essentially tied throughout training on both variants (Figure 4, first column). What changes is trajectory and per-step response length. Under the standard $1/|\tau_i|$ normalizer the average trajectory is consistently longer, and the per-step response grows to around 240 tokens; the constant $1/k$ run uses fewer steps per trajectory and fewer tokens per step at matched test reward. Measured within each run (same hardware throughout), the trajectory contraction translates into a per-step gradient-update time reduction of **11–15%** from the first 20 to the last 20 training steps under the constant $1/k$ fix, against only 3–5% under the standard $1/|\tau_i|$ loss; the corresponding total per-step wall-clock reductions are **18–19%** vs. 4–5% (full table in Section A.5). The mechanism behind this, length-coupled memory drift, is analyzed in Section 5.

**Decoupled importance sampling halves the clip rate.** A separate ablation isolates the off-policy correction described in Section 3.2. We run a coupled loss baseline that folds rollout staleness and current-update movement into a single $\pi_\theta/\pi_{\text{behave}}$ ratio with PPO-style clipping centered on the coupled ratio. The coupled run sits at roughly $2\times$ the $\epsilon$-clip-trigger rate of the decoupled run at start and improves training reward substantially more slowly (Figure 5). We therefore keep the decoupled factorization in every AsyncWebRL configuration above.

## 5 Dynamics Analysis

### 5.1 Main Analysis

The $1/|\tau_i|$ factor in multi-step GRPO is the step-level analog of the token-level normalizer Dr. GRPO (Liu et al., 2025b) identifies as biased: each trajectory enters the loss with total weight 1 regardless of length, so any token in a long trajectory carries a $1/|\tau_i|$ share. In a setting like ours where failure is dominated by horizon exhaustion rather than clearly wrong actions, this attenuates the penalty on exactly the trajectories the policy needs to learn to avoid. On the base model, only around 30% to 40% of rollouts succeed in the DAPO-filtered training batch, and failures average

**Base**

```
step 4
Memory: {
  "Current website": "https://www.ml.cmu.edu",
  "Task": "Find specific email addresses for prospective
     PhD and Master's student inquiries at the CMU
     Machine Learning Department.",
  "Progress": "Scrolled down the webpage multiple times
     to find contact information or a 'Join Us' section."
}
```

**RAFT++ (with length norm)**

```
step 4
Memory: {
  "target_website": "https://www.foodnetwork.com/",
  "task": "Find a recipe for a vegetarian dish that
     includes a list of ingredients, cooking
     instructions, and a photo.",
  "search_query": "vegetarian recipe",
  "found_recipe": "Baked Penne with Roasted Vegetables"
}
```

**GRPO (with length norm)**

```
step 4
Memory: {
  "task": "Find a recipe for a healthy lunch that can
     be made in 30 minutes or less and includes a
     list of ingredients and step-by-step
     instructions.",
  "current_page": "https://www.foodnetwork.com",
  "search_term": "healthy lunch recipe 30 minutes or less",
  "search_result": "Roasted Cauliflower (25 minutes)",
  "recipe_title": "Roasted Cauliflower",
  "recipe_page": "https://www.foodnetwork.com",
  "ingredients_and_instructions": "not found"
}
```

**GRPO (no length norm)**

```
step 4
Memory: {
  "insulin protein from Gallus gallus":
     "not found yet",
  "review status of insulin protein from
     Gallus gallus": "not found yet"
}
```

Figure 6: `Memory` at step 4 of one representative 10/20/30-horizon rollout per checkpoint. $1/|\tau_i|$ accumulates verbose generic keys; the constant $1/k$ fix re-keys `Memory` to task sub-questions.

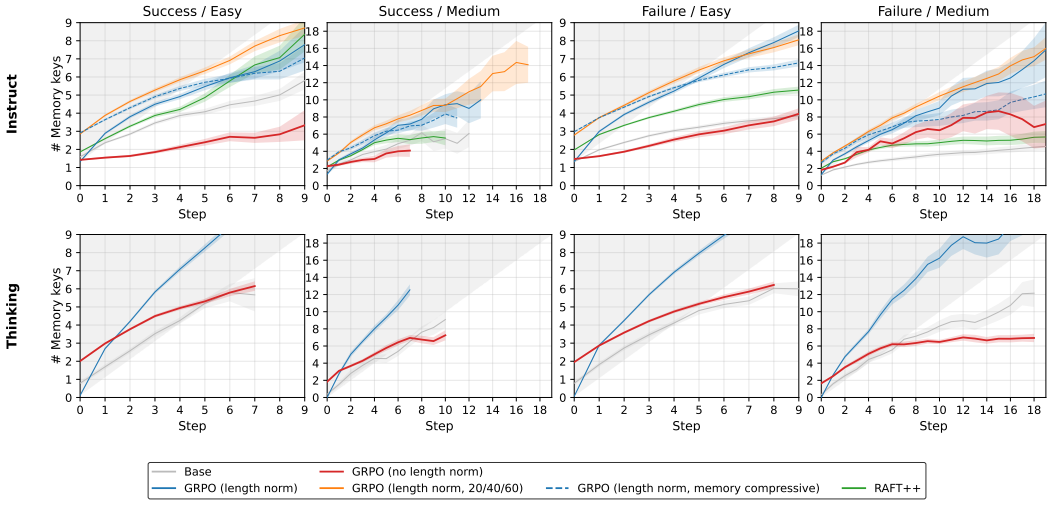



Figure 7: Trajectory-mean number of `Memory` JSON keys per agent step, split by outcome (Success, Failure) and difficulty (Easy, Medium). $1/|\tau_i|$ tracks the one-new-key-per-step diagonal; the constant $1/k$ fix stays close to Base.

12.5 steps against 5.1 for successes (a $2.4\times$ gap) when RL starts. The short successful population is both rarer and shorter and cannot counterbalance the effect of long failed trajectories, so early in training, the policy is pulled toward longer rollouts (Figure 4, second column).

This lengthening of trajectories then propagates into longer per-step responses through the prompt structure. Under the WebGym additive-memory prompt against a multi-step visual environment (Bai et al., 2026), each step appends to `Memory`, and each new screenshot exposes new information the policy can legitimately log, so `Memory` grows roughly linearly in step count. Under $1/|\tau_i|$ the per-step `Memory` key count tracks the one-new-key-per-step diagonal, while under the constant $1/k$ fix it stays close to the base-model baseline (Figure 7). Since `Memory` is part of the response, the larger step count drags per-step response length up by $\approx 33\%$ (Figure 4, fourth column), and the entropy drop (third column) follows because the extra response length is mostly low-entropy `Memory` boilerplate (per-token entropy annotations on representative responses in Section C.3). The schema itself also drifts: under $1/|\tau_i|$ the typical trajectory rewrites `Memory` every step with generic placeholders like `task_1` or `current_step` ($34\%$ of all observed key-occurrences are generic-slot, only $7\%$ of trajectories preserve their key set end-to-end, only $36\%$ of step pairs leave the `Memory` key set unchanged), while under the constant $1/k$ fix the policy declares a small set of task-anchored keys at step 0 and holds them throughout ($65\%$, $76\%$, generic slots drop to $11\%$, Table 2 and Figure 6; token-level last-step

| | GRPO | No LN |
|---|---|---|
| $\text{keys}_0 = \text{keys}_{\text{last}}$ | 7% | **65%** |
| No-edit step pairs | 36% | **76%** |
| Generic-slot keys | 34% | **11%** |

Table 2: Schema signatures of `Memory` under GRPO with $1/|\tau_i|$ vs. the constant $1/k$ fix (Instruct, additive prompt, first 30 steps per trajectory). Rows: fraction of trajectories with first-step and last-step key sets equal; fraction of consecutive step pairs with the same key set; fraction of key occurrences matching a generic placeholder regex (e.g. `task`, `current_step`, `search_query`).

responses in Sections C.1 and C.2). The constant $1/k$ fix restores full per-token weight on long failures, and every downstream symptom above (memory bloat, longer responses, lower entropy, generic schemas) disappears.

### 5.2 Ablations

We run three ablations to confirm that the diagnosis above identifies the right cause. The first checks that a closely related algorithm (RAFT++) exhibits the same pathology when it shares the same loss factor, ruling out group-relative advantage as the driver. The second checks that the cause lives in the loss rather than the prompt, by trying to fix the symptom from the prompt side and observing that it does not work. The third checks that the effect scales as the mechanism predicts when we make long failures even longer.

**RAFT++ exhibits the same pathology, more mildly.** RAFT++ also normalizes per-trajectory loss by $1/|\tau_i|$, and the diagnosis predicts the same memory bloat in kind. Figure 7 (Instruct row) confirms it: the RAFT++ per-step `Memory` key-count curve drifts upward in the same direction as GRPO with $1/|\tau_i|$, closely shadowing the GRPO length-norm curve on Easy successful trajectories and sitting well above the constant-$1/k$ run. The effect is muted on failure trajectories, consistent with RAFT++'s success-conditioned filter ($r > 0$) providing no direct gradient on failed rollouts: the same length-coupled attenuation is present in the loss but only propagates through the kept-positives. The matched per-step Add/Del/Mod edit-op decomposition (Section A.3) tells the same story at a smaller magnitude than full GRPO. RAFT++ and GRPO differ in nearly every other respect, yet both show the same length-coupled drift whenever they share the $1/|\tau_i|$ factor. The factor itself, not the surrounding algorithm, is the driver.

**Prompt-level intervention fails.** If the loss is what drives the pathology, not the prompt, then intervening at the prompt level should not help. Swapping the additive-memory prompt for a compressive one that explicitly instructs the model to compress `Memory` at every step, while keeping $1/|\tau_i|$ in the loss, leaves per-step Add/Del/Mod rates elevated relative to the Base run (Table 3, "GRPO, compressive"). The prompt change does not reach the cause.

**Scaled horizons amplify the effect.** The mechanism predicts that anything making long failures even longer should make the symptoms worse. Scaling per-difficulty horizons from 10/20/30 to 20/40/60 while keeping $1/|\tau_i|$ does exactly that, and we see the predicted amplification in Figure 7: the *GRPO (length norm, 20/40/60)* curve tracks the one-new-key-per-step diagonal across both Easy and Medium panels, exceeding the default $10/20/30$ length-norm run.

## 6 Conclusion

We presented AsyncWebRL, a fully asynchronous multi-step RL framework for visual web agents. An everlasting rollout pool and lightweight screenshot handling give a 2.4–2.9× end-to-end speedup over the fastest prior open pipeline; replacing $1/|\tau_i|$ with a constant $1/k$ then contracts trajectories at matched reward, with an 18–19% within-run reduction in total time per training step. A decoupled importance-sampling correction sets a new open SOTA on WebGym's OOD split (45.4% vs. 42.9%).

### AI use statement

We used generative AI tools to write part of the code and to edit the paper.

### Ethics statement

Web agents can be misused to automate harmful or unauthorized actions. Systems built from this work should therefore be evaluated in sandboxed environments and deployed with access controls, monitoring, and explicit user authorization. Our experiments use benchmark tasks and do not introduce new human-subject data. We report aggregate evaluation results and do not use benchmark outcomes to make decisions about individuals.

### Reproducibility statement

The main text specifies the training objective, system design, evaluation protocol, and principal ablations. The appendix provides the implementation details, prompts, hyperparameters, hardware allocation, and additional analyses needed to reproduce and diagnose the reported results. The cited WebGym task set and baseline implementation provide the underlying benchmark resources.

## References

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

Hao Bai, Yifei Zhou, Li Erran Li, Sergey Levine, and Aviral Kumar. Digi-q: Learning q-value functions for training device-control agents. *arXiv preprint arXiv:2502.15760*, 2025.

Hao Bai, Alexey Taymanov, Tong Zhang, Aviral Kumar, and Spencer Whitehead. Webgym: Scaling training environments for visual web agents with realistic tasks. *arXiv preprint arXiv:2601.02439*, 2026.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International conference on machine learning*, pp. 1407–1416. PMLR, 2018.

Wei Fu, Jiaxuan Gao, Xujie Shen, Chen Zhu, Zhiyu Mei, Chuyi He, Shusheng Xu, Guo Wei, Jun Mei, Jiashu Wang, Tongkai Yang, Binhang Yuan, and Yi Wu. Areal: A large-scale asynchronous reinforcement learning system for language reasoning, 2025.

Zhenyu Han, Ansheng You, Haibo Wang, Kui Luo, Guang Yang, Wenqi Shi, Menglong Chen, Sicheng Zhang, Zeshun Lan, Chunshi Deng, et al. Asyncflow: An asynchronous streaming rl framework for efficient llm post-training. *arXiv preprint arXiv:2507.01663*, 2025.

Jacob Hilton, Karl Cobbe, and John Schulman. Batch size-invariance for policy optimization, 2021.

Dongyoung Kim, Sumin Park, Huiwon Jang, Jinwoo Shin, Jaehyung Kim, and Younggyo Seo. Robot-r1: Reinforcement learning for enhanced embodied reasoning in robotics. *arXiv preprint arXiv:2506.00070*, 2025.

Pengxiang Li, Zechen Hu, Zirui Shang, Jingrong Wu, Yang Liu, Hui Liu, Zhi Gao, Chenrui Shi, Bofei Zhang, Zihao Zhang, Xiaochuan Shi, Zedong Yu, Yuwei Wu, Xinxiao Wu, Yunde Jia, Liuyu Xiang, Zhaofeng He, and Qing Li. Efficient multi-turn rl for gui agents via decoupled training and adaptive data curation. *arXiv preprint arXiv:2509.23866*, 2025.

Jijia Liu, Feng Gao, Bingwen Wei, Xinlei Chen, Qingmin Liao, Yi Wu, Chao Yu, and Yu Wang. What can rl bring to vla generalization? an empirical study. *arXiv preprint arXiv:2505.19789*, 2025a.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding r1-zero-like training: A critical perspective, 2025b.

Han Lu, Zichen Liu, Shaopan Xiong, Yancheng He, Wei Gao, Yanan Wu, Weixun Wang, Jiashun Liu, Yang Li, Haizhou Zhao, et al. Part ii: Roll flash–accelerating rlvr and agentic training with asynchrony. *arXiv preprint arXiv:2510.11345*, 2025.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pp. 1928–1937. PMLR, 2016.

Michael Noukhovitch, Shengyi Huang, Sophie Xhonneux, Arian Hosseini, Rishabh Agarwal, and Aaron Courville. Asynchronous rlhf: Faster and more efficient off-policy rl for language models. *arXiv preprint arXiv:2410.18252*, 2024.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

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

Junhong Shen, Hao Bai, Lunjun Zhang, Yifei Zhou, Amrith Setlur, Shengbang Tong, Diego Caples, Nan Jiang, Tong Zhang, Ameet Talwalkar, et al. Thinking vs. doing: Agents that reason by scaling test-time interaction. *arXiv preprint arXiv:2506.07976*, 2025.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv:2409.19256*, 2024.

Guangming Sheng, Yuxuan Tong, Borui Wan, Wang Zhang, Chaobo Jia, Xibin Wu, Yuqi Wu, Xiang Li, Chi Zhang, Yanghua Peng, et al. Laminar: A scalable asynchronous rl post-training framework. *arXiv preprint arXiv:2510.12633*, 2025.

Haoming Wang, Haoyang Zou, Huatong Song, Jiazhan Feng, Junjie Fang, Junting Lu, Longxiang Liu, Qinyu Luo, Shihao Liang, Shijue Huang, et al. Ui-tars-2 technical report: Advancing gui agent with multi-turn reinforcement learning. *arXiv preprint arXiv:2509.02544*, 2025.

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

Bo Wu, Sid Wang, Yunhao Tang, Jia Ding, Eryk Helenowski, Liang Tan, Tengyu Xu, Tushar Gowda, Zhengxing Chen, Chen Zhu, et al. Llamarl: A distributed asynchronous reinforcement learning framework for efficient large-scale llm training. *arXiv preprint arXiv:2505.24034*, 2025.

Wei Xiong, Jiarui Yao, Yuhui Xu, Bo Pang, Lei Wang, Doyen Sahoo, Junnan Li, Nan Jiang, Tong Zhang, Caiming Xiong, et al. A minimalist approach to llm reasoning: from rejection sampling to reinforce. *arXiv preprint arXiv:2504.11343*, 2025.

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. *arXiv preprint arXiv:2606.02031*, 2026a.

Rui Yang, Qianhui Wu, Zhaoyang Wang, Hanyang Chen, Ke Yang, Hao Cheng, Huaxiu Yao, Baoling Peng, Huan Zhang, Jianfeng Gao, et al. Gui-libra: Training native gui agents to reason and act with action-aware supervision and partially verifiable rl. *arXiv preprint arXiv:2602.22190*, 2026b.

Deheng Ye, Zhao Liu, Mingfei Sun, Bei Shi, Peilin Zhao, Hao Wu, Hongsheng Yu, Shaojie Yang, Xipeng Wu, Qingwei Guo, et al. Mastering complex control in moba games with deep reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 6672–6679, 2020.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025.

Siliang Zeng, Quan Wei, William Brown, Oana Frunza, Yuriy Nevmyvaka, Yang Katie Zhao, and Mingyi Hong. Reinforcing multi-turn reasoning in llm agents via turn-level credit assignment. In *ICML 2025 Workshop on Computer Use Agents*, 2025.

Yuexiang Zhai, Hao Bai, Zipeng Lin, Jiayi Pan, Shengbang Tong, Yifei Zhou, Alane Suhr, Saining Xie, Yann LeCun, Yi Ma, et al. Fine-tuning large vision-language models as decision-making agents via reinforcement learning. *Advances in neural information processing systems*, 37:110935–110971, 2024.

Yinmin Zhong, Zili Zhang, Xiaoniu Song, Hanpeng Hu, Chao Jin, Bingyang Wu, Nuo Chen, Yukun Chen, Yu Zhou, Changyi Wan, et al. Streamrl: Scalable, heterogeneous, and elastic rl for llms with disaggregated stream generation. *arXiv preprint arXiv:2504.15930*, 2025.

Yuzhen Zhou, Jiajun Li, Yusheng Su, Gowtham Ramesh, Zilin Zhu, Xiang Long, Chenyang Zhao, Jin Pan, Xiaodong Yu, Ze Wang, Kangrui Du, Jialian Wu, Ximeng Sun, Jiang Liu, Qiaolin Yu, Hao Chen, Zicheng Liu, and Emad Barsoum. APRIL: Active partial rollouts in reinforcement learning to tame long-tail generation. *arXiv preprint arXiv:2509.18521*, 2025.

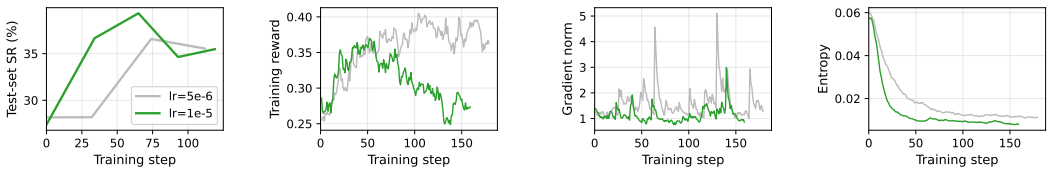



Figure 8: Learning-rate ablation on RAFT++ (`Qwen3-VL-8B-Instruct`, $b$=120, off-policy = 2, no KL). From left to right: held-out test success rate (the larger-LR run wins by a wide margin at the peak), training reward, where the trend reverses, pre-clipping gradient L2 norm (consistently bounded for the larger LR), and per-token policy entropy, which decays faster under the larger LR. The train/test flip suggests low-LR RAFT++ overfits the filtered positives buffer, noisier optimization generalizes better.

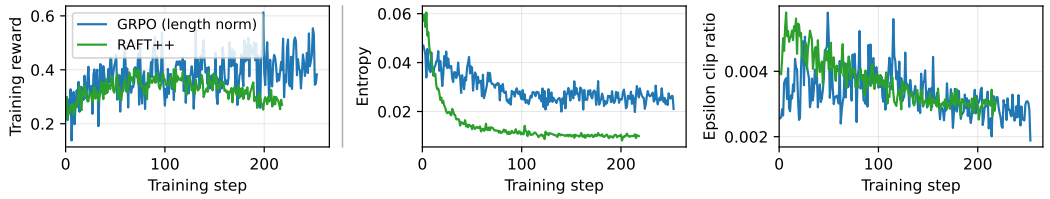



Figure 9: GRPO vs. RAFT++ on `Qwen3-VL-8B-Instruct`. *Left:* training reward per optimizer step. *Middle:* per-token policy entropy averaged over the loss mask, the step-zero offset is induced by the conditional vs. unconditional denominator (see text). *Right:* fraction of tokens hitting the $\epsilon$-clip, both runs sit near 0.3%, ruling out clip mechanics as the driver of the entropy gap.

# A Additional Analyses

This appendix collects supporting analyses for the main-text results. Section A.1 reports a learning-rate ablation on RAFT++. Section A.2 unpacks the GRPO-vs-RAFT++ gap with entropy and clip-rate diagnostics. Section A.3 provides the per-step `Memory` Add/Del/Mod table referenced by the Section 5 ablations. Section A.4 checks batch-size sensitivity.

## A.1 RAFT++ Learning Rate

Two RAFT++ runs on `Qwen3-VL-8B-Instruct` that share dataset, batch size ($b$=120), and every other hyperparameter except the learning rate ($5\times10^{-6}$ vs. $1\times10^{-5}$) move in *opposite* directions on training and held-out reward (Figure 8): the smaller-LR run reaches a higher peak training reward but a markedly lower peak test success rate; the larger-LR run gives up some training fit and gains substantially on the test set. We do not have a controlled experiment to attribute the flip causally, but the pattern is consistent with the RAFT++ objective behaving like behavior cloning on a slowly-changing positives buffer, where many small gradient steps can over-fit the kept successful trajectories. This observation motivates our choice of the higher learning rate ($1 \times 10^{-5}$) for the Thinking RAFT++ baseline reported in Table 5.

## A.2 Why GRPO Outperforms RAFT++

RAFT++ falls short of GRPO on both variants. We attribute this primarily to RAFT++ providing no contrastive signal on below-average rollouts: its $r > 0$ filter only pulls mass *onto* successful trajectories, while GRPO's group-normalized advantage simultaneously pushes mass *down* on below-average trajectories within the same group. The entropy gap on Instruct in Figure 9 is consistent with this picture: RAFT++'s entropy collapses from $0.057$ to $0.010$ while GRPO's settles near $0.025$, and $\epsilon$-clip rates are essentially identical ($\approx 0.3\%$ in both runs), so the gap is not driven by clipping mechanics. We do not run a controlled ablation that isolates the contrastive signal from other differences between the two losses.

Table 3: Per-step edit operations on `Memory` (instruct prompt, first 30 steps), split by trajectory outcome. Add / Del / Mod count per consecutive step pair the inserted / removed / value-changed keys.

| | | | Success | | | Failure | | |
|---|---|---|---|---|---|---|---|---|
| Algorithm | Prompt | Step-Num. Norm? | Add | Del | Mod | Add | Del | Mod |
| Base | additive | N/A | 0.65 | 0.08 | 0.41 | 0.22 | 0.02 | 0.23 |
| RAFT++ | additive | Yes | 0.64 | 0.08 | 0.47 | 0.44 | 0.06 | 0.43 |
| GRPO | additive | Yes | 1.11 | 0.26 | 0.83 | 1.13 | 0.31 | 1.10 |
| GRPO | compressive | Yes | 0.84 | 0.28 | 1.14 | 0.71 | 0.25 | 1.46 |
| **GRPO** | **additive** | **No** | **0.24** | **0.04** | **0.43** | **0.39** | **0.06** | **0.34** |

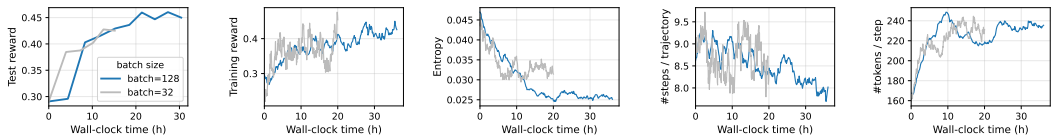



Figure 10: Effect of consumer batch size on GRPO Instruct training, plotted against wall-clock hours: batch=128 (the canonical setting used throughout the paper) versus batch=32, all other hyperparameters held fixed. From left: test reward, training reward, policy entropy, average steps per trajectory, and average response tokens per step.

### A.3 Memory Edit-Op Rates

Table 3 reports per-step Add/Del/Mod edit-op rates on the agent's `Memory` field, split by trajectory outcome, for the algorithm-prompt-loss combinations studied in Section 5. The matched per-step Memory key-count curves are plotted in Figure 7; this table reports the operation-level decomposition, and tells four qualitative stories. RAFT++ edits `Memory` more aggressively than Base, consistent with the shared $1/|\tau_i|$ factor pushing the policy toward longer memory schemas. GRPO with $1/|\tau_i|$ edits even more aggressively than RAFT++, because the group-relative advantage adds direct negative gradient on below-average trajectories that RAFT++'s success-conditioned filter does not provide. The compressive prompt under $1/|\tau_i|$ brings rates down from full additive GRPO but stays above Base, indicating that prompt-level intervention only mitigates the loss-driven effect rather than removing it. Finally, replacing $1/|\tau_i|$ with the constant $1/k$, even using the additive memory prompt, brings the rates back to roughly Base levels (and below on Success), confirming that the loss aggregation is the root cause analyzed in Section 5.

### A.4 Batch Size Sensitivity

We chose a consumer batch size of 128 for the canonical Instruct GRPO run (Section C.5). To check whether our conclusions depend on this choice, we re-ran the same configuration with batch size 32, holding learning rate, group size, off-policy lag cap, and all other hyperparameters fixed (Figure 10). The small-batch run improves faster early (reaching $\approx 0.38$ test reward several hours ahead of batch-128, since a four-times-higher optimizer-update rate cashes in when the gradient signal is large) but intersects with the batch-128 run around 10 hours (reaching $\approx 0.40$), so over the full training horizon the choice is essentially neutral on test reward. The early gain comes with visibly higher per-iteration variance: batch-32 training reward, entropy, and steps per trajectory all swing within a wider envelope than batch-128 at every wall-clock time. Crucially, the qualitative behaviors analyzed in Section 5 (entropy collapse, falling steps per trajectory, rising tokens per step) still reproduce at batch size 32, so the length-coupled memory drift mechanism is a property of the loss shape rather than of the batch-size choice. The canonical batch size of 128 is therefore a sensible default, but the main qualitative claims of the paper do not depend on it.

### A.5 Within-Run Per-Step Time Contraction

To quantify per-step compute savings from the constant $1/k$ fix without cross-run hardware confounds, we report *within-run* ratios: the average time per training step over the last 20 steps divided by the average over the first 20 (Table 4). Each run executes on the same hardware throughout, so any

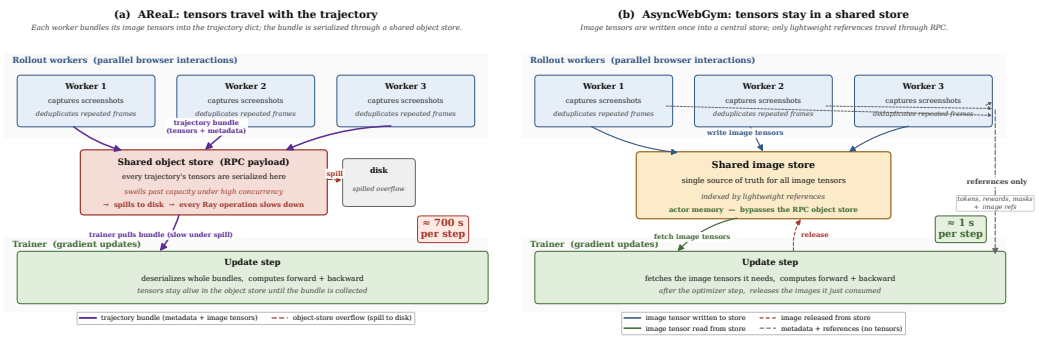



Figure 11: Lightweight Screenshot Handling. Compared to WebGym, which serializes every screenshot through the shared RPC object store and spills to disk under concurrent rollouts, AsyncWebRL keeps all image tensors in a dedicated in-memory actor and routes only lightweight *references* through RPC, eliminating the per-step object-store bottleneck.

change reflects the policy contracting (or expanding) its trajectories over training. We report two measurements. The first is the gradient-update time only (forward, backward, optimizer), which the trainer records directly for every training step. The second is the total time elapsed per training step, also including rollout collection, evaluation, and checkpointing. For the total time, we skip a small number of initial steps whose recorded timing reflects setup or resume overhead rather than steady-state training; for one of the Thinking $1/k$ fragments this means the first window starts a few dozen training steps in.

Table 4: Within-run ratio of last-20 to first-20 per-step time. Lower is more contraction.

| Run | Gradient | Total |
|---|---|---|
| Instruct, $1/\lvert\tau_i\rvert$ | 0.97 | 0.96 |
| Instruct, $1/k$ | **0.85** | **0.81** |
| Thinking, $1/\lvert\tau_i\rvert$ | 0.95 | 0.95 |
| Thinking, $1/k$ | **0.89** | **0.82** |

Both measurements agree. Under the standard $1/|\tau_i|$ loss, per-step time is essentially flat over the run (0.95–0.97): the policy's trajectory-length distribution drifts slowly enough that per-step cost stays roughly constant. Under the constant $1/k$ fix, per-step time contracts noticeably within a single run (0.81–0.89, an 11–19% reduction from start to end of training). This is the within-run signature of the trajectory contraction analyzed in Section 5: as training proceeds, the $1/k$ policy emits shorter trajectories and fewer tokens per step, so each gradient update consumes less compute.

# B Implementation Details

This appendix documents the system, loss, prompt, and hyperparameter details needed to reproduce our runs. The "System" subsection below describes the lightweight screenshot handling promised in Section 3.1. Section B.2 defines the dual-clip variant used by every AsyncWebRL GRPO run. Section C.4 provides the full agent prompt template (Instruct and Thinking share it). Section C.5 lists optimizer, loss, rollout, and hardware-allocation settings for all three main-table runs.

## B.1 System

**Lightweight screenshot handling.** Different from WebGym, which embeds raw image tensors directly into each trajectory dictionary and ships the entire bundle through the shared RPC object store, AsyncWebRL stores all screenshots in a **dedicated in-memory actor** and routes only lightweight references between rollout workers and the trainer, as shown in Figure 11. This redesign is forced by the multi-step web-agent setting: each trajectory carries tens of high-resolution screenshots, and at hundreds of concurrent rollouts the baseline pipeline pushes well over 100 GiB of pixel tensors

through the object store per training step. Once the object store exceeds its memory budget, the framework spills to disk and *every* downstream operation, trajectory deserialization, RPC dispatch, and tensor localization, degrades by orders of magnitude. By keeping image tensors out of the RPC payload entirely, AsyncWebRL confines the screenshot footprint to a single bounded actor, the trainer fetches only the slices it needs at gradient-update time and immediately releases them once the optimizer step completes.

### B.2 Optimization

There are two considerations of algorithm design that we adopt in our experiments.

**Dual-clipped GRPO loss.** Let $f(\rho)\hat{A}_{i,t}$ denote the inner $\min$ argument of Equation (1). This advantage-based clipping framework bounds the surrogate objective when $\hat{A}_{i,t} \geq 0$, but fails to do so when $\hat{A}_{i,t} < 0$ and $\rho \gg 1+\epsilon$: in this regime the inner $\min$ selects the unclipped term $\rho\hat{A}_{i,t}$, which is unbounded below in $\rho$. A single token assigned a low probability by the rollout policy can therefore drive the per-token contribution arbitrarily negative. Following the dual-clip extension of PPO (Ye et al., 2020), we cap $f(\rho)$ from below whenever the advantage is negative:

$$f_{\text{dual}}(\rho) = \begin{cases} f(\rho), & \hat{A}_{i,t} \geq 0, \\ \max\left[f(\rho),\ c \cdot \hat{A}_{i,t}\right], & \hat{A}_{i,t} < 0, \end{cases} \tag{2}$$

with constant $c > 1$. The dual-clipped GRPO loss is obtained by substituting $f_{\text{dual}}(\rho)$ for $f(\rho)$ in Equation (1), providing an absolute lower bound on the per-token contribution and eliminating the tail risk introduced by rare low-probability tokens with exploded importance sampling ratios.

**On not penalizing overlong trajectories specially.** A trajectory that exhausts its horizon is marked as a failure. DAPO (Yu et al., 2025) notes that this loses some potentially useful signal. We accept the loss: given the horizon, a trajectory that exhausted it means some of its actions were suboptimal, at least when another trajectory in the same group did succeed. Actions like *goback* that appear more often in long trajectories are something we want the policy to take only when necessary, not on every long rollout. When every trajectory in the group fails by exhausting the horizon, either those *goback*-style actions were genuinely unhelpful, or the horizon itself is too tight; either way, treating overlong rollouts as failures is the correct default.

## C Qualitative Examples

This appendix shows sample-level evidence for the length-coupled memory drift mechanism analyzed in Section 5. Section C.1 and Section C.2 contrast last-step responses produced by the $1/|\tau_i|$ run against the constant $1/k$ run on the two Qwen3-VL-8B variants, making the generic-slot memory schema versus task-anchored schema split visible at the token level. Section C.3 overlays the same responses with per-token entropy annotations, which lets the reader see directly that the extra tokens emitted under $1/|\tau_i|$ sit in low-entropy boilerplate regions rather than in decision-bearing positions.

### C.1 Instruct Last-Step Response

These examples illustrate the qualitative difference described in Section 5.

**Base · n_steps=10 · reward=1.0 · 178 tokens**

**Memory:** {"Task": "Find out when the next full moon is scheduled according to Space.com's 'Space Calendar'."}**Progress:** {"Go to Space.com": "finished", "Navigate to 'Space Calendar'": "finished", "Find next full moon date": "finished"}Intention: The next full moon is scheduled for May 1, 2026, at 1:23 p.m. ( EDT).**Action:** The next full moon is scheduled for May 1, 2026, at 1:23 p.m. (EDT).**<tool_call>**{"name": " computer_use", "arguments": {"action": "answer", "text": "The next full moon is scheduled for May 1, 2026, at 1:23 p.m. (EDT)."}}**</tool_call><|im_end|>**

**RAFT++ · n_steps=10 · reward=1.0 · 174 tokens**

**Memory:** {"target_website": "https://www.foodnetwork.com/", "task": "Search for a recipe for a dessert that is suitable for a romantic dinner, such as a chocolate soufflé or a fruit tart.", "search_term": "fruit tart", " results_found": "yes", "recipe_displayed": "Fruit-Tart Flag"}**Progress:** {"Go to foodnetwork.com": "finished ", "Search for a romantic dinner dessert recipe": "finished"}Intention: The task is completed as a suitable

recipe for a romantic dinner has been found.**Action:** The task is completed as a suitable recipe for a romantic dinner has been found.**<tool_call>**{"name": "computer_use", "arguments": {"action": "answer", "text": "A suitable recipe for a romantic dinner has been found: Fruit-Tart Flag."}}**</tool_call><|im_end|>**

**GRPO without length norm · n_steps=10 · reward=1.0 · 150 tokens**

**Memory:** {"accepted_cookies": "finished", "navigated_to_jackets": "finished", "found_leather_jacket_price": "finished"}**Progress:** {"Go to Diesel.com": "finished", "Accept cookies": "finished", "Navigate to 'Jackets' category for men": "finished", "Find price of any leather jacket": "finished"}Intention: Complete the task by providing the price of a leather jacket found in the 'Jackets' category**Action:** The task is completed. The price of a leather jacket is $395.00.**<tool_call>**{"name": "computer_use", "arguments": {"action": "answer", "text": "$395.00"}}**</tool_call><|im_end|>**

**GRPO with length norm · n_steps=10 · reward=1.0 · 589 tokens**

**Memory:** {"task": "Search for news articles about the latest technological advancements in the field of medicine and summarize the findings of the articles.", "search_query": "latest technological advancements in the field of medicine", "current_page": "NIH website article page", "search_results": ["AI teachers and cybertnetics - what could the world look like in 2050?", "'There is not enough time or space in the day'"], "article_title": "AI teachers and cybertnetics - what could the world look like in 2050?", "popup": "survey popup", "article_content": "AI teachers and cybertnetics - what could the world look like in 2050?", "subscription_popup": "active", "attempted_navigation": "https://www.bbc.com/news/health-64810698", "error_page": "500 Internal Server Error", "new_website": "https://www.nih.gov/", "search_bar": "used", "search_results_count": "2,141 results", "article_found": "SLEEP 2024, the Associated Professional Sleep Societies (APSS)'s 38th Annual Meeting | NHLBI, NIH", "article_content_summary": "The SLEEP 2024 meeting will discuss the latest advances in sleep science and new developments in the diagnosis and treatment of sleep disorders, disseminate cutting-edge research, and promote the translation of basic science into clinical practice."}
**Progress:** {"Go to BBC News": "finished", "Search for technological advancements in medicine": "finished", "Review search results": "in progress", "Read the article": "interrupted by subscription popup and error page", "Summarize the findings": "not finished", "Navigate to NIH website": "finished", "Search for articles on NIH website": "finished", "Review NIH search results": "finished", "Select and read an article": "finished", "Summarize the findings from the selected article": "in progress"}Intention: Provide the final answer based on the article summary**Action:** The task is completed. The latest technological advancements in the field of medicine, as discussed in the SLEEP 2024 meeting, include the latest advances in sleep science and new developments in the diagnosis and treatment of sleep disorders, disseminating cutting-edge research, and promoting the translation of basic science into clinical practice.**<tool_call>**{"name": "computer_use", "arguments": {"action": "answer", "text": "The latest technological advancements in the field of medicine, as discussed in the SLEEP 2024 meeting, include the latest advances in sleep science and new developments in the diagnosis and treatment of sleep disorders, disseminating cutting-edge research, and promoting the translation of basic science into clinical practice."}}**</tool_call><|im_end|>**

### C.2 THINKING LAST-STEP RESPONSE

**Base · n_steps=10 · reward=1.0 · 387 tokens**

<think>**Memory:** We have confirmed that ThinkTokyo.com is for sale on HugeDomains at $3,095. The page shows purchase options including "Buy now" and "Start payment plan," with additional details about the price and a money-back guarantee.**Progress:**- Task: Determine the current status of the domain 'thinktokyo.com' and if it is for sale, identify where it is listed for sale.- Completed: Confirmed the domain is for sale on HugeDomains with price and purchase options identified.- Remaining: No further steps needed since the task requirements are met.Intention: Complete the task by providing the final answer based on the observed information.**Action:** Provide the final answer.Given the information gathered from the page, the current status of the domain is clear: it is for sale on HugeDomains, priced at $3,095, with specific purchase options available. No additional actions are necessary because the task's requirements have been fully satisfied.
</think>**Memory:** {"Status of ThinkTokyo.com": "For sale", "Price": "$3,095", "Listing Site": "Huge Domains", "Purchase Options": ["Buy now", "Start payment plan"]}**Progress:** {"Check domain status": "finished", "Identify sale price": "finished", "Determine listing site": "finished", "Summarize findings": "not finished"}Intention: Summarize the findings to confirm the domain's status and sale details**Action:** Provide the final answer based on the observed information**<tool_call>**{"name": "computer_use", "arguments": {"action": "answer", "text": "The domain 'thinktokyo.com' is currently for sale on HugeDomains at a price of $3,095. It is listed with purchase options including 'Buy now' and 'Start payment plan'."}}**</tool_call><|im_end|>**

**GRPO without length norm · n_steps=10 · reward=1.0 · 612 tokens**

<think>Looking at the current screenshot, I can see we're on the Food Network website viewing the Simple Egg Drop Soup recipe. The task is to find a recipe for a popular soup dish that includes a list of ingredients

and cooking time.From the previous steps, I can see that:1. We successfully navigated to Food Network2. We searched for "popular soup recipes"3. We selected the "Simple Egg Drop Soup" recipe4. We've now scrolled down to see the recipe detailsThe current screenshot shows the recipe page with:- The recipe title "No Recipe Recipe: Simple Egg Drop Soup"- Level: Easy- Prep: 2 min- Cook: 10 min- Total: 12 min- Ingredients list (which includes chicken broth, soy sauce, Asian hot sauce, fresh ginger, and eggs)- Directions for making the soupThe task requires finding a recipe with a list of ingredients and cooking time. Based on what's visible in the screenshot, we have both:- The cooking time is clearly shown as "Total: 12 min" (with Prep: 2 min and Cook: 10 min)- The ingredients list is clearly visible in the "Ingredients:" sectionSince we've found a soup recipe with both the ingredients list and cooking time, the task is complete. The recipe meets all the criteria specified in the task.The next action should be to confirm that we've successfully completed the task.</think>
**Memory:** {"Current page": "Food Network website", "Task": "Find a recipe for a popular soup dish that includes a list of ingredients and cooking time", "Search term": "popular soup recipes", "Results": ["No Recipe Recipe: Simple Egg Drop Soup (12 minutes)", "Good Eats Roast Turkey"], "Selected recipe": "Simple Egg Drop Soup", "Cooking time": "12 minutes", "Ingredients": ["low-sodium chicken broth", "soy sauce", "Asian hot sauce", "fresh ginger", "eggs"]}**Progress:** {"Access Allrecipes": "failed", "Access Food Network": "success", "Search for soup recipes": "completed", "Selected recipe": "Simple Egg Drop Soup", "Cooking time found": "12 minutes", "Ingredients list found": "yes"}Intention: The task is complete - we've found a soup recipe with ingredients and cooking time**Action:** The task is complete. We've found a popular soup recipe (Simple Egg Drop Soup) with a list of ingredients and cooking time (12 minutes total).
**<tool_call>**{"name": "computer_use", "arguments": {"action": "answer", "text": "The Simple Egg Drop Soup recipe from Food Network has a cooking time of 12 minutes total (2 min prep, 10 min cook) and includes the following ingredients: low-sodium chicken broth, soy sauce, Asian hot sauce, fresh ginger, and eggs."}}
**</tool_call><|im_end|>**

**GRPO with length norm · n_steps=10 · reward=1.0 · 911 tokens**

<think>Let me analyze the current state and what we need to do:1. The task is to find the most common tree species in Central Park according to map data.2. We've been trying to access the NYC Tree Map website (https://tree-map.nycgovparks.org) but encountered permission issues.3. We've switched to using Wikipedia's Central Park page as an alternative source.4. We've scrolled through the Trees and plants section and found some information about tree species in Central Park.From the current screenshot, I can see we're still on the Wikipedia page for Central Park, specifically in the "Trees and plants" section. The text mentions:- Central Park has more than 20,000 trees (from a 2011 survey)- More than 170 tree species- American elms are mentioned as "largest remaining stands"- Other species mentioned include sugar maples, red maples, oak species, white spruce, tamarack, white oak, black cherryThe key information we need is the most common tree species. The text mentions "American elms are the largest remaining stands" which suggests they might be the most common species, but it's not explicitly stated as "most common" - just "largest remaining stands."The current view shows information about tree diversity and ecological importance. It mentions that "American elms are mentioned as largest remaining stands" which is the closest we have to identifying the most common species.Since we've already identified that American elms are mentioned as the largest remaining stands (which likely means they're the most common species), we've essentially completed our task.The next logical action would be to confirm this information and conclude our search since we've found the answer (American elms appear to be the most common tree species in Central Park based on the Wikipedia information).We've already scrolled through the relevant section and found the information we need. The task is essentially complete since we've identified that American elms are mentioned as the largest remaining stands, which is the closest we can get to identifying the most common tree species from the available information.</think>**Memory:** {"Initial website attempt": "https://tree-map.nycgovparks.org", "Second website attempt": "https://www.nycgovparks.org", "Error encountered": "Permission Denied", "Task objective": "Find most common tree species in Central Park according to map data", "Current status": "Unable to access NYC Parks websites due to permission issues", "Alternative source accessed": "Wikipedia's Central Park page", "Wikipedia page content": "Main article about Central Park with table of contents visible", "Trees and plants section accessed": "success", "Information extracted": "Central Park has more than 20,000 trees (2011 survey), more than 170 tree species, American elms are mentioned as largest remaining stands", "Most common species identification": "American elms are mentioned as largest remaining stands", "Additional species mentioned": "sugar maples, red maples, oak species (red oak, willow oak, swamp white oak, chestnut oak), white spruce, tamarack, white oak, black cherry", "Current section": "Trees and plants", "Task completion status": "completed"}**Progress:** {"Attempt to access NYC Tree Map": "completed", "Initial website loading": "failed", "Second website attempt": "completed", "Current website access": "denied", "Alternative information sources": "attempted (Wikipedia)", "Wikipedia page loaded": "success", "Relevant section identification": "completed (Trees and plants section)", "Information extracted": "Central Park has more than 20,000 trees, more than 170 tree species, American elms are mentioned as largest remaining stands", "Most common species identification": "completed (American elms identified as largest remaining stands)", "Task completion": "completed"}Intention: "Confirm that American elms are the most common tree species in Central Park based on the information found"**Action:** "The task is completed - the most common tree species in Central Park is American elms, as they are mentioned as the largest remaining stands in the Trees

and plants section of the Wikipedia page."**<tool_call>**{"name": "computer_use", "arguments": {"action": "answer", "text": "American elms"}}**</tool_call><|im_end|>**

### C.3 TOKEN-LEVEL ENTROPY

Each high-entropy token is annotated as `token (`$P_{\text{sampled}}$`) [alt₁:P₁, alt₂:P₂, . . . ]`, where $P_{\text{sampled}}$ is the probability the policy assigned to the token it actually sampled and the bracket lists the top-$k$ alternative tokens with their probabilities.
Qualitative results below shows the per-token entropy of one rollout from the Qwen3-VL-8B-Instruct policy trained with the $1/|\tau_i|$ normalizer, on the first and last turn of the trajectory. Token background opacity is proportional to entropy, high-entropy tokens carry a subscript listing the chosen token's probability and the top alternatives. Mean entropy collapses from 0.0297 nats at the first turn to 0.0077 nats at the last, with the few remaining high-entropy positions concentrated on coordinate slots and free-form summary fields that must integrate fresh screenshot evidence.

**step 0 · mean entropy=0.0297**

**Memory:** {"task": "Find the contact email for vendor inquiries for the Crawfordsville Strawberry Festival.", (0.73) [",:0.27] "initial_website": "https://crawfordsvillestrawberryfestival.com", "current_issue(0.90) [_page:0.07]": " Blocked from accessing thekitchennatchez.com",(0.62) ["}:0.38] "action_taken": "Attempted to navigate (0.10) [·access:0.90] to thek(0.92) [·festival:0.08]itchennatchez.com but was(0.49) [·encountered:0.49] blocked.",(0.32) ["}:0.68] }
**Progress:** {"Go to Crawfordsville Strawberry Festival website": "not(0.90) [attempt:0.06] finished", "Find contact email for vendor inquiries": "not finished"}Intention: Navigate to the Crawfordsville Strawberry Festival website**Action:** Navigate to(0.73) [·directly:0.27] https://crawfordsvillestrawberryfestival.com**<tool_call>**{"name": "computer_use", "arguments": {"action": "navigate", "url": "https://crawfordsvillestrawberryfestival.com"}} **</tool_call><|im_end|>**

**last step · mean entropy=0.0077**

**Memory:** {"task": "Find the contact email for vendor inquiries for the Crawfordsville Strawberry Festival.", "initial_website": "https://crawfordsvillestrawberryfestival.com", "current_issue": "Blocked from accessing the kitchennatchez.com", "action_taken": "Attempted to navigate to thekitchennatchez.com but was blocked.", "navigation_attempted": "https://crawfordsvillestrawberryfestival.com", "alternative_navigation_attempted": "https://www.crawfordsvillestrawberryfestival.com", "current_location": "Google search homepage", "search_query": "Crawfordsville Strawberry Festival vendor contact email", "current_page": "reCAPTCHA verification page", "new_navigation_attempted": "https://www.crawfordsville.in.gov", "search_bar_clicked": "finished", "search_query_entered": "Crawfordsville Strawberry Festival vendor contact", "search_results_found": "yes", "first_result_title": "Annual Community Events", "first_result_url": "https://www.crawfordsville.in.gov/428/Annual-Community-Events", "first_result_description": "Strawberry Festival - Second week of JuneThe Strawberry Festival has been a staple event in our community for over 50 years featuring a weekend filled with music, vendors, car shows, food trucks, and of course", "page_navigated_to": "Annual Community Events page", "content_viewed": "Information about USCA's Friends of Sugar Creek Canoe Race", "vendor_contact_info_found": "no", "strawberry_festival_link_found": "https://www.crawfordsville.in.gov/strawberryfestival", "strawberry_festival_description": "Strawberry Festival - Second week of JuneThe Strawberry Festival has been a staple event in our community for over 50 years featuring a weekend filled with music, vendors, car shows, food trucks, and of course", "current_page_navigated_to": "Crawfordsville Strawberry Festival website", "page_content": "Welcome message and festival dates (June 12-14, 2026)", "navigation_menu": "Home, History, Entertainment, Vendors, Events, Contact", "vendors_page_navigated_to": "finished", "vendors_page_content": "Sections for Arts and Crafts, Children's Area, Commercial Vendors, Food Vendors", "scroll_action_performed": "yes", "current_scroll_position": "Photos section", "contact_email_found": "no", "additional_scroll_action_performed": "yes", "current_scroll_position_after_second_scroll": "Social media and copyright section", "vendor_contact_info_still_not_found": "yes", "attempted_to_click_on_contact_link": "yes", "contact_page_not_loaded": "yes", "current_page_content": "Photos section and social media statistics", "need_to_return_to_contact_page": "yes", "went_back_to_previous_page": "yes", "current_page": "Crawfordsville Strawberry Festival homepage", "contact_link_visible": "yes", "attempted_to_click_on_contact_link_again": "yes", "contact_page_loaded": "yes", "contact_page_content": "Contact form with fields for name, email, and message", "contact_email_found": "no", "scrolled_down_contact_page": "yes", "individual(0.85) [contact:0.15]_contacts_section_visible": "yes", "vendor(0.88) [contact:0.12]_contact(0.77) [_emails:0.13 _email:0.08]_emails(0.44) [_email:0.56]_found": "yes(0.76) [no:0.22]", "food_vendors_email": "food@crawfordsvillestrawberryfest.com",(0.88) ["}:0.12] "arts_c(0.22) [_and:0.78]rafts_vendor_email": "dcookse(0.78) [se:0.22]strawberryfest@yahoo.com", "5(0.01) [chair:0.81 contact:0.14]k(0.38) [K:0.62]_run_email": "jthompson@nm.k12.in.us", "chairperson_email": "sfestqueen@gmail.com", "entertainment_email": "entertainment@crawfordsvillestrawberryfest.com"}**Progress:** {"Go to Crawfordsville Strawberry Festival website": "finished", "Find contact email for vendor inquiries": "not finished", "Navigate to Google search": "finished", "Type search query": "finished", "Facing reCAPTCHA": "not finished", "Navigate to Crawfordsville IN government website": "finished", "Click on search bar": "finished", "Type query into search bar": "finished", "Review search results": "finished",

"Click on first search result": "finished", "Scroll down page": "finished", "Click on Strawberry Festival website link": "finished", "Click on 'Vendors' link": "finished", "Scroll down vendors page": "finished", "Scroll down vendors page again": "finished", "Click on 'Contact' link": "finished", "Contact page not loaded": "yes", "Current page content": "Photos section and social media statistics", "Go back to previous page": "finished", "Current page": "Crawfordsville Strawberry Festival homepage", "Contact link visible": "yes", "Attempted to click on contact link again": "yes", "Contact page loaded": "yes", "Contact page content": "Contact form with fields for name, email, and message", "Contact email found": "no", "Scrolled down contact page": "yes", "Individual contacts section visible": "yes", "Vendor contact emails found": "yes", "Food vendors email": "food@crawfordsvillestrawberryfest.com", "Arts & Crafts(0.85) [·crafts:0.15] vendor email": "dcookseystrawberryfest@yahoo.com", "5K Run email": "jthompson@nm.k12.in.us", "Chairperson email": "sfestqueen@gmail.com", "Entertainment email": "entertainment@crawfordsvillestrawberryfest.com"}Intention: Provide(0.88) [·Find:0.07] the contact email for vendor(0.44) [·food:0.56] inquiries as the answer(0.02) [·task:0.92 ·final:0.06]
**Action:** The contact email for vendor inquiries is food@crawfordsvillestrawberryfest.com**<tool_call>**{"name": "computer_use", "arguments": {"action": "answer", "text": "The contact email for vendor inquiries is food@crawfordsvillestrawberryfest.com"}}**</tool_call><|im_end|>**

### C.4 PROMPTS

Both the Instruct and Thinking checkpoints share the same agent prompt template, the only configurable variation is the single rule line that controls whether `Memory` is updated additively (default) or compressively (ablation). Both variants are shown inline below, color-coded.

**Prompt: AsyncWebRL Agent**

--- MESSAGE 1: SYSTEM ---
You are a helpful assistant.

# Tools

You may call one or more functions to assist with the user query.

You are provided with function signatures within <tools></tools> XML tags:
<tools>
[computer_use_tool_def]
</tools>

For each function call, return a JSON object with function name and arguments within <tool_call></tool_call> XML tags:
<tool_call>
{"name": <function-name>, "arguments": <args-json-object>}
</tool_call>

# Response format

Response format for every step:
1) Memory: facts you would like to memorize for future actions in json format. Include the current step.
2) Progress: Decompose the task into subtasks and what has been finished so far with json format. Include progress of the current step.
3) Intention: clearly state which subtask you're working on at this step with the json key.
4) Action: a short sentence describing what to do in the UI to accomplish the next subtask.
5) A single <tool_call>...</tool_call> block containing only the JSON: {"name": <function-name>, "arguments": <args-json-object>}.

Rules:
- Output exactly in the order: Memory, Progress, Intention, Action, <tool_call>.
- You MUST use json format for the Memory and Progress parts.
- Example Task: "Search and compare the prices and locations of product 1 and product 2 on Amazon."
- Example of Memory json format: {"Price of product 1": "10.00", "Location of product 1": "10.00", "Price of product 2": "12.00"}.
- Example of Progress json format: {"Go to Amazon.com": "finished", "Search for price of product 1": "finished", "Search for location of product 1": "finished", "Search for price of product 2": "finished", "Search for location of product 2": "not finished", "Compare product 1 and product 2": "not finished"}.

– Example of Intention json key format: "Search for location of product 2".
- (default, additive memory prompt) You CANNOT modify previous Memory. Only append to it.
- (ablation, compressive memory prompt) Each step, you should COMPRESS previous memory by identifying only those relevant.
– You CAN modify Progress from previous conversation to further decompose the task and guide your next action.
– For example, if the previous assistant message specifies Progress: {"Go to Amazon.com": "finished", "Search for product 1": "finished", "Search for product 2": "not finished", "Compare product 1 and 2": "not finished"},
– You should further decompose "Search for product 1" and "Search for product 2" into "Search for price of product 1" and "Search for location of product 1", and "Search for price of product 2" and "Search for location of product 2".
– Do not output anything else outside those five parts.

––– MESSAGE 2: USER –––
Please generate the next action according to the UI screenshot and task.

Task: [task_description]

Initial website: [initial_website_url]

Generate the next action to complete the task.

[current_screenshot]

### C.5 HYPERPARAMETERS

Table 5 reports the hyperparameters used for the three main runs on the Instruct variant: GRPO with the $1/|\tau_i|$ normalizer, GRPO with the constant-$1/k$ fix, and RAFT++. The upper sub-table lists settings shared across all three runs; the lower sub-table lists the settings that differ.
RAFT++ uses a consumer batch size of 120 rather than 128 because we find that using 6 GPUs for training is more speed-optimal for RAFT++, while because we use a group size of 8 we naturally use a consumer batch size of 128 for GRPO. This is purely a scheduling choice; no other hyperparameter was retuned.
The Thinking-variant GRPO runs share the same prompt template (Section C.4) and the same loss, optimizer, and rollout settings as the Instruct GRPO runs; only the base model differs (Qwen3-VL-8B-Thinking) and, at runtime, the previous step's thinking tokens are hidden from history. The Thinking-variant RAFT++ run additionally uses a higher learning rate ($1 \times 10^{-5}$), a single update epoch per batch, and a relaxed off-policy lag cap of 4.

Table 5: **Hyperparameters for the three main Qwen3-VL-8B-Instruct runs.** Values are taken from each run's wandb config.

| *Settings common to all three runs* | |
|---|---|
| Base model | `Qwen/Qwen3-VL-8B-Instruct`, bfloat16 |
| Cluster | 2 nodes $\times$ 8 GPUs (B200) |
| Optimizer | AdamW ($\beta_1$=0.9, $\beta_2$=0.999, $\epsilon$=$10^{-8}$) |
| Learning rate | $5 \times 10^{-6}$ |
| LR scheduler | cosine, max steps 300, min ratio 0.1, warmup proportion 0.0167 |
| Weight decay | 0.01 |
| Gradient clipping | 1.0 |
| Update epochs per batch | 3 |
| Minibatches per update | 2 |
| Importance-sampling level | token |
| Off-policy correction | decoupled PPO, proximal logprobs recomputed |
| $\epsilon$-clip | 0.2 |
| Max off-policy lag ($\eta$) | 2 |
| KL coefficient (to reference) | 0 |
| Disable dropout | true |
| Gradient checkpointing | true |
| Rollout temperature | 1.0 |
| Rollout top-$p$ / top-$k$ / freq. penalty | 1.0 / $10^8$ / 0 (effectively disabled) |
| Max new tokens per step | 4096 |
| Inference engine | SGLang, flashinfer backend, context length 16,384 |
| Evaluator backbone | `GPT-4o` |
| SGLang `mem_fraction_static` | 0.75 |
| Max concurrent inference requests | 48 |
| WebGym prompt version | complete |
| Interaction mode | coordinates-only |
| History window | previous step only |
| Train horizons (easy, medium, hard) | $(10, 20, 30)$ |
| Test horizons (easy, medium, hard) | $(30, 50, 70)$ |
| Test-time decoding | greedy |

| Hyperparameter | GRPO (with LN) | GRPO (no LN) | RAFT++ |
|---|---|---|---|
| Loss (`actor_loss`) | `grpo` | `grpo` | `fbc` |
| Step-number normalizer | $1/\lvert\tau_i\rvert$ | constant $1/k$ with $k$=10 | $1/\lvert\tau_i\rvert$ |
| Group mean/std normalization | group, size 8 | group, size 8 | disabled |
| Dual-clip $c$ (Equation (2)) | 3.0 | 3.0 | disabled |
| Samples per task (group size $G$) | 8 | 8 | 1 |
| Consumer batch size | 128 | 128 | 120 |
| Filter steps with unchanged screenshot | no | no | yes |