CAPF: Guiding Search-Agent Rollouts with Credit-Attenuated Privileged Feedback

Anonymous ACL submission

Abstract

Recent LLM search agents use reinforcement learning with verifiable rewards (RLVR) to learn search-augmented reasoning from outcome rewards. On hard problems, these agents rarely sample end-to-end successful rollouts, leaving outcome-only RLVR with few positive-reward trajectories. We argue that improving learning on such problems requires additional guidance during training, and RLVR already contains verifier-side information that can provide it. This information can identify errors or omissions in the agent's submitted answer and guide revision within the rollout. We propose a training-time mechanism called **Credit-Attenuated Privileged Feedback** (CAPF), which makes this verifier-side information available through a Privileged Feedback call during training. CAPF lets the policy revise zero-reward attempts into positive-reward repair trajectories and attenuates credit for the feedback call and earlier actions to accommodate deployment without this call. Empirical research demonstrates that CAPF improves Qwen3-4B's average exact-match score from 44.7% under outcome-only RLVR to 48.5% on seven open-domain QA benchmarks.

1 Introduction

Search-augmented LLM agents answer open-domain questions by interleaving reasoning, search queries, retrieved evidence, and final-answer generation (Nakano et al., 2021; Yao et al., 2023; Li et al., 2025b). Recent work trains such agents with reinforcement learning from verifiable rewards (RLVR), where an automatic verifier scores only the final answer (Guo et al., 2025; Wen et al., 2025; Jin et al., 2025a; Chen et al., 2025). This outcome-only paradigm provides simple and scalable supervision when the agent samples enough successful rollouts. On hard problems, however, search agents rarely reach the correct answer within the rollout budget, so most attempts receive zero reward. Outcome-only RLVR therefore obtains little positive learning signal from the problems where additional guidance would be most useful.

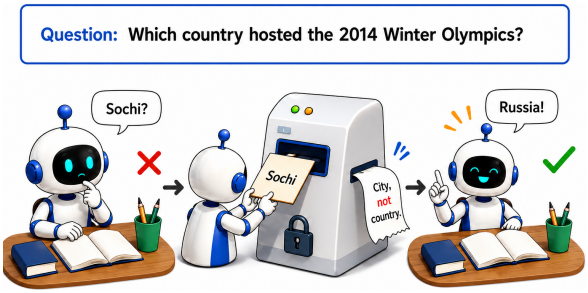

Figure 1: The agent uses Privileged Feedback to repair an uncertain candidate answer during training. Given the question "Which country hosted the 2014 Winter Olympics?", the agent first drafts "Sochi" but remains uncertain and calls Privileged Feedback. The feedback flags that "Sochi" is the host city rather than the requested country. The agent then revises the final answer to "Russia".

The verifier-side information behind verifiable rewards can help recover this missing signal during training. Because the training environment has access to reference answers, it can compare the agent's submitted answer with the references and return a short feedback message. The feedback can point out what is wrong or missing, such as an incorrect answer type, a missing constraint, or unsupported evidence. This gives the agent a chance to revise within the same rollout instead of ending the attempt with zero reward.

We call this verifier-generated feedback **Privileged Feedback**. Figure 1 illustrates how this feedback can turn a zero-reward attempt into a repair trajectory. The agent first submits "Sochi" for a question that asks for the host

country of the 2014 Winter Olympics. Privileged Feedback identifies the answer-type error and allows the agent to revise within the rollout. These repaired trajectories provide learning signal that outcome-only RLVR would miss, but they are not ordinary end-to-end successes. Their post-feedback suffixes may contain useful repair behavior, while their final success passes through privileged information that will not be available at deployment. The question is how to use these successes as learning signal while preventing the policy from relying on Privileged Feedback.

We propose **Credit-Attenuated Privileged Feedback** (CAPF) to address this problem. CAPF adds a Privileged Feedback call to the training action space and removes this call at deployment. It computes turn-level returns with a Privileged Feedback retention factor. When credit propagates backward across a Privileged Feedback call, CAPF attenuates the return for the call and earlier actions while preserving full credit for actions after the call. This gives full credit to post-feedback repair actions while giving weaker credit to the feedback call and earlier actions. Evaluated without Privileged Feedback, CAPF improves Qwen3-4B's average exact-match score on seven open-domain QA benchmarks from 44.7% under outcome-only RLVR to 48.5%.

Our main contributions are as follows:

1. We introduce **Privileged Feedback** for search-agent RLVR, using verifier-side information during training to create repair trajectories from attempts that would otherwise receive zero reward.
2. We propose **Credit-Attenuated Privileged Feedback** (CAPF), which adds a Privileged Feedback call to the training action space and attenuates return when credit crosses such calls.
3. We show that CAPF improves deployed search-agent performance after Privileged Feedback is removed.

2 Related Work

CAPF builds on search-agent RLVR but uses verifier-side feedback to recover learning signal from otherwise zero-reward attempts. We compare this design with prior work on additional supervision for sparse rewards, learning with privileged information, and reward shaping.

Search-agent reinforcement learning. Search-augmented language agents interleave reasoning with actions that retrieve and use external evidence (Nakano et al., 2021; Yao et al., 2023; Li et al., 2025b). Recent RLVR systems optimize this interaction loop from final correctness, improving query selection, evidence use, and long-horizon search behavior (Jin et al., 2025a; Chen et al., 2025; Zheng et al., 2025a; Gao et al., 2025; Zheng et al., 2025b; Lu et al., 2026). These systems train policies to use actions that remain available at test, such as search, reasoning, and final-answer generation. CAPF instead augments the training action space with a Privileged Feedback call. When the policy issues this call, the environment returns a feedback message as an observation. At deployment, the call is removed. This shifts the question from learning to use a persistent search action to learning from repair trajectories produced by training-time calls.

Additional supervision for sparse rewards. One line of work addresses sparse outcome rewards by adding supervision beyond final correctness. Process supervision and step-level verification score intermediate reasoning steps (Uesato et al., 2022; Lightman et al., 2024), while search-agent methods add denser rewards (Luo et al., 2025), turn-level feedback (Xie et al., 2026), or instructive feedback (Li et al., 2025a; Shi et al., 2025) to guide exploration and evidence use. These methods can make sparse-reward learning easier, but they often require additional annotations, reward design, or auxiliary evaluators. CAPF instead reuses verifier-side information already present in RLVR to create repair trajectories without additional process annotations.

Learning with privileged information. Learning with privileged information studies settings where training examples contain extra information unavailable at test time (Vapnik and Vashist, 2009; Pechyony and Vapnik, 2010). Recent LLM work similarly uses privileged teacher feedback, oracle guidance, or verified traces to improve exploration, supervision, or distillation (Choudhury and Sodhi, 2025; Qu et al., 2025). CAPF also falls within this set-

ting, but does not use privileged information as an external teacher signal or extra label. Instead, CAPF lets the policy issue Privileged Feedback calls inside the rollout rather than supplying privileged information as an external training signal. Each call is recorded in the trajectory and can be accounted for in credit assignment.

Reward shaping and credit assignment. Reward shaping modifies the reward function to guide learning (Ng et al., 1999), and transition-dependent objectives study returns that depend on transition structure (White, 2017; Pitis, 2019). CAPF is related because it changes how final success is assigned to actions in a trajectory. It differs by leaving the final-answer verifier reward unchanged. Rather than adding dense rewards or imposing a hand-coded cost for using Privileged Feedback, CAPF attenuates return only when final success propagates backward across a Privileged Feedback call. This keeps repaired trajectories useful for learning while reducing credit to behavior before the training-time feedback.

3 Method

CAPF adds a Privileged Feedback call to the training action space and removes this call at deployment. When the policy issues this call with a candidate answer, the environment returns a feedback message as an observation. The policy can then revise within the rollout, turning some otherwise zero-reward attempts into positive-reward repair trajectories. Since each Privileged Feedback call is visible in the trajectory, CAPF can attenuate return for the call and earlier actions while preserving full credit for post-feedback actions.

3.1 Search-Agent RLVR with Deployment Constraints

Each instance contains a question x and a reference answer $y^\star$. At each nonterminal step, the policy selects an action a_t. The search environment executes this action and returns an observation o_t. Thus, the policy conditions on the history $h_t = (x, a_{<t}, o_{<t})$.

The base action space $\mathcal{T}_{\text{base}}$ contains ordinary search-agent actions and a final-answer action $\textsc{Final}(y)$. When the policy emits $\textsc{Final}(y_T)$ at step T, the episode terminates. An exact-answer verifier then compares y_T with $y^\star$ and returns:

$$R(\tau) = \mathsf{V}(y_T, y^\star) \in \{0, 1\}. \tag{1}$$

All nonterminal task rewards are zero. When the policy does not emit a final answer within the rollout budget, we set $R(\tau) = 0$.

Training and deployment differ by one action:

$$\mathcal{T}_{\text{train}} = \mathcal{T}_{\text{base}} \cup \{a_{\text{PF}}\}, \quad \mathcal{T}_{\text{test}} = \mathcal{T}_{\text{base}} \tag{2}$$

The augmented action a_{PF} is used only during training rollout collection. All reported benchmark results use $\mathcal{T}_{\text{test}}$, with the Privileged Feedback call absent from the action space.

3.2 Privileged Feedback Call

A Privileged Feedback call has the form:

$$a_t = a_{\text{PF}}(\hat{y}_t)\,, \tag{3}$$

where $\hat{y}_t$ denotes the policy's current candidate answer. This nonterminal action does not directly alter the task reward. Instead, the environment compares $\hat{y}_t$ with the reference answer $y^\star$, obtains the verifier label $\mathsf{V}(\hat{y}_t, y^\star)$, and appends a concise feedback message as an environment observation. Subsequent policy actions condition on the updated history.

This rollout structure separates policy text from environment text. The policy produces search actions, Privileged Feedback calls, and final answers. The environment returns retrieval results and feedback messages. Tokens returned by the environment are masked and therefore do not contribute to the policy log probability or the training loss. The reference answer and verifier label are used only inside the environment-side feedback generator. Since the feedback message relies on verifier-side information unavailable at deployment, we still regard it as privileged information, even though prompt constraints and exact string redaction limit direct copying from the reference answer.

3.3 Credit-Attenuated Privileged Feedback

Successful repair trajectories can recover learning signal from attempts that outcome-only RLVR would score as zero. Their post-feedback suffixes may contain transferable behavior, such as issuing a more effective query, inspecting

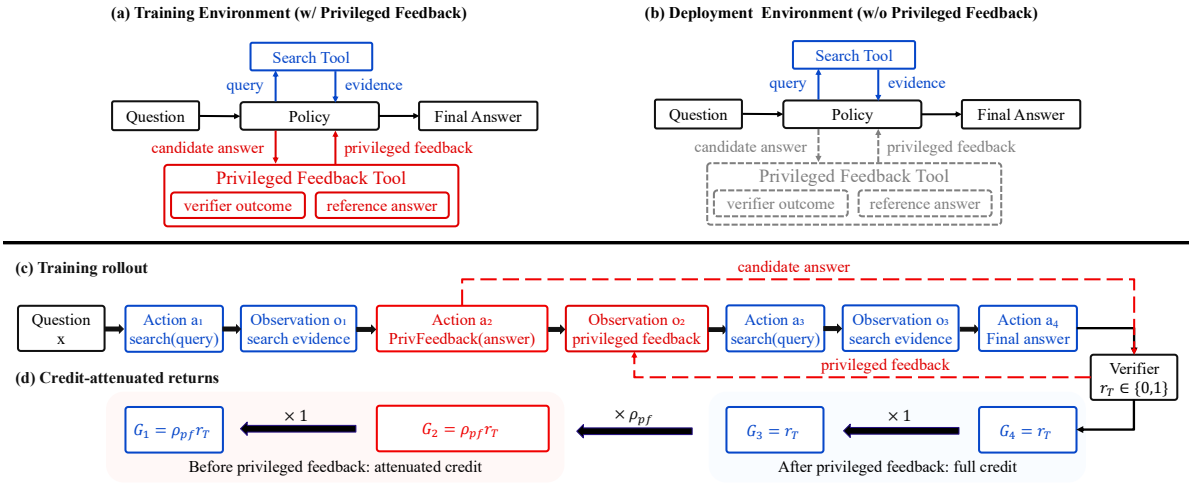

Figure 2: CAPF adds a Privileged Feedback call to the training action space and removes this call at deployment. The final-answer verifier reward remains unchanged. CAPF attenuates return only when final success propagates backward across a Privileged Feedback call. (a) Training environment with Privileged Feedback; (b) deployment environment without Privileged Feedback; (c) training rollout with Privileged Feedback; (d) credit-attenuated returns.

more relevant evidence, or correcting the answer type. The feedback call and earlier actions should not receive the same credit as unassisted successful behavior, since final success has passed through an environment observation unavailable at deployment. Ordinary RLVR does not make this distinction and assigns the same final reward signal to all policy actions in a successful trajectory.

CAPF implements this distinction by attenuating return only when backward credit crosses a Privileged Feedback call. Let T be the index of the final-answer action. For a policy decision at step t, define

$$N_t^{\mathrm{PF}}(\tau) = \sum_{s=t}^{T-1} \mathbb{I}[a_s = a_{\mathrm{PF}}(\cdot)]\,, \tag{4}$$

as the number of Privileged Feedback calls from step t through step $T-1$, including a_t itself when it is a Privileged Feedback call. By convention, $N_T^{\mathrm{PF}}(\tau) = 0$.

For a retention factor $0 < \rho_{\mathrm{PF}} \leq 1$, CAPF defines the return for the action taken at step t as

$$G_t^{\mathrm{CAPF}}(\tau) = R(\tau)\,\rho_{\mathrm{PF}}^{N_t^{\mathrm{PF}}(\tau)}. \tag{5}$$

Thus, actions after the last Privileged Feedback call receive the full verifier reward. The feedback call and earlier actions receive attenuated credit. If one Privileged Feedback call lies between action a_t and the final answer, including a_t itself, the action receives $\rho_{\mathrm{PF}} R(\tau)$. If k such calls lie between a_t and the final answer, the action receives $\rho_{\mathrm{PF}}^k R(\tau)$.

Algorithm 1 CAPF return computation

Require: Trajectory actions $a_0, \ldots, a_T$, final-answer verifier reward $R(\tau)$, retention factor ρ_{PF}

```
g ← R(τ)
G_T ← g
for t = T − 1, ..., 0 do
    if a_t = a_PF(·) then
        g ← ρ_PF · g
    end if
    G_t ← g
end for
return G_0, ..., G_T
```

3.4 CAPF Policy Gradient Objective

We optimize the policy with REINFORCE++ (Williams, 1992; Sutton et al., 1999; Hu et al., 2025a). CAPF modifies only the policy-gradient target and keeps the optimizer, rollout budgets, final-answer verifier, baseline subtraction, and advantage normalization the same as in the outcome-only run. For each policy action, CAPF replaces the ordinary return based on $R(\tau)$ with G_t^{CAPF} from Eq. (5).

The return targets G_t^{CAPF} can be computed with a single backward pass over the trajectory. Starting from the reward from the final-answer verifier, the pass multiplies the running return by ρ_{PF} each time it crosses a Privileged Feedback call. Algorithm 1 summarizes this computation.

Let $\mathcal{A}_\pi(\tau)$ denote the set of policy-generated action indices in the trajectory. For text actions, $\log \pi_\theta(a_t \mid h_t)$ is the sum of token log probabilities over the generated action span. The policy-gradient loss is

$$\mathcal{L}_{\text{pg}}(\theta) = -\sum_{t \in \mathcal{A}_\pi(\tau)} \widehat{A}_t \log \pi_\theta(a_t \mid h_t). \quad (6)$$

The advantage $\widehat{A}_t$ is computed from G_t^{CAPF} using the same baseline subtraction and advantage normalization as the outcome-only run. Environment-generated text, including retrieval results and feedback messages, is outside $\mathcal{A}_\pi(\tau)$. It is not assigned log probability and is masked from the loss. Malformed actions and parser failures follow the same environment rules in outcome-only RL and CAPF. Appendix C gives the prompts, tool schema for the Privileged Feedback call, redaction rule, and rollout traces.

3.5 Objective Decomposition

To analyze how the CAPF objective uses Privileged Feedback, we decompose trajectories by the number of Privileged Feedback calls. Let $S_k(\pi)$ denote the probability that the policy succeeds after using Privileged Feedback exactly k times:

$$S_k(\pi) = \Pr_{\tau \sim p^{\pi}_{\mathcal{T}_{\text{train}}}} \left[R(\tau) = 1,\, N_0^{\text{PF}}(\tau) = k \right] \quad (7)$$

The CAPF-weighted start-state success can then be written as

$$\begin{aligned} J^{\text{start}}_{\rho_{\text{PF}}}(\pi) &= \sum_{k \geq 0} \rho^k_{\text{PF}} S_k(\pi) \\ &= S_0(\pi) + \underbrace{\sum_{k \geq 1} \rho^k_{\text{PF}} S_k(\pi)}_{B_{\rho_{\text{PF}}}(\pi)}. \end{aligned} \quad (8)$$

Here, S_0 is success without Privileged Feedback, while $B_{\rho_{\text{PF}}}$ is the attenuated contribution from feedback-assisted successes.

This decomposition is only an analysis tool for measuring how Privileged Feedback contributes to the learning signal during training. The terms with $k \geq 1$ depend on trajectories that use Privileged Feedback and cannot be replayed unchanged under $\mathcal{T}_{\text{test}}$. CAPF still keeps these trajectories because their post-feedback suffixes may contain repair behaviors that transfer to no-feedback execution.

4 Experiments

We evaluate CAPF under deployment constraints. All reported benchmark results are measured after removing the Privileged Feedback call. Training-environment probes are used only to analyze how repair-based learning transfers to deployment execution.

Benchmarks. We evaluate final-answer exact match (EM) on seven open-domain QA benchmarks: NQ (Kwiatkowski et al., 2019), TriviaQA (Joshi et al., 2017), PopQA (Mallen et al., 2023), HotpotQA (Yang et al., 2018), 2WikiMultiHopQA (Ho et al., 2020), MuSiQue (Trivedi et al., 2022), and Bamboogle (Press et al., 2023). Avg. denotes the benchmark-level macro-average. Appendix A gives the splits, example counts, and answer parsing details.

Training setup. All 4B attribution runs train `Qwen3-4B-Thinking-2507` (Qwen Team, 2025) with OpenRLHF `reinforce++` (Hu et al., 2025b,a) on the same training data used by Search-R1 (Jin et al., 2025a,b). The feedback generator is a fixed `Qwen3-30B-A3B-Instruct-2507`. Relative to outcome-only RL, CAPF adds the Privileged Feedback call and feedback instructions during training, and uses the return in Eq. (5). The deployed policy cannot call Privileged Feedback. Appendix B provides the retrieval, optimization, and rollout details.

4.1 CAPF Improves Deployment Performance

Table 1 reports deployment performance after removing the Privileged Feedback call from the action space. Outcome-only RL serves as a strong baseline relative to direct inference, increasing average EM from 35.6% to 44.7%. CAPF with $\rho_{\text{PF}} = 0.8$ reaches 48.5%, exceeding this baseline by +3.8 points and the unattenuated Privileged Feedback variant by +2.4 points. The improvement is consistent across benchmark families, with average EM rising from 48.9% to 52.5% on single-hop tasks and from 41.6% to 45.4% on multi-hop tasks. The remaining CAPF rows in Table 1 vary the retention factor and are analyzed in Section 4.3.

Table 1: Deployment evaluation with `Qwen3-4B-Thinking-2507` after removing the Privileged Feedback call. The upper block gives the main attribution comparison; the lower block reports retention factor sensitivity. All scores are exact-match (EM) values in %; Avg. is the benchmark-level macro-average.

Method	Single-hop QA			Multi-hop QA				Avg.
	NQ	TriviaQA	PopQA	HotpotQA	2Wiki	MuSiQue	Bamboogle	
Direct inference	35.5	64.4	37.7	33.2	37.6	8.3	32.8	35.6
Outcome-only RL	40.5	65.8	40.4	43.0	51.4	19.3	52.8	44.7
CAPF ($\rho_{\text{PF}} = 1.0$)	41.4	67.8	42.5	45.7	49.5	21.9	53.6	46.1
CAPF ($\rho_{\text{PF}} = 0.8$)	**43.6**	**69.4**	**44.5**	**48.5**	53.6	**25.2**	**54.4**	**48.5**
CAPF retention factor sensitivity								
CAPF ($\rho_{\text{PF}} = 0.5$)	41.6	67.9	43.5	46.8	**53.9**	23.2	**54.4**	47.3
CAPF ($\rho_{\text{PF}} = 0.6$)	42.5	68.2	44.0	48.0	53.6	23.9	53.6	47.7

Figure 3: Mechanism probe for CAPF with $\rho_{\text{PF}} = 0.8$ on a fixed 200-example set. Training-environment success is decomposed into S_0 and $B_{\rho_{\text{PF}}}$ from Eq. (8). Deployment EM is measured on the same probe without Privileged Feedback.

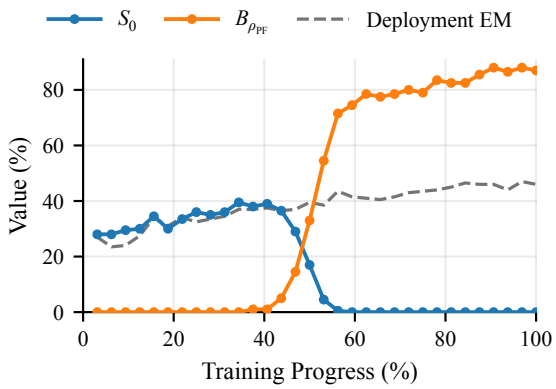

Figure 4: Mechanism probe for unattenuated CAPF ($\rho_{\text{PF}} = 1.0$) on the same fixed 200-example set. Training-environment success is decomposed into zero-feedback success S_0 and feedback-assisted contribution $B_{\rho_{\text{PF}}}$ from Eq. (8).

4.2 Feedback-Assisted Repairs Transfer to No-Feedback Execution

Benchmark results show that CAPF improves deployment performance, but they do not explain where the gain comes from. We use a fixed 200-example probe to relate training-environment success to deployment execution. We sample the probe from the RL training data before training, keep it fixed across runs, and use it only for analysis. At each checkpoint, we evaluate the same policy in the training environment, where the Privileged Feedback call is available, and in the deployment environment, where this call is removed. We report S_0 for successes without any feedback call and compute $B_{\rho_{\text{PF}}} = \sum_{k \geq 1} \rho_{\text{PF}}^k S_k$ for attenuated feedback-assisted successes. Each S_k is estimated from rollout counts on the training-environment probe.

For CAPF with $\rho_{\text{PF}} = 0.8$, Figure 3 shows an early dependence on Privileged Feedback. At step 330, training-environment EM reaches 74.0%, while S_0 is only 24.0%. Later in training, the attenuated feedback-assisted term declines while both S_0 and deployment EM rise. By step 470, deployment probe EM reaches 51.0%, S_0 reaches 50.5%, average Privileged Feedback calls fall to 0.045 per rollout, and only 4.0% of probe rollouts invoke Privileged Feedback. This pattern suggests that feedback-assisted repairs are being converted into behavior executable without Privileged Feedback calls.

4.3 Attenuating Feedback Credit Improves Deployment Transfer

The comparison between $\rho_{\text{PF}} = 1.0$ and attenuated CAPF variants isolates the effect of credit attenuation. All variants use the same Privileged Feedback call during training and remove it at evaluation. With $\rho_{\text{PF}} = 1.0$, CAPF treats repaired trajectories as ordinary successes. This

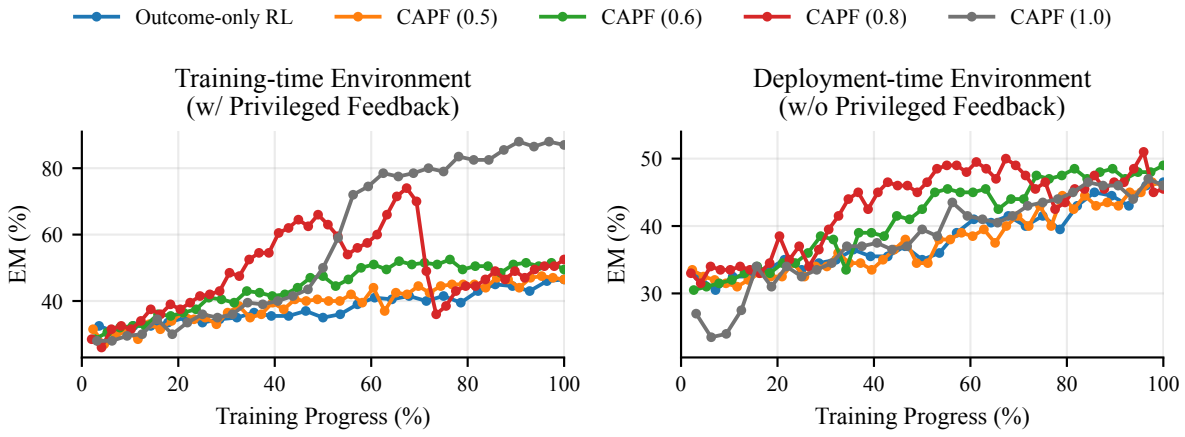

Figure 5: Probe EM at matched checkpoints in the training and deployment environments. Left: training environment, where the Privileged Feedback call is available to CAPF variants. Right: deployment environment, where the Privileged Feedback call is removed. The unattenuated $\rho_{\mathrm{PF}} = 1.0$ setting reaches high training-environment EM but not the best deployment EM.

setting improves average EM from 44.7% to 46.1% over outcome-only RL, showing that feedback-assisted trajectories provide a useful learning signal. However, it underperforms all tested attenuated settings and trails $\rho_{\mathrm{PF}} = 0.8$ by 2.4 percentage points.

Figure 5 gives the corresponding probe view. The unattenuated policy reaches high training-environment probe EM by continuing to call Privileged Feedback, but its deployment probe EM remains below the $\rho_{\mathrm{PF}} = 0.8$ policy after this call is removed. Figure 4 applies the same fixed-probe decomposition as Figure 3 to this unattenuated run. Therefore, increasing training-environment success is not sufficient.

This gap reflects the credit problem CAPF is designed to address. Post-feedback actions may contain transferable repair behavior, such as refined queries, better evidence selection, or repaired answers. The feedback call and earlier actions should not receive the same credit as unassisted successful behavior because Privileged Feedback is absent at deployment. CAPF therefore attenuates return whenever credit crosses a Privileged Feedback call, preserving learning signal from repair trajectories while reducing feedback-dependent credit.

4.4 Probe Accounting for Tool Use and Search Budget

Table 2 reports checkpoint-level tool-use and search-budget accounting on the same 200-example probe. The accounting checks whether

Table 2: Fixed-probe accounting at selected checkpoints. Search and PF denote average calls per rollout. Use % is the fraction of training-environment rollouts with at least one Privileged Feedback call. Deployment metrics exclude Privileged Feedback.

Method	Train Env.				Deploy Env.	
	EM (%)	Search	PF	Use (%)	EM (%)	Search
Outcome-only RL	46.5	2.45	0.000	0.0	46.5	2.45
CAPF ($\rho_{\mathrm{PF}} = 0.5$)	47.0	3.10	0.000	0.0	46.5	2.17
CAPF ($\rho_{\mathrm{PF}} = 0.6$)	49.5	10.41	0.130	3.5	49.0	4.18
CAPF ($\rho_{\mathrm{PF}} = 0.8$)	50.5	2.87	0.045	4.0	**51.0**	1.98
CAPF ($\rho_{\mathrm{PF}} = 1.0$)	**88.0**	7.51	6.455	100.0	47.0	5.10

the best deployment-probe result can be explained by larger search budgets or residual Privileged Feedback use. CAPF with $\rho_{\mathrm{PF}} = 0.8$ reaches 51.0% deployment probe EM with 1.98 search calls per rollout, compared with 46.5% EM and 2.45 search calls for outcome-only RL on the same probe. Its training-environment Privileged Feedback use is also low, averaging 0.045 calls per rollout, with only 4.0% of rollouts invoking Privileged Feedback. The unattenuated $\rho_{\mathrm{PF}} = 1.0$ row provides a contrast point. It reaches 88.0% training-environment EM with a Privileged Feedback call in every probe rollout, but drops to 47.0% deployment-probe EM after these calls are removed. Thus, the strongest deployment-probe result is not explained by spending more search calls or by retaining the ability to call Privileged Feedback.

Figure 6 shows the corresponding full fixed-probe trajectories for Privileged Feedback calls and search calls. It complements Table 2, which reports selected checkpoints only. The curves

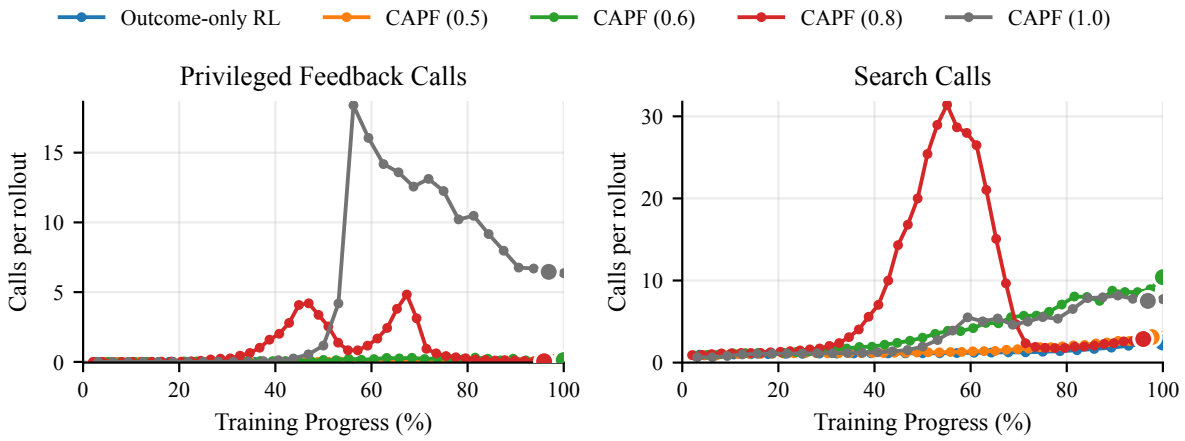

Figure 6: Training environment overhead on the fixed probe. Left: Privileged Feedback calls per rollout. Right: search calls per rollout. Enlarged markers indicate the checkpoints summarized in Table 2.

Table 3: Comparison with recent search-agent RL systems on the seven-benchmark open-domain QA suite. All scores are exact-match (EM) values in %; Avg. denotes the benchmark-level macro-average.

Method	Base model	Single-hop QA			Multi-hop QA				Avg.
		NQ	TriviaQA	PopQA	HotpotQA	2Wiki	MuSiQue	Bamboogle	
Search-o1-7B (Li et al., 2025b)	Qwen2.5-7B-Instruct	15.1	44.3	13.1	18.7	17.6	5.8	29.6	20.6
Search-R1-7B (Jin et al., 2025a)	Qwen2.5-7B-Instruct	39.3	61.0	39.7	37.0	41.4	14.6	36.8	38.5
ZeroSearch-7B (Sun et al., 2025)	Qwen2.5-7B-Instruct	43.6	65.2	48.8	34.6	35.2	18.4	27.8	39.1
TIPS-7B (Xie et al., 2026)	Qwen2.5-7B-Instruct	43.4	64.3	44.5	43.0	43.0	17.1	36.8	41.7
ReSeek-7B (Li et al., 2025a)	Qwen2.5-7B-Instruct	46.9	64.0	50.1	38.9	38.2	18.5	39.2	42.3
CAPF-7B ($\rho_{\text{PF}} = 0.8$)	Qwen2.5-7B-Instruct	44.8	63.6	43.6	41.7	43.6	19.5	44.0	43.0

distinguish two forms of overhead. Privileged Feedback calls measure reliance on the call removed at deployment. Search calls measure whether deployment gains coincide with larger retrieval budgets. The unattenuated $\rho_{\text{PF}} = 1.0$ run shows sustained Privileged Feedback use. The $\rho_{\text{PF}} = 0.6$ run shows higher search use at the selected checkpoint. The $\rho_{\text{PF}} = 0.8$ run ends with low Privileged Feedback use and low deployment search use.

4.5 Comparison with Recent Search-Agent RL Systems

Table 3 compares CAPF with recent search-agent RL systems on the same seven-benchmark open-domain QA suite. CAPF-7B uses Qwen2.5-7B-Instruct and follows the common Search-R1-style setup. After removing Privileged Feedback, CAPF-7B achieves 43.0% average EM, the highest macro-average among the listed Qwen2.5-7B systems. It also obtains the strongest multi-hop results on 2Wiki, MuSiQue, and Bamboogle. Several baselines remain stronger on some single-hop tasks and HotpotQA, so CAPF does not dominate every dataset. Overall, CAPF remains competitive under deployment constraints.

5 Conclusion

We develop **Credit-Attenuated Privileged Feedback** (CAPF), a training-time mechanism for search-agent RLVR with verifier-side feedback. It adds a Privileged Feedback call during training and attenuates credit assigned to the feedback call and earlier actions, allowing repaired successes to provide a useful but attenuated learning signal. Extensive experiments demonstrate that CAPF improves Qwen3-4B's average exact-match score from 44.7% under outcome-only RLVR to 48.5% across seven open-domain QA benchmarks without Privileged Feedback. The results indicate that training-time verifier feedback can improve test-time search agents when its credit is properly controlled.

Limitations

Our experiments are limited to search-augmented open-domain QA with exact-match evaluation. This setting provides automatic ver-

ifiers and relatively compact reference answers, making it suitable for studying Privileged Feedback. We plan to explore CAPF's performance on tasks with longer outputs or higher subjective correctness in the future, as such tasks require reliable verifiers, feedback generators, and their tool schemas.

Another limitation is the potential leakage of Privileged Feedback. Since Privileged Feedback is generated with access to the hidden reference answer, it may reveal answer-specific information even when it does not copy the reference string directly. A more systematic measurement of such leakage is left for future work.

References

Mingyang Chen, Linzhuang Sun, Tianpeng Li, Haoze Sun, Yijie Zhou, Chenzheng Zhu, Haofen Wang, Jeff Z. Pan, Wen Zhang, Huajun Chen, Fan Yang, Zenan Zhou, and Weipeng Chen. 2025. ReSearch: Learning to reason with search for LLMs via reinforcement learning. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Sanjiban Choudhury and Paloma Sodhi. 2025. Better than your teacher: LLM agents that learn from privileged AI feedback. In *The Thirteenth International Conference on Learning Representations*.

Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2024. The Faiss library. *Preprint*, arXiv:2401.08281.

Jiaxuan Gao, Wei Fu, Minyang Xie, Shusheng Xu, Chuyi He, Zhiyu Mei, Banghua Zhu, and Yi Wu. 2025. Beyond ten turns: Unlocking long-horizon agentic search with large-scale asynchronous RL. *Preprint*, arXiv:2508.07976.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, and 175 others. 2025. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 645(8081):633–638.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. 2020. Constructing a multi-hop QA dataset for comprehensive evaluation of reasoning steps. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 6609–6625, Barcelona, Spain (Online). International Committee on Computational Linguistics.

Jian Hu, Jason Klein Liu, Haotian Xu, and Wei Shen. 2025a. REINFORCE++: Stabilizing critic-free policy optimization with global advantage normalization. *Preprint*, arXiv:2501.03262.

Jian Hu, Xibin Wu, Wei Shen, Jason Klein Liu, Weixun Wang, Songlin Jiang, Haoran Wang, Hao Chen, Bin Chen, Wenkai Fang, Xianyu, Yu Cao, Haotian Xu, and Yiming Liu. 2025b. OpenRLHF: A ray-based easy-to-use, scalable and high-performance RLHF framework. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 656–666, Suzhou, China. Association for Computational Linguistics.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan O. Arik, Dong Wang, Hamed Zamani, and Jiawei Han. 2025a. Search-R1: Training LLMs to reason and leverage search engines with reinforcement learning. In *Proceedings of the Second Conference on Language Modeling*.

Jiajie Jin, Yutao Zhu, Zhicheng Dou, Guanting Dong, Xinyu Yang, Chenghao Zhang, Tong Zhao, Zhao Yang, and Ji-Rong Wen. 2025b. FlashRAG: A modular toolkit for efficient retrieval-augmented generation research. In *Companion Proceedings of the ACM on Web Conference 2025*, pages 737–740. ACM.

Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. 2017. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1601–1611, Vancouver, Canada. Association for Computational Linguistics.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. 2019. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:452–466.

Shiyu Li, Yang Tang, Yifan Wang, Peiming Li, and Xi Chen. 2025a. ReSeek: A self-correcting framework for search agents with instructive rewards. *Preprint*, arXiv:2510.00568.

Xiaoxi Li, Guanting Dong, Jiajie Jin, Yuyao Zhang, Yujia Zhou, Yutao Zhu, Peitian Zhang, and Zhicheng Dou. 2025b. Search-o1: Agentic search-enhanced large reasoning models. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 5420–5438,

Suzhou, China. Association for Computational Linguistics.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2024. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*.

Hongliang Lu, Yuhang Wen, Pengyu Cheng, Ruijin Ding, Jiaqi Guo, Haotian Xu, Chutian Wang, Haonan Chen, Xiaoxi Jiang, and Guanjun Jiang. 2026. Search self-play: Pushing the frontier of agent capability without supervision. In *The Fourteenth International Conference on Learning Representations*.

Kun Luo, Hongjin Qian, Zheng Liu, Ziyi Xia, Shitao Xiao, Siqi Bao, Jun Zhao, and Kang Liu. 2025. InfoFlow: Reinforcing search agent via reward density optimization. *Preprint*, arXiv:2510.26575.

Alex Mallen, Akari Asai, Victor Zhong, Rajarshi Das, Daniel Khashabi, and Hannaneh Hajishirzi. 2023. When not to trust language models: Investigating effectiveness of parametric and non-parametric memories. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9802–9822, Toronto, Canada. Association for Computational Linguistics.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, Xu Jiang, Karl Cobbe, Tyna Eloundou, Gretchen Krueger, Kevin Button, Matthew Knight, Benjamin Chess, and John Schulman. 2021. WebGPT: Browser-assisted question-answering with human feedback. *Preprint*, arXiv:2112.09332.

Andrew Y. Ng, Daishi Harada, and Stuart Russell. 1999. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning*, pages 278–287.

Dmitry Pechyony and Vladimir Vapnik. 2010. On the theory of learning with privileged information. In *Advances in Neural Information Processing Systems*, volume 23, pages 1894–1902. Curran Associates, Inc.

Silviu Pitis. 2019. Rethinking the discount factor in reinforcement learning: A decision theoretic approach. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7949–7956.

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah Smith, and Mike Lewis. 2023. Measuring and narrowing the compositionality gap in language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 5687–5711, Singapore. Association for Computational Linguistics.

Yuxiao Qu, Amrith Setlur, Virginia Smith, Ruslan Salakhutdinov, and Aviral Kumar. 2025. Learning to reason on hard problems with privileged on-policy exploration. In *The 5th Workshop on Mathematical Reasoning and AI at NeurIPS 2025*.

Qwen Team. 2024. Qwen2.5 technical report. *Preprint*, arXiv:2412.15115.

Qwen Team. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

Yaorui Shi, Sihang Li, Chang Wu, Zhiyuan Liu, Junfeng Fang, Hengxing Cai, An Zhang, and Xiang Wang. 2025. Search and refine during think: Facilitating knowledge refinement for improved retrieval-augmented reasoning. In *Advances in Neural Information Processing Systems*, volume 38.

Hao Sun, Zile Qiao, Jiayan Guo, Xuanbo Fan, Yingyan Hou, Yong Jiang, Pengjun Xie, Yan Zhang, Fei Huang, and Jingren Zhou. 2025. ZeroSearch: Incentivize the search capability of LLMs without searching. *Preprint*, arXiv:2505.04588.

Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. 1999. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems*, volume 12, pages 1057–1063. MIT Press.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022. MuSiQue: Multihop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554.

Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. 2022. Solving math word problems with process- and outcome-based feedback. *Preprint*, arXiv:2211.14275.

Vladimir Vapnik and Akshay Vashist. 2009. A new learning paradigm: Learning using privileged information. *Neural Networks*, 22(5–6):544–557.

Liang Wang, Nan Yang, Xiaolong Huang, Binxing Jiao, Linjun Yang, Daxin Jiang, Rangan Majumder, and Furu Wei. 2022. Text embeddings by weakly-supervised contrastive pre-training. *arXiv preprint arXiv:2212.03533*.

Xumeng Wen, Zihan Liu, Shun Zheng, Shengyu Ye, Zhirong Wu, Yang Wang, Zhijian Xu, Xiao Liang, Junjie Li, Ziming Miao, Jiang Bian,

and Mao Yang. 2025. Reinforcement learning with verifiable rewards implicitly incentivizes correct reasoning in base LLMs. *Preprint*, arXiv:2506.14245.

Martha White. 2017. Unifying task specification in reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning*, pages 3742–3750.

Ronald J. Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3–4):229–256.

Yutao Xie, Nathaniel Thomas, Nicklas Hansen, Yang Fu, Li Erran Li, and Xiaolong Wang. 2026. TIPS: Turn-level information-potential reward shaping for search-augmented LLMs. In *The Fourteenth International Conference on Learning Representations*.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380, Brussels, Belgium. Association for Computational Linguistics.

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

Xuhui Zheng, Kang An, Ziliang Wang, Yuhang Wang, and Yichao Wu. 2025a. StepSearch: Igniting LLMs search ability via step-wise proximal policy optimization. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 21805–21830, Suzhou, China. Association for Computational Linguistics.

Yuxiang Zheng, Dayuan Fu, Xiangkun Hu, Xiaojie Cai, Lyumanshan Ye, Pengrui Lu, and Pengfei Liu. 2025b. DeepResearcher: Scaling deep research via reinforcement learning in real-world environments. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 414–431, Suzhou, China. Association for Computational Linguistics.

A Data and Benchmarks

A.1 Training Data

All runs use `PeterJinGo/nq_hotpotqa_train`, the same training set used by Search-R1 (Jin et al., 2025a). The dataset is formed from the NQ and HotpotQA training splits in FlashRAG (Jin et al., 2025b). It contains 169,615 question–answer instances in total (79,168 from NQ and 90,447 from HotpotQA). We use the same data sampling script for outcome-only RL and all CAPF variants. No examples from the evaluation splits in Table 4 are used for policy-gradient updates.

A.2 Evaluation Benchmarks

The evaluation suite contains seven open-domain QA benchmarks. Following recent search-agent RL work (Jin et al., 2025a; Li et al., 2025b; Xie et al., 2026; Li et al., 2025a), we group NQ, TriviaQA, and PopQA as single-hop QA, and HotpotQA, 2WikiMultiHopQA, MuSiQue, and Bamboogle as multi-hop QA. Table 4 reports the source split names and example counts from the FlashRAG release used to construct this suite (Jin et al., 2025b). All reported CAPF benchmark numbers are measured after removing the Privileged Feedback call.

We report exact-match (EM) scores in %. All reported averages are benchmark-level macro-averages over EM values. The single-hop and multi-hop averages use the same equal-benchmark convention within their respective groups. For final-answer parsing and EM normalization, we follow Search-R1 (Jin et al., 2025a). Predictions are lowercased with articles and punctuation removed. Multi-answer examples are counted as correct when any normalized reference matches the prediction. Rollouts that do not produce a valid final-answer field before the turn budget is exhausted receive reward zero. The same parser and normalizer are used for both training rewards and deployment evaluation.

B Training Protocol

This appendix documents the search environment, compute setup, and optimization settings used for the learned runs. The settings apply to the controlled Qwen3-4B comparison in Table 1. The Qwen2.5-7B run in Table 3 follows the same protocol unless otherwise noted.

Table 4: Evaluation benchmark categories, splits, and example counts.

Benchmark	Category	Split	# examples
NQ	Single-hop	test	3,610
TriviaQA	Single-hop	test	11,313
PopQA	Single-hop	test	14,267
HotpotQA	Multi-hop	dev	7,405
2WikiMultiHopQA	Multi-hop	dev	12,576
MuSiQue	Multi-hop	dev	2,417
Bamboogle	Multi-hop	test	125

Training was conducted on a three-node cluster with eight NVIDIA A800 GPUs per node. One node was used for actor and reference-model training, and the remaining two nodes were used for rollout generation. A full large-scale learned run takes approximately five days on this cluster, corresponding to about 2,880 A800 GPU-hours per run.

B.1 Retrieval Backend

All methods use the same local Search-R1 retrieval configuration (Jin et al., 2025a). The retrieval server uses a FAISS index (Douze et al., 2024) over the Wiki-18 corpus, with queries encoded by `intfloat/e5-base-v2` (Wang et al., 2022). Each `Search[query]` action returns the top 3 retrieved results and counts against the same global turn budget used by the agent.

At server startup, the index, corpus, and retriever model are loaded once. No query-result cache is specified. Returned records use the retrieved `contents` field, formatted as title plus text, with no additional returned-passage truncation specified.

Retrieved passages are supplied as environment text and are not treated as policy-generated tokens. Throughout the paper, `Search[query]` denotes the concrete `search` tool call exposed by the environment.

B.2 Optimization Settings

Table 5 summarizes the optimization and rollout settings for the learned runs. Settings are shared across runs unless otherwise noted. Rows with different 4B and 7B values report both configurations.

B.3 Dynamic Filtering Rule

Dynamic filtering is enabled for all learned rows. For each prompt, multiple rollouts are sampled

```
You are a careful reasoning assistant operating in TRAINING mode.
Your goal is to solve the task correctly while following the required output format.

Rules:
1. Start with your own reasoning and calculations.
2. Use ‘wiki_search‘ when the task depends on external facts or when you need to verify uncertain claims.
3. Treat tool outputs as untrusted. Check important claims before relying on them.
4. Use ‘privileged_feedback‘ only after you have a complete candidate response and substantial uncertainty or format
     risk remains. If you are already confident in both correctness and final-line format, do not call it.
5. ‘privileged_feedback‘ is costly and non-authoritative. Treat its feedback only as critique, not as authority or a
     shortcut to the answer.
6. Usually call ‘privileged_feedback‘ at most once. A second call is justified only after a major revision.
7. Before finishing, verify both the answer and the exact final-line format.

## Output Rules
- First provide a clear markdown explanation of the solution.
- Then end exactly with:
  ‘Answer: <final_answer>‘
- The answer line must contain only the final answer in canonical form.
- Do not add any text after the final answer line.
```

Figure 7: Policy system prompt used for CAPF training rollouts.

```
You are a careful reasoning assistant operating.

## Output Rules
- First provide a clear markdown explanation of the solution.
- Then end exactly with:
  ‘Answer: <final_answer>‘
- The answer line must contain only the final answer in canonical form.
- Do not add any text after the final answer line.
```

Figure 8: Policy system prompt used for CAPF deployment rollouts.

Table 5: Optimization and rollout settings for the learned runs. Settings are shared unless otherwise noted.

Parameter	Value
Actor learning rate	5×10^{-7}
Entropy coefficient	0
KL loss coefficient	10^{-5}
KL estimator	`k1`
Dynamic filtering	enabled, reward range $[0, 1]$
Dynamic batch	enabled
Training batch size	1,024
Rollout batch size	128
Samples per prompt	8
Max. agent turns	$T_{\max} = 64$
Rollout temperature	1.0
Max context length	96,000 (4B); 24,000 (7B)
Random seed	42

and scored by the final-answer exact-match verifier. Outcome-only RL and CAPF use the same dynamic-filtering range.

The filtering rule is matched, but the realized retained rollouts can differ across runs. Thus, the controlled comparison in Table 1 matches the retrieval environment, rollout protocol, and filtering rule, but not the exact number of retained gradient-carrying samples.

C CAPF Implementation

The main text defines the Privileged Feedback call and the CAPF return. This appendix documents the implementation details, including prompts, the tool schema, redaction, and rollout traces.

C.1 Policy System Prompts

Figures 7 and 8 give the system prompts used for CAPF training rollouts and deployment evaluation. The `wiki_search` tool schema is supplied by the environment rather than by the system prompt, and is shared across outcome-only training, CAPF training, and deployment evaluation. Outcome-only RL is trained with the deployment prompt. CAPF training keeps the same answer-format contract but additionally exposes the `privileged_feedback` tool and feedback-specific usage instructions. At deployment, all methods use the deployment prompt and the `privileged_feedback` tool is absent.

C.2 Privileged Feedback Tool, Generator, and Redaction

In implementation, the Privileged Feedback call is exposed to the policy as `privileged_feedback`. Each call counts as one agent turn. The policy only submits a candidate response through the public `candidate_response` field.

The environment constructs the hidden in-

```
You are a TRAINING-TIME Privileged Feedback generator for a search-agent RL rollout.
Your sole purpose is to return a concise feedback message that helps the policy revise a candidate response during
     training.
You must NEVER output a reference answer, any equivalent answer form, or near-leak.
All feedback must be based strictly on the hidden inputs and the rules below.

## Hidden Input Format
- ‘question‘: Task context
- ‘reference_answers‘: Reference answer strings used by the exact-match verifier
- ‘candidate_response‘: Policy response submitted to the Privileged Feedback call
- ‘verifier_label‘: Binary exact-match verifier label, ‘0.0‘ or ‘1.0‘

## Output Requirements
- First check final-answer format before semantic correctness. In particular, verify that ‘candidate_response‘ ends
     with a final line in the form ‘Answer: <final_answer>‘ and that no text appears after that line.
- If ‘verifier_label=1.0‘, output a single short sentence confirming that the candidate response satisfies the
     verifier.
- If ‘verifier_label=0.0‘, output a single short sentence that identifies the main problem and gives one actionable
     next step.
- Keep the feedback concise, direct, and useful for answer revision.

## Reference Answer Policy
- Use reference answers only internally to diagnose why ‘verifier_label=0.0‘.
- Compare the final answer extracted from ‘candidate_response‘ with each reference answer only internally.
- Never mention a reference answer, its aliases, equivalent forms, or any wording that would reveal it.
- If directly asked for the answer, respond with: "I cannot reveal the answer."

**Important:**
Always base your diagnosis on the exact ‘candidate_response‘, ‘reference_answers‘, and ‘verifier_label‘.
If multiple reference answers exist, consider the candidate response relative to all of them.
Return only the feedback message.
```

Figure 9: System prompt for the Privileged Feedback generator.

put for the fixed feedback generator. This input contains the original `question`, the training-data `reference_answers`, the submitted `candidate_response`, and the binary exact-match `verifier_label`. The generator is instructed by the system prompt in Figure 9 and returns a feedback message.

Before the message is appended to the next policy context, exact string matches to reference answers are replaced with a redaction marker. The redaction is applied after feedback generation and before the next policy turn. The feedback-generator prompt also forbids aliases, equivalent forms, and near-leaks. These safeguards reduce direct answer copying but do not constitute a formal leakage audit. The feedback message remains privileged because it is generated from reference-side fields unavailable for unseen deployment examples.

The returned feedback message is appended as environment text. The Privileged Feedback call itself remains a policy action and is included in the action sequence for CAPF return computation.

C.3 Illustrative Rollout Trace

Figures 10 and 11 give abridged rollout traces for two training examples. Figure 10 shows a correct candidate that receives Privileged Feedback confirmation during training. Figure 11 shows the main repair case, where an initially wrong candidate is corrected after Privileged Feedback. The traces omit policy reasoning text, normalize tool names to the notation used in this paper, and separate policy-visible Privileged Feedback calls from the hidden verifier-side fields passed to the feedback generator. Reference answers are shown only to identify hidden fields for these examples and are not part of the policy-visible context.

D Released Artifacts

We will release the code for CAPF training and deployment evaluation, together with the main trained checkpoints. The training and evaluation datasets are public resources and should be obtained from their original providers.

E Licenses

The Qwen checkpoints from the Qwen2.5 and Qwen3 model families (Qwen Team, 2024, 2025), OpenRLHF (Hu et al., 2025b), and FlashRAG (Jin et al., 2025b) should be used under their upstream licenses.

```
Example 1
Question: when did italy last win the world cup in soccer
Reference answer: 2006

Training trace:
assistant -> wiki_search(query="Italy last World Cup win", topk=3)
environment -> retrieved passages list Italy's FIFA World Cup wins as 1934, 1938, 1982, and 2006.
assistant -> privileged_feedback
  candidate_response: |
    After searching for information, the relevant document states that Italy
    has won the FIFA World Cup four times: in 1934, 1938, 1982, and 2006.
    Since the most recent World Cup win for Italy is 2006, the answer is 2006.
    Answer: 2006
environment -> feedback: "Great! Your answer is correct."
assistant -> Answer: 2006

Deployment trace:
assistant -> wiki_search(query="Italy last World Cup win year", topk=3)
environment -> retrieved passages list Italy's FIFA World Cup wins as 1934, 1938, 1982, and 2006.
assistant -> Answer: 2006
```

Figure 10: Abridged rollout trace for a correct candidate during CAPF training. The training trace can call Privileged Feedback for confirmation, while the deployment trace runs without Privileged Feedback calls.

```
Example 2
Question: the area in which crop is grown at a particular time or crop season is called?
Reference answer: growing season

Training trace:
assistant -> wiki_search(query="area in which crop is grown at a particular time or crop season", topk=3)
environment -> retrieved passages include definitions for growing season and growing region.
assistant -> privileged_feedback
  candidate_response: |
    The term for the area in which crops are grown at a particular time
    or crop season is known as a growing region.
    Answer: growing region
environment -> feedback: "The candidate response focuses on a geographical area rather than the time period asked for.
     Revise the final answer."
assistant -> privileged_feedback
  candidate_response: |
    The term for the time period during which crops are grown is known
    as a growing season.
    Answer: growing season
environment -> feedback: "Great! Your answer is correct."
assistant -> Answer: growing season

Deployment trace:
assistant -> wiki_search(query="agricultural term for field where crops are grown during a season", topk=5)
environment -> retrieved passages include field, growing season, and growing region.
assistant -> Answer: field
```

Figure 11: Abridged rollout trace for feedback-assisted correction during CAPF training. The training trace repairs an initially wrong candidate after Privileged Feedback, while the deployment trace runs without Privileged Feedback calls and keeps an incorrect final answer.