MDS-ZERO: SELF-IMPROVING MULTIMODAL DEEP SEARCH VIA ONLINE QA–SOLVER CO-TRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

Multimodal large language models (MLLMs) have become strong visual reasoners, yet remain limited in open-world, multi-step search that requires grounding images, selecting tools, and integrating web evidence. Existing approaches typically construct and filter fixed offline corpora whose question distributions cannot adapt as the target deep-search agent improves. We present MDS-Zero (**M**ultimodal **D**eep **S**earch with **Zero** precomputed multi-hop corpora), an online multimodal QA–Solver co-training framework for self-improving multimodal deep-search agents. Starting from image-grounded VQA seed contexts, a trainable QA Generator retrieves evidence and constructs verified multi-hop questions online. A separate deep-search Solver then selects tools, gathers evidence, and answers the generated questions. To focus generation on the Solver's learning frontier, we derive a Goldilocks reward for the QA Generator from the Solver's group-wise rollout success rate: questions solved by all rollouts or by none receive zero reward, whereas questions near the current frontier are favored. We optimize the two policies independently with GRPO, without sharing parameters or gradients. Across six multimodal deep-search benchmarks, MDS-Zero improves a Qwen3-VL-8B Solver from 36.91% to 51.58% mean accuracy and achieves performance comparable to GPT-5 under the same agent workflow. The same framework also improves a newer Qwen3.6-27B Solver by 10.82 points. These results show that verified online question generation provides a policy-specific curriculum for continually improving multimodal deep-search agents.

1 INTRODUCTION

Multimodal large language models (MLLMs) have made substantial progress in visual understanding and reasoning (Xue et al., 2024; DeepSeek-AI, 2025; Huang et al., 2025; Qwen-Team, 2025). Yet, turning them into open-world agents that can identify entities in images, gather web evidence, and compose multi-hop answers through external tools remains challenging (Geng et al., 2025; Chng et al., 2025; Wu et al., 2025). Improving a target multimodal deep-search agent remains an open problem. We refer to this deployment-time agent as the *Solver agent*. The Solver must learn not only to reason over an image, but also to decide what to search for, which tools to invoke, and how to integrate evidence collected across modalities.

A dominant approach is to improve multimodal deep-search agents through an offline data factory (Peng et al., 2026; Zhang et al., 2026; Yao et al., 2026; Chen et al., 2026). Starting from seed images or questions, capable models and tools synthesize candidate multi-hop QA pairs and search trajectories; verification and filtering then turn them into SFT or RL corpora such as MTA-Agent (Peng et al., 2026), MM-DeepResearch (Yao et al., 2026), and REDSearcher (Chu et al., 2026). These pipelines create valuable, broadly reusable datasets, but their question distributions are fixed before the target Solver begins to evolve. As the Solver evolves, a generic question may be unanswerable at an early stage, trivial at a later one, or simply unrelated to its current capability gaps. Consequently, a fixed offline corpus may devote much of its generation, verification, and tool-interaction workload to questions that are no longer informative for the evolving policy.

This mismatch suggests an alternative: allowing training questions to evolve with the Solver itself. Self-improvement and Generator–Solver co-evolution have shown that adaptive task generation can create curricula that track an agent's changing capabilities, in both text-based search and vision-

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.



Figure 1: **Why co-training the QA policy matters. Left:** a frozen but valid QA policy produces progressively fewer informative questions as the Solver improves: the fraction of held-out QA groups with non-degenerate rollout outcomes ($0 < p_i < 1$) falls from 55.8% at S_0 to 44.3% at S_{50}. When the QA policy is updated online alongside the Solver, matched checkpoints retain or restore this training signal. **Right:** per-benchmark accuracy for selected 8B agents on MMSrch-Plus, BC-VL, MMSrch, and FVQA. `MDS-Zero` is highlighted in teal; the MMSrch bar includes the stronger reported specialist baseline. Missing bars denote benchmarks not reported by the corresponding prior work.

language reasoning (Lu et al., 2026; Yue et al., 2026; He et al., 2025; Thawakar et al., 2025; Wang et al., 2026). However, this idea is substantially harder in multimodal deep search. A useful question must be grounded in the seed image, supported by external evidence, answerable through an independent multi-hop tool trajectory, and free of ambiguous answers or evidence leakage. Moreover, optimizing a QA Generator only for Solver failures encourages unanswerable questions, whereas optimizing only for success collapses the curriculum to trivial ones. The challenge is therefore to train a QA Generator that produces verified questions at the evolving Solver's learning frontier. Figure 1 previews this curriculum mismatch and the effect of adapting the QA policy online.

We address this challenge with `MDS-Zero`, an asymmetric online co-training framework with two separate trainable models: a QA Generator and a Solver. The QA Generator is trained to produce appropriately challenging questions for the Solver, while the Solver is trained to answer multimodal deep-search questions. Given a seed image and QA pair, the QA Generator proposes multi-hop multimodal deep search QA pairs, and the Solver receives only the image and final question, then produces multiple independent tool-use rollouts. For each generated question, the fraction of correct Solver rollouts determines both learning signals: the QA Generator is rewarded for mixed outcomes, which indicate questions that are challenging but solvable, while the Solver is updated with GRPO on the same rollout groups. As the Solver improves, its rollout feedback trains the QA Generator to target the Solver's remaining capability gaps. The two models use separate parameters, optimizers, and gradient updates.

Contributions. We propose `MDS-Zero`, an asymmetric online QA–Solver co-training framework that updates both a **m**ultimodal **d**eep-**s**earch QA Generator and a Solver online. The QA Generator is trained from Solver rollout feedback to generate appropriately challenging questions, while the Solver is trained on the resulting question groups with mixed rollout outcomes. `MDS-Zero` improves an 8B Solver from 36.91% to 51.58% mean accuracy across six evaluation benchmarks, outperforming the compared 7–8B specialist agents on four of six benchmarks and approaching GPT-5 agent performance with an 8B model. The same pipeline also transfers to a newer 27B backbone. It requires no human intervention or manual filtering, no pre-generated multi-hop corpus, and no additional task-specific SFT stage.

2 Related Work

2.1 Multimodal Deep Search and Data Construction

Multimodal deep-search agents combine visual reasoning with external tools such as text and image search, browsing, and OCR (Wu et al., 2025; Narayan et al., 2025; Geng et al., 2026; Chng et al., 2025). Their long-horizon supervision is commonly prepared offline. A typical pipeline first selects seed images or questions, uses capable models and tools to synthesize candidate multi-hop ques-

tions, answers, and search trajectories, then verifies evidence support and filters invalid, ambiguous, duplicate, or shortcut-prone examples before assembling SFT or RL corpora. MTA-Agent (Peng et al., 2026), DeepMMSearch-R1 (Narayan et al., 2025), VSearcher (Zhang et al., 2026), MM-DeepResearch (Yao et al., 2026), and OpenSearch-VL (Chen et al., 2026) instantiate variants of this recipe with tool-assisted generation and quality, difficulty, or trajectory filtering. REDSearcher (Chu et al., 2026) and HyperEyes (Li et al., 2026) further combine such offline curation with controllable task complexity and efficiency-oriented trajectory selection, respectively. ODE (Huang et al., 2026b) makes the pipeline more policy-aware through on-policy data evolution. In contrast, we jointly train an image-grounded QA generator and a multi-tool Solver online. The Solver's group-wise success rate provides a Goldilocks signal that updates the QA generator; as the Solver improves, the generator learns to pose new questions at its shifting learning frontier, and the Solver is in turn trained on these online-generated questions. Unlike a fixed, potentially stronger offline generator that prepares a broadly useful corpus for many downstream models, our QA generator is optimized against the current Solver and can focus on questions that expose its specific capability gaps. Thus, the training distribution adapts online rather than being fixed in a pre-generated multi-hop corpus.

2.2 Self-Improving and Co-Evolution

In text-based search, Search Self-play (Lu et al., 2026) and Dr. Zero (Yue et al., 2026) co-evolve task proposal and solving to create an adaptive curriculum. R-Zero (Huang et al., 2026a), Absolute Zero (Zhao et al., 2026), Tool-R0 (Acikgoz et al., 2026), and Agent0 (Xia et al., 2025) extend this principle to reasoning and general tool-use environments. For vision-language models, VisPlay (He et al., 2025), EvoLMM (Thawakar et al., 2025), and V-Zero (Wang et al., 2026) use Questioner–Solver loops over visual inputs, while Vision-Zero (Wang et al., 2025), iReasoner (Sunil et al., 2026), and Agent0-VL (Liu et al., 2025) explore strategic self-play, intrinsic rewards, or tool-grounded self-refinement. Our setting instead targets open-world multimodal deep search. Although `MDS-Zero` follows the same co-evolutionary intuition, multimodal deep search poses a substantially harder task-generation problem: the QA Generator must retrieve and verify external evidence, preserve visual grounding and entity continuity across multiple hops, and calibrate question difficulty for an independently acting Solver that uses heterogeneous tools. A question that appears plausible from an image may still be unanswerable, ambiguous, or unsupported by the retrieved evidence, making naive task proposal unreliable. `MDS-Zero` therefore combines a fixed verification harness with Solver-calibrated rewards to construct a reliable, capability-adaptive curriculum.

3 Method

As Figure 2 shows, `MDS-Zero` co-trains a training-time QA Generator π_{Q} and a target multimodal deep-search Solver π_{S}. At iteration t, we start from an image-grounded seed context $C = (I, q^1, a^1)$. The QA generator π_{Q}^t (§3.1) produces N_{QA} evidence-grounded candidate QA pairs $\{(I, q_i, a_i)\}_{i=1}^{N_{\mathrm{QA}}}$. For every accepted q_i, the Solver π_{S}^t (§3.2) receives (I, q_i) and makes K independent multi-step tool-use rollouts. Let p_i be the fraction of those attempts whose final answers are correct. This one scalar closes the loop in two distinct ways. First, we map p_i to a Goldilocks reward (§3.3) for the QA trajectory that proposed q_i: questions solved by all rollouts or by none receive no QA reward, while questions near the target success rate are preferred. Second, the K attempts form one Solver GRPO group. The QA generator receives only this scalar-derived reward; the Solver is updated from its own individual rollout outcomes. No gradient is passed between the two policies, which retain separate parameters and optimizers.

3.1 Online QA Generation with a Harness

The QA Generator is a trainable agent that constructs challenging multi-hop multimodal QA pairs for training the deep-search Solver agent. It includes: (1) a trainable LLM policy that writes web-search queries, then proposes a single-hop question–answer pair grounded in the retrieved evidence; and (2) a fixed, non-trainable harness that executes the policy's actions, checks the validity of each proposed hop, and merges accepted single-hop QA pairs into a final multi-hop QA task.

Growing an n-hop question. Rather than precomputing a generic training corpus, the QA Generator π_{Q} converts a seed context $C = (I, q^1, a^1)$ into a multi-hop QA pair online. We count C as the first of n hops and set $\bar{q}^1 := q^1$; its answer a^1 is the initial entity to explore. For each subsequent hop $h \in \{2, \ldots, n\}$, π_{Q} searches for information about the previous answer a^{h-1}, then proposes an

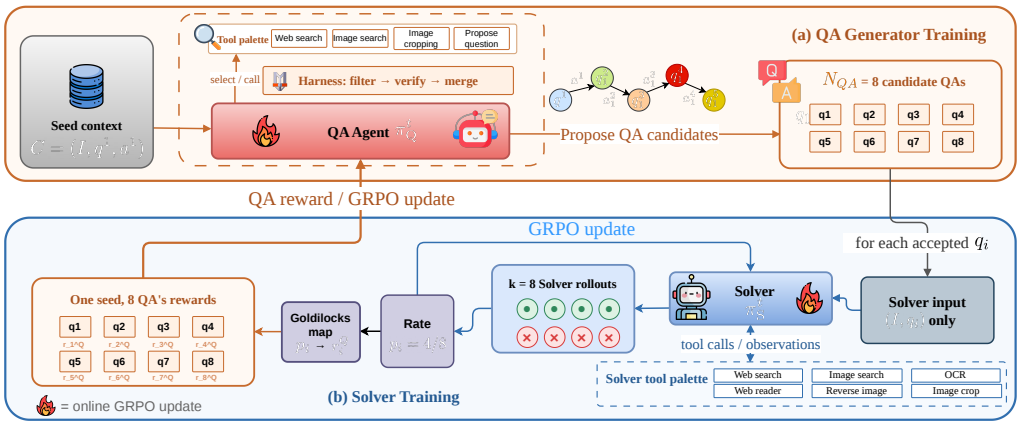


Figure 2: **Overview of `MDS-Zero`.** The QA Generator and Solver are updated online from separate GRPO objectives, while the fixed harness executes and validates QA-construction actions. The QA-side tool palette depicts the general harness interface; the main experiments enable web-text search only for QA construction.

evidence-based QA pair (q^h, a^h). The new question q^h asks for a fact about the previous entity a^{h-1}, while its answer a^h identifies the entity used to extend the next hop. For each non-final hop, this entity must be specific, and evidence-supported (Appendix A.1). Once the proposal is accepted, the harness combines q^h with the running question $\bar{q}^{h-1}$ to produce $\bar{q}^h$. Crucially, this merge removes the explicit mention of a^{h-1}: the resulting question reaches a^h through the preceding description rather than revealing the intermediate answer. Repeating this process to the target depth n yields $(I, q_i, a_i) = (I, \bar{q}^n, a^n)$. An example is illustrated in Appendix A.2.

Learned LLM policy. The learned policy selects the next action and writes that action's arguments. It alternates between (1) a *web-search* action containing a query and (2) a *propose-next-hop* action containing a candidate question, answer, supporting evidence, and a source field. Thus, the policy learns both what information to seek and how to turn that information into the next QA hop.

Non-trainable QA harness. Our fixed harness adapts the offline verification-and-merging pipeline of MTA-Agent (Peng et al., 2026) for online policy training. After a search action, it retrieves pages, filters them into bounded evidence chunks, and returns those chunks as an observation for the subsequent proposal action. After a proposed hop, it checks that the question is anchored to, and actually requires, the current entity. It then checks the answer uniqueness, temporal stability, and self-contained wording. It rejects source-referential, vague, generic, or otherwise non-chainable proposals. Finally, the harness merges an accepted proposal with the running question, verifies that the merged question uniquely resolves to the intended answer, and discards questions that a fixed direct-answer check can solve without search. More details can be found in the Appendix B.

Harness feedback and accepted-hop training. For every proposed hop, the harness appends a feedback message to the context as a user turn. When a hop is accepted, this message contains the updated merged question $\bar{q}^h$, the next entity a^h, and the acceptance signal; the QA Generator then uses this updated context to select its next action. Only action trajectories that lead to an accepted hop are used to train π_{Q}. If a proposal is rejected, its retrieved evidence and proposed hop are retained only as context and diagnostic feedback for a new attempt; all policy tokens associated with that unsuccessful attempt are masked from the training loss. Representative feedback and token-level credit assignment are provided in Appendix B.1 and Appendix B.3, respectively.

3.2 Multi-tool Solver Rollouts

For each accepted training question q_i, we sample K independent ReAct rollouts. At step t of rollout j, the Solver either selects a tool and generates its arguments or terminates by producing a final answer. When a tool is selected, the environment executes the call and appends the returned observation to the history, yielding H_{ij}^{t+1} for the next decision. This interaction loop continues until the Solver returns a final answer or reaches the maximum number of interaction steps. Let $c_{ij} \in \{0, 1\}$ denote the terminal correctness of rollout j for question q_i, determined by answer

matching (Appendix D.4). The group success rate is

$$p_i = \frac{1}{K} \sum_{j=1}^{K} c_{ij}. \tag{1}$$

We update π_S with standard GRPO only when $0 < p_i < 1$. These groups contain both correct and incorrect rollouts and therefore provide a useful relative learning signal.

3.3 Solver-Calibrated QA Scores

The same rate p_i supplies an adaptive difficulty signal for the QA generator. We map it to a Goldilocks score,

$$r_i^{\mathrm{Q}} = \begin{cases} 0, & p_i \in \{0, 1\}, \\ \dfrac{p_i}{\tau}, & 0 < p_i \le \tau, \\ \dfrac{1 - p_i}{1 - \tau}, & \tau < p_i < 1, \end{cases} \tag{2}$$

where $\tau \in (0, 1)$ is the target Solver success rate. The reward peaks at $p_i = \tau$, while questions solved by every rollout or by none receive zero reward. Optimizing π_Q with GRPO therefore favors questions that are difficult but still solvable by the current Solver, rather than questions that merely maximize failure. For clarity, our main setting uses $K = 8$ and $\tau = 0.5$. Thus, when $k = 0, \ldots, 8$ of the eight Solver rollouts answer a question correctly, its QA score is

k	0	1	2	3	4	5	6	7	8
r_i^{Q}	0	.25	.50	.75	1.00	.75	.50	.25	0

Every QA candidate is scored separately. For one seed, the figure's $N_{\mathrm{QA}} = 8$ candidates yield the reward set $\{r_i^{\mathrm{Q}}\}_{i=1}^{N_{\mathrm{QA}}}$, which is used to optimize π_Q with GRPO over these QA-generation trajectories. This per-candidate assignment is what allows the curriculum to track the Solver's changing learning frontier.

4 Experiments

4.1 Experiment Setup

4.1.1 Datasets

Evaluation. We evaluate the trained Solver on six benchmarks that require visual evidence understanding and open-web research: MMSearch-Plus (MMSrch-Plus) (Tao et al., 2025), HR-MMSearch-VL (HR-MMSrch) (Chng et al., 2025), BrowseComp-VL (BC-VL) (Geng et al., 2025), MMSearch-VL (MMSrch) (Jiang et al., 2024), the testmini split of VDR-Bench (VDR) (Huang et al., 2026c), and the official FVQA test split (FVQA) (Wu et al., 2025).

Training seeds. Following the publicly described seed-source protocol of MTA-Agent (Peng et al., 2026), we initialize co-training from a fixed image-grounded one-hop VQA pool: InfoSeek (Chen et al., 2023), the training split of FVQA (Wu et al., 2025), InfoVQA (Mathew et al., 2022), and OK-VQA (Marino et al., 2019). Each seed contains only $C = (I, q^1, a^1)$: an image, its original one-hop question, and its answer. Appendix D.1 gives the exact pool composition and filtering rules. The pool provides initial visual grounding rather than an offline deep-search corpus: it contains no pre-generated multi-hop QA pairs, retrieved evidence, or search trajectories. All multi-hop questions used for optimization are subsequently constructed and verified online by the QA generator and harness in Section 3.1.

4.1.2 Agent Setup

Solver Setup. We use a **ReAct** (Reasoning and Acting) framework (Yao et al., 2022) for the deep-search Solver (§3.2). It operates for at most $T = 10$ iterations. At each iteration, it generates a structured output comprising (i) a *reasoning* trace based on the current state, (ii) an *action* specifying a tool and its parameters, (iii) a binary `should_stop` flag, and (iv) a scalar *confidence* score. The loop terminates early when `should_stop=true` and the confidence score exceeds 0.7, or the

maximum budget of $T = 10$ iterations is exhausted. Upon termination, the agent generates its final answer conditioned on the complete interaction history. Further details are provided in Appendix C.

QA Agent setup. The QA Generator follows the construction-specific interaction loop described in Section 3.1. For each seed context, we sample 8 QA-generation trajectories with `max_turns` set to 16. At a search step, the Generator emits `action`, `thought`, `tool`, and `query`; the harness executes web-text search, filters the returned pages into evidence chunks, and appends them to the next interaction state. At a proposal step, it emits `action`, `thought`, `question`, `answer`, `evidence`, and `source`. The harness verifies the proposal and, if accepted, appends the merged question and next entity; otherwise, it returns diagnostic feedback. A trajectory terminates after reaching the target depth or exhausting its configured turn budget. Appendix A.1 specifies the state and action schemas.

Tool Palette. The Solver has access to: *Web search* retrieves web results, including snippets, titles, and URLs. *Image search* returns textual descriptions of images found on the web. *Web URL reader* extracts the full text of a given webpage. *Reverse image search* uses Google Lens to identify objects and landmarks. *OCR* extracts visible text from an image, and *image cropping* extracts a specified region from an image. All Solver tool calls are sent to a local observation server, and the returned observation is appended to the chat history for the next action. More details are in Appendix C.1.

4.1.3 BASELINES

We compare to three commercial large language models: GPT-5 (Singh et al., 2025), Claude Sonnet 4.5, and Gemini 3 Pro (DeepMind, 2025). We evaluate each model under two settings: (1) direct answering without tools and (2) agent-based answering using the same framework and tool configuration as our method. In addition, we report results from prior work: MTA-DeepSearch (Peng et al., 2026), MMSearch-R1 (Wu et al., 2025), SenseNova-MARS (Chng et al., 2025), WebWatcher (Geng et al., 2025), VSearcher (Zhang et al., 2026), MM-DeepResearch (Yao et al., 2026), OpenSearch-VL (Chen et al., 2026), ODE (Huang et al., 2026b), and Vision-DeepResearch.

4.1.4 TRAINING SETUP

The QA generator and Solver begin from separate copies of `Qwen3-VL-8B-Instruct` (Qwen-Team, 2025). Instead of synthesizing a multi-hop corpus before training, we initialize online generation from a fixed pool of image-grounded, one-hop VQA seed contexts in Section 4.1.1. Each optimization iteration draws 8 seed QAs from the pool. The QA policy generates 8 trajectories targeting a total depth of 5 hops (one seed hop and four extensions) per seed, and every retained QA is answered by 8 independently sampled Solver rollouts. This produces 64 QA trajectories and $512 = 8 \times 8 \times 8$ Solver trajectories in one iteration. We optimize the two policies independently with GRPO for 50 iterations, with learning rate 10^{-6}, and Goldilocks target $\tau = 0.5$. The QA generator and Solver have turn caps of 16 and 10; their prompt and response limits are $8,192$ and $24,576$ tokens, respectively. All main 8B numbers come from the Solver after the 50th co-training iteration. Full configuration is in Appendix D and online data and resource accounting are in Appendix D.3.

4.2 MAIN RESULTS

Strong improvement without a pre-generated multi-hop corpus. As shown in Table 1, comparing to the matched `Qwen3-VL-8B-Instruct` Agent Workflow baseline (in gray), `MDS-Zero` raises the six benchmark mean accuracy from 36.91% to 51.58% ($+14.67$ points). The run uses 400 one-hop seed-context draws over 50 co-training iterations: the QA generator and Solver are optimized directly with GRPO, without a separate multi-hop data factory or an additional task-specific SFT stage. Full resource and service-use accounting is reported in Appendix D.3.

Comparison with 7–8B specialist deep-search agents. Among the specialist agents listed in Table 1, `MDS-Zero` records the highest score on four of six benchmarks. Prior specialist agents including MTA-Agent (Peng et al., 2026), VSearcher (Zhang et al., 2026), and OpenSearch-VL (Chen et al., 2026) first construct, filter, or reuse offline teacher-assisted QA or trajectory corpora before applying SFT and/or RL. Because their results use their original tool stacks and training pipelines, this is not a cost-matched comparison. However, `MDS-Zero` and MTA-DeepSearch (Peng et al., 2026) share the same raw one-hop seed pool: `MDS-Zero` replaces MTA-DeepSearch's offline multi-hop data-construction pipeline with online QA–Solver co-training. Without pre-generating a multi-hop corpus or adding an additional task-specific SFT stage, `MDS-Zero` outperforms MTA-DeepSearch on four of the five benchmarks for which both methods report results.

Table 1: Comparison on multimodal deep-search benchmarks. Results report mean accuracy (%) over 3 generation seeds. Avg is the unweighted mean across all six benchmarks and is reported only when all six scores are available. † marks results reported by the original papers.

Model	MMSrch-Plus	HR-MMSrch	BC-VL	MMSrch	FVQA	VDR	Avg
Direct Answer							
GPT-5	13.18	16.39	41.60	36.47	50.83	14.40	28.81
Claude 4.5 Sonnet	5.40	14.40	36.10	40.00	49.60	7.00	25.42
Gemini-3-Pro	26.37	23.61	41.35	65.88	58.94	14.00	38.36
Qwen3-VL-8B-Instruct	1.29	4.59	20.05	12.35	26.94	3.00	11.37
Agent Workflow							
GPT-5	31.61	52.13	51.63	77.65	72.28	26.60	51.98
Claude 4.5 Sonnet	43.00	53.40	65.70	84.70	77.00	24.40	58.03
Gemini-3-Pro	33.51	53.20	51.78	82.94	76.67	27.60	54.28
Qwen3-VL-8B-Instruct	18.38	32.13	35.89	57.06	64.22	13.80	36.91
Multimodal Deep-Search Agents							
MMSearch-R1-7B	–	–	–	53.80†	58.40†	–	–
SenseNova-MARS-8B	–	41.64†	–	67.84†	67.11†	–	–
WebWatcher-7B	–	–	21.20†	49.10†	–	–	–
VSearcher-8B	–	–	30.80†	47.20†	–	–	–
MM-DeepResearch-8B	–	–	37.90†	67.80†	69.20†	–	–
OpenSearch-VL-8B	–	–	37.60†	64.50†	71.50†	20.80†	–
ODE-8B-RL	24.90†	–	41.90†	66.00†	64.70†	20.40†	–
Vision-DeepResearch-8B	20.40†	–	42.60†	69.60†	64.70†	**29.20**†	–
MTA-DeepSearch-8B	26.77†	47.54†	44.36†	**79.41**†	73.06†	–	–
`MDS-Zero` (Qwen3-VL-8B)	**29.90**	**53.10**	**48.40**	78.20	**76.20**	23.65	51.58

Approaching GPT-5 at the 8B scale. Despite using an 8B open model, `MDS-Zero` reaches 51.58% average accuracy across six benchmarks, coming within 0.40 percentage points of GPT-5 under the same Agent Workflow (51.98%). Although stronger commercial agents remain ahead overall, this result shows that online QA–Solver co-training can make a compact model competitive with a strong frontier system without increasing the Solver backbone size.

5 Analysis

5.1 Does the QA Agent Learn an Adaptive Curriculum?

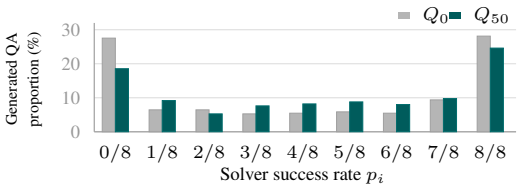
Before online co-training, the pretrained base model and our fixed QA harness already function as a QA agent: they can search evidence, propose next-hop QA pairs, and generate verified multi-hop questions. The key question is *whether further training of this agent is necessary*. We use checkpoint cross-play to assess it. Starting from 128 held-out seed QA pairs, the agent generates 4 questions q_i per seed. For each generated question, we estimate the Solver's success rate p_i using 8 independent rollouts. We define the *informative group rate* (I.G. rate) as the proportion of questions with $0 < p_i < 1$, because only these questions yield non-degenerate GRPO groups for the Solver.

The left panel of Figure 1 tests whether the initial QA agent, Q_0 (`Qwen3-VL-8B-Instruct`), remains suitable as the Solver evolves. Its dashed red curve fixes Q_0 and varies the Solver from S_0 to S_{50}. The I.G. rate falls from 55.8% to 44.3%. Thus, even a capable frozen QA agent and fixed harness become mismatched to a stronger Solver. We then fix the Solver at S_{50} and use successive QA checkpoints to generate 4 questions from each of the same 128 held-out seed QA pairs (512 questions in total). The right panel of Table 2 shows that the I.G. rate rises from 44.3% for Q_0 to 56.8% for Q_{50}, a gain of 12.5 percentage points. Since the Solver, seed contexts, and fixed harness are unchanged, this gain comes from the learned QA policy.

The left plot of Table 2 reveals how the distribution changes. Relative to Q_0, Q_{50} reduces universally failed groups ($p_i = 0$) from 27.5% to 18.6% and universally solved groups ($p_i = 1$) from 28.1% to 24.6%. At the same time, it raises the proportion of groups in the central $3/8$–$5/8$ success-rate

Table 2: QA–Solver checkpoint cross-play. **Left:** with the final Solver fixed at S_{50}, the learned QA policy shifts generated tasks away from universally failed and universally solved rollout groups toward the Goldilocks region. **Right:** the informative-group rate increases across QA checkpoints with S_{50} fixed.

(a) Rollout-success distribution at fixed S_{50}



(b) QA policy improves with fixed Solver

Fixed Solver: S_{50}; vary the QA policy.

QA checkpoint	Q_0	Q_{10}	Q_{20}	Q_{30}	Q_{40}	Q_{50}
I.G. rate (%)	44.3	46.4	48.4	49.8	49.1	56.8
Net change			$44.3\% \rightarrow 56.8\%$ (+12.5 **pp**)			

Table 3: Ablating the QA difficulty target on the 8B Solver. Overall is the unweighted mean across the four benchmarks.

QA target	QA reward profile $k = 0, \ldots, 8$	MMSrch-Plus	HR-MMSrch	BC-VL	MMSrch	Overall
$\tau = 0.50$		29.9%	53.1%	48.4%	78.2%	52.4%
$\tau = 0.25$		25.1%	51.1%	45.4%	74.7%	49.1%

range from 16.6% to 24.6%. The solid teal curve in Figure 1 shows the complementary matched-checkpoint view: its I.G. rate stays between 50.1% and 56.8% throughout co-training. Together, these results show that the QA policy learns to generate questions that remain useful for the current Solver, rather than supplying a static distribution that is useful only near initialization.

5.2 Ablating the QA Difficulty Target

The Goldilocks reward does not merely reject degenerate question groups: its target τ specifies the Solver success rate that the QA policy is encouraged to produce. Our default $\tau = 0.5$ assigns maximal reward to questions answered correctly by 4 of 8 Solver rollouts. We also evaluate a more aggressive low-success target motivated by Absolute Zero (Zhao et al., 2026). Its inverse-accuracy reward is largest when exactly 1 of 8 Solver rollouts is correct. Rather than directly use such a strict target, we set $\tau = 0.25$, which peaks at $2/8$ correct rollouts. A lone correct rollout can be a lucky sample rather than reliable evidence that the generated question is solvable, whereas 2 correct rollouts provide a more robust indication of a difficult but repeatably solvable question. For $k = 0, \ldots, 8$ correct rollouts, this target gives the QA reward vector $(0, .5, 1, .833, .667, .5, .333, .167, 0)$.

Table 3 compares both targets after the same 50-step co-training budget. The lower target obtains an unweighted overall score of 49.1%, 3.3 points below the 52.4% obtained by $\tau = 0.5$. For multi-tool deep search, difficult but reliably solvable questions are more useful than questions that are merely close to failure. With $K = 8$, a $2/8$ target has only one lower-success outcome before a question becomes completely unsolved ($k = 0$) and is discarded from the Solver update. In contrast, a $4/8$ target keeps a wider margin of mixed-success groups and supplies both successful and unsuccessful rollouts for relative learning.

5.3 Does Updating the QA Policy Improve the Solver?

The cross-play evidence above measures whether a QA policy remains informative as its Solver changes. Table 4 provides the outcome-level complement: how much does a valid but fixed QA policy improve the Solver, and what does updating that policy add? The untuned `Qwen3-VL-8B-Instruct` agent begins at 35.87% average accuracy under the same agent workflow. Holding the QA policy fixed while retaining the online harness raises this score to 44.28% (+8.41 points). The frozen and adaptive runs start from the same Solver initialization and use exactly the same seed contexts, harness, tools, rollout budget, and training schedule; the only training difference is whether the QA policy is updated. Thus, the initial frozen QA policy and harness are already useful—they generate verified questions that train the Solver beyond its base capability. Updating the QA policy throughout training raises the average further to 52.40%, an additional +8.12

Table 4: Comparison of fixed and adaptive QA policies for an 8B Solver.

Method	MMSrch-Plus	HR-MMSrch	BC-VL	MMSrch	Avg
Qwen3-VL-8B-Instruct (base)	18.38	32.13	35.89	57.06	35.87
MDS-Zero (frozen QA)	24.89	42.30	42.86	67.06	44.28
MDS-Zero (adaptive QA policy)	**29.90**	**53.10**	**48.40**	**78.20**	**52.40**

Table 5: Large-model context for scaling MDS-Zero to Qwen3.6-27B.

Method	MMSrch-Plus	HR-MMSrch	BC-VL	MMSrch	Overall (ours)
Reported 27–32B Multimodal Deep-Search Agents					
MTA-DeepSearch-32B†	31.93	53.95	53.77	82.35	–
ODE-30B-A3B-RL (MoE)†	28.10	–	46.10	69.70	–
REDSearcher-MM-RL-30B (MoE)†	26.60	–	57.20	72.90	–
HyperEyes-30B-RL (MoE)†	31.50	–	57.90	86.90	–
Vision-DeepResearch-30B-A3B (MoE)†	28.50	–	53.70	69.60	–
Our Local Evaluation					
Qwen3.6-27B (Base)	36.20	45.90	46.60	58.80	46.88
MDS-Zero (Qwen3.6-27B)	38.50	57.00	54.10	81.20	57.70

points over the frozen-QA control and $+16.53$ points over the base Solver. This pattern agrees with the checkpoint cross-play: online QA construction alone helps, but a policy that continues to adapt to Solver outcomes supplies a substantially stronger curriculum.

5.4 Scaling to a Larger, Newer Backbone

Our gains are not confined to the 8B setting. We also initialize both policies from Qwen3.6-27B (Qwen Team, 2026), a larger and newer backbone than the Qwen3-VL-8B Solver used in our main experiments. On the same seed VQA contexts, the resulting MDS-Zero raises the overall OOD accuracy from 46.88% for its Qwen3.6-27B base model to 57.70% ($+10.82$ points; Table 5). The improvement is positive on every constituent benchmark: $+2.3$ on MMSrch-Plus, $+11.1$ on HR-MMSrch, $+7.5$ on BC-VL, and $+22.4$ on MMSrch. Thus, the online QA–Solver loop continues to improve a substantially stronger Solver, rather than being specific to small models.

Table 5 additionally provides context from reported 27–32B multimodal deep-search agents. The external rows use earlier Qwen3-VL backbones, including dense 32B and 30B-A3B MoE variants, whereas our matched experiment uses Qwen3.6-27B. These values are not a controlled ranking: benchmark subsets, tool stacks, search budgets, and judges differ across papers. We therefore omit their aggregate scores and make the scaling claim only from the local, matched base-versus-MDS-Zero comparison.

6 Conclusion

We introduced MDS-Zero, an online QA–Solver co-training framework for improving multimodal deep-search agents without a pre-generated multi-hop corpus or an additional task-specific SFT stage. A trainable QA Generator retrieves evidence and builds multi-hop questions. The Solver's group rollout outcomes then update both policies with separate GRPO objectives: the QA Generator learns to propose questions that are difficult but solvable for the current Solver, while the Solver learns from the resulting mixed-outcome groups. Across six multimodal deep-search evaluation benchmarks, MDS-Zero improves a Qwen3-VL-8B Solver from 36.91% to 51.58% mean accuracy under the same agent workflow, while also improving a newer Qwen3.6-27B backbone. The experiments show that a valid static QA policy becomes increasingly mismatched as the Solver changes, whereas updating the QA policy restores a larger supply of useful training groups. The reward ablation further shows that questions should be challenging without being pushed too close to universal failure. These results support verified online task generation as a practical route to self-improving multimodal deep-search agents. Extending the harness to richer tool environments while reducing multi-tool rollout cost is a promising direction for future work.

AI USE STATEMENT

We used generative AI tools for literature organization, plotting and analysis code, and experiment-orchestration and debugging scripts. The research questions, method design, interpretation, and conclusions are the authors' own; the authors reviewed and tested AI-assisted code and verified reported results against experiment outputs and raw logs. Generative models also have substantive roles in the method and evaluation: Qwen models instantiate the trainable QA Generator and Solver, a frozen Claude Sonnet 5 endpoint supports model-assisted harness checks, `gpt-5-mini` adjudicates answers not resolved by deterministic matching, and Claude 4.8 Opus is used for the external QA-quality audit. Proprietary models are additionally evaluated as baselines. These uses are described in the corresponding method, experiment, and appendix sections. We take responsibility for the final text, claims, code, and experimental artifacts reported in this work.

ETHICS STATEMENT

This work uses publicly released models and VQA datasets together with publicly accessible web search results. The reported manual audit evaluates model outputs rather than participant behavior or personal attributes. We do not intentionally target private or sensitive information, although live web retrieval can surface personal, copyrighted, biased, outdated, or otherwise unreliable material. The fixed harness and answer adjudication reduce but do not eliminate these risks. Improved deep-search agents also carry dual-use risks by making large-scale information collection and synthesis easier. Deployments in high-stakes settings should therefore use appropriate access controls, source review, and human oversight. We report the main run's compute and external-call footprint in Appendix D.3.

REPRODUCIBILITY STATEMENT

Sections 3 and 4.1 specify the model, data, tool-use protocol, rewards, and optimization setup. The appendix provides the QA state and entity definitions, ordered harness checks, Solver interfaces, token-level masking rules, scoring procedure, training configuration, and verified workload and resource accounting. Training uses publicly released Qwen backbones and seed contexts drawn from public VQA datasets. Exact replay of live retrieval and proprietary model-assisted checks may vary with web content and service versions; we therefore separate deterministic checks from model-assisted checks and report their roles and failure behavior. These disclosures support protocol-level reproduction while making the remaining dependency on time-varying external services explicit.

REFERENCES

Emre Can Acikgoz, Cheng Qian, Jonas Hübotter, Heng Ji, Dilek Hakkani-Tür, and Gokhan Tur. Tool-r0: Self-evolving llm agents for tool-learning from zero data. *arXiv preprint arXiv:2602.21320*, 2026.

Shuang Chen, Kaituo Feng, Hangting Chen, Wenxuan Huang, Dasen Dai, Quanxin Shou, Yunlong Lin, Xiangyu Yue, Shenghua Gao, and Tianyu Pang. Opensearch-vl: An open recipe for frontier multimodal search agents. *arXiv preprint arXiv:2605.05185*, 2026.

Yang Chen, Hexiang Hu, Yi Luan, Haitian Sun, Soravit Changpinyo, Alan Ritter, and Ming-Wei Chang. Can pre-trained vision and language models answer visual information-seeking questions? In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 14948–14968, 2023.

Yong Xien Chng, Tao Hu, Wenwen Tong, Xueheng Li, Jiandong Chen, Haojia Yu, Jiefan Lu, Hewei Guo, Hanming Deng, Chengjun Xie, et al. Sensenova-mars: Empowering multimodal agentic reasoning and search via reinforcement learning. *arXiv preprint arXiv:2512.24330*, 2025.

Zheng Chu, Xiao Wang, Jack Hong, Huiming Fan, Yuqi Huang, Yue Yang, Guohai Xu, Chenxiao Zhao, Cheng Xiang, Shengchao Hu, et al. Redsearcher: A scalable and cost-efficient framework for long-horizon search agents. *arXiv preprint arXiv:2602.14234*, 2026.

Google DeepMind. Gemini 3. `https://deepmind.google`, 2025. Large language model.

DeepSeek-AI. Deepseek-v3.2: Pushing the frontier of open large language models, 2025.

Xinyu Geng, Peng Xia, Zhen Zhang, Xinyu Wang, Qiuchen Wang, Ruixue Ding, Chenxi Wang, Jialong Wu, Yida Zhao, Kuan Li, et al. Webwatcher: Breaking new frontier of vision-language deep research agent. *arXiv preprint arXiv:2508.05748*, 2025.

Xinyu Geng, Peng Xia, Zhen Zhang, Xinyu Wang, Qiuchen Wang, Ruixue Ding, Chenxi Wang, Jialong Wu, Kuan Li, Yida Zhao, et al. Webwatcher: Breaking new frontiers of vision-language deep research agent. In *International Conference on Learning Representations*, volume 2026, pp. 61240–61271, 2026.

Yicheng He, Chengsong Huang, Zongxia Li, Jiaxin Huang, and Yonghui Yang. Visplay: Self-evolving vision-language models from images. *arXiv preprint arXiv:2511.15661*, 2025.

Chengsong Huang, Wenhao Yu, Xiaoyang Wang, Hongming Zhang, Zongxia Li, Ruosen Li, Jiaxin Huang, Haitao Mi, and Dong Yu. R-zero: Self-evolving reasoning llm from zero data. In *International Conference on Learning Representations*, volume 2026, pp. 130770–130790, 2026a.

Shijue Huang, Hangyu Guo, Guanting Dong, Chenxin Li, Junting Lu, Xinyu Geng, Zhaochen Su, Zhenyu Li, Shuang Chen, Hongru Wang, et al. Towards on-policy data evolution for visual-native multimodal deep search agents. *arXiv preprint arXiv:2605.10832*, 2026b.

Wenxuan Huang, Bohan Jia, Zijie Zhai, Shaosheng Cao, Zheyu Ye, Fei Zhao, Zhe Xu, Xu Tang, Yao Hu, and Shaohui Lin. Vision-r1: Incentivizing reasoning capability in multimodal large language models. *arXiv preprint arXiv:2503.06749*, 2025.

Wenxuan Huang, Yu Zeng, Qiuchen Wang, Zhen Fang, Shaosheng Cao, Zheng Chu, Qingyu Yin, Shuang Chen, Zhenfei Yin, Lin Chen, et al. Vision-deepresearch: Incentivizing deepresearch capability in multimodal large language models. *arXiv preprint arXiv:2601.22060*, 2026c.

Dongzhi Jiang, Renrui Zhang, Ziyu Guo, Yanmin Wu, Jiayi Lei, Pengshuo Qiu, Pan Lu, Zehui Chen, Chaoyou Fu, Guanglu Song, et al. Mmsearch: Benchmarking the potential of large models as multi-modal search engines. *arXiv preprint arXiv:2409.12959*, 2024.

Guankai Li, Jiabin Chen, Yi Xu, Xichen Zhang, and Yuan Lu. Hypereyes: Dual-grained efficiency-aware reinforcement learning for parallel multimodal search agents. *arXiv preprint arXiv:2605.07177*, 2026.

Jiaqi Liu, Kaiwen Xiong, Peng Xia, Yiyang Zhou, Haonian Ji, Lu Feng, Siwei Han, Mingyu Ding, and Huaxiu Yao. Agent0-vl: Exploring self-evolving agent for tool-integrated vision-language reasoning. *arXiv preprint arXiv:2511.19900*, 2025.

Hongliang Lu, Yuhang Wen, Pengyu Cheng, Ruijin Ding, Jiaqi Guo, Haotian Xu, Chutian Wang, Haonan Chen, et al. Search self-play: Pushing the frontier of agent capability without supervision. In *International Conference on Learning Representations*, volume 2026, pp. 65132–65165, 2026.

Kenneth Marino, Mohammad Rastegari, Ali Farhadi, and Roozbeh Mottaghi. Ok-vqa: A visual question answering benchmark requiring external knowledge. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

Minesh Mathew, Viraj Bagal, Rubèn Tito, Dimosthenis Karatzas, Ernest Valveny, and CV Jawahar. Infographicvqa. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 1697–1706, 2022.

Kartik Narayan, Yang Xu, Tian Cao, Kavya Nerella, Vishal M Patel, Navid Shiee, Peter Grasch, Chao Jia, Yinfei Yang, and Zhe Gan. Deepmmsearch-r1: Empowering multimodal llms in multimodal web search. *arXiv preprint arXiv:2510.12801*, 2025.

Xiangyu Peng, Can Qin, An Yan, Xinyi Yang, Zeyuan Chen, Ran Xu, and Chien-Sheng Wu. Mta-agent: An open recipe for multimodal deep search agents. *arXiv preprint arXiv:2604.06376*, 2026.

Qwen-Team. Qwen3 technical report, 2025. URL `https://arxiv.org/abs/2505.09388`.

Qwen Team. Qwen3.6-27B: Flagship-level coding in a 27B dense model, April 2026. URL `https://qwen.ai/blog?id=qwen3.6-27b`.

Aaditya Singh, Adam Fry, Adam Perelman, Adam Tart, Adi Ganesh, Ahmed El-Kishky, Aidan McLaughlin, Aiden Low, AJ Ostrow, Akhila Ananthram, et al. Openai gpt-5 system card. *arXiv preprint arXiv:2601.03267*, 2025.

Meghana Sunil, Manikandarajan Venmathimaran, and Muthu Subash Kavitha. ireasoner: Trajectory-aware intrinsic reasoning supervision for self-evolving large multimodal models. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 29361–29372, 2026.

Xijia Tao, Yihua Teng, Xinxing Su, Xinyu Fu, Jihao Wu, Chaofan Tao, Ziru Liu, Haoli Bai, Rui Liu, and Lingpeng Kong. Mmsearch-plus: Benchmarking provenance-aware search for multimodal browsing agents. *arXiv preprint arXiv:2508.21475*, 2025.

Omkar Thawakar, Shravan Venkatraman, Ritesh Thawkar, Abdelrahman Shaker, Hisham Cholakkal, Rao Muhammad Anwer, Salman Khan, and Fahad Khan. Evolmm: Self-evolving large multimodal models with continuous rewards. *arXiv preprint arXiv:2511.16672*, 2025.

Han Wang, Yi Yang, Jingyuan Hu, Minfeng Zhu, and Wei Chen. V-zero: Self-improving multimodal reasoning with zero annotation. *arXiv preprint arXiv:2601.10094*, 2026.

Qinsi Wang, Bo Liu, Tianyi Zhou, Jing Shi, Yueqian Lin, Yiran Chen, Hai Helen Li, Kun Wan, and Wentian Zhao. Vision-zero: Scalable vlm self-improvement via strategic gamified self-play. *arXiv preprint arXiv:2509.25541*, 2025.

Jinming Wu, Zihao Deng, Wei Li, Yiding Liu, Bo You, Bo Li, Zejun Ma, and Ziwei Liu. Mmsearch-r1: Incentivizing lmms to search. *arXiv preprint arXiv:2506.20670*, 2(4), 2025.

Peng Xia, Kaide Zeng, Jiaqi Liu, Can Qin, Fang Wu, Yiyang Zhou, Caiming Xiong, and Huaxiu Yao. Agent0: Unleashing self-evolving agents from zero data via tool-integrated reasoning. *arXiv preprint arXiv:2511.16043*, 2025.

Le Xue, Manli Shu, Anas Awadalla, Jun Wang, An Yan, Senthil Purushwalkam, Honglu Zhou, Viraj Prabhu, Yutong Dai, Michael S Ryoo, et al. xgen-mm (blip-3): A family of open large multimodal models. *arXiv preprint arXiv:2408.08872*, 2024.

Huanjin Yao, Qixiang Yin, Min Yang, Ziwang Zhao, Yibo Wang, Haotian Luo, Jingyi Zhang, and Jiaxing Huang. Mm-deepresearch: A simple and effective multimodal agentic search baseline. *arXiv preprint arXiv:2603.01050*, 2026.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*, 2022.

Zhenrui Yue, Kartikeya Upasani, Xianjun Yang, Suyu Ge, Shaoliang Nie, Yuning Mao, Zhe Liu, and Dong Wang. Dr. zero: Self-evolving search agents without training data. *arXiv preprint arXiv:2601.07055*, 2026.

Ruiyang Zhang, Qianguo Sun, Chao Song, Yiyan Qi, and Zhedong Zheng. Vsearcher: Long-horizon multimodal search agent via reinforcement learning. *arXiv preprint arXiv:2603.02795*, 2026.

Andrew Zhao, Yiran Wu, Tong Wu, Quentin Xu, Yang Yue, Matthieu Lin, Shenzhi Wang, Qingyun Wu, Zilong Zheng, and Gao Huang. Absolute zero: Reinforced self-play reasoning with zero data. *Advances in Neural Information Processing Systems*, 38:105816–105879, 2026.

Table 6: Bridge-entity taxonomy used by the QA construction system, adapted from the seed-selection convention of MTA-Agent (Peng et al., 2026). The taxonomy constrains intermediate answers, which must support another retrieval step; a final-hop answer may instead be a short non-entity fact.

Category	Included referents	Valid bridge-answer examples
People	Uniquely named public, historical, scientific, or athletic figures	Ada Lovelace; Serena Williams
Organizations	Named institutions, companies, universities, government bodies, or sports teams	UNESCO; FC Barcelona
Locations and landmarks	Specific geographic, natural, or architectural referents	Nairobi; Machu Picchu
Products	Commercially named devices, vehicles, software, or other products	PlayStation 5; Toyota Prius
Events	Named sporting, historical, academic, or cultural events	Cannes Film Festival; 2016 Summer Olympics
Creative works	Titled films, books, artworks, television series, or music	*Spirited Away*; *The Starry Night*
Science and technology	Named species, compounds, devices, models, or datasets	Tyrannosaurus rex; BERT
Medical and health	Named diseases, conditions, or other specific medical entities	Alzheimer's disease; dengue fever
Business and finance	Named brands, firms, indexes, or financial instruments	Patagonia; Nasdaq-100
Space and astronomy	Named celestial objects, missions, telescopes, or astronomical events	Voyager 1; Andromeda Galaxy

A Detailed Agent Protocols

A.1 Bridge-Entity Definition and Generator State

Following the entity convention used to construct our seed pool (Peng et al., 2026), we define a *bridge entity* at every non-final hop as a single proper-name answer that denotes one unambiguous real-world referent, is supported by the retrieved evidence, and can anchor the next retrieval step. This covers named referents from the categories summarized in Table 6.

A multi-entity answer, unsupported alias, descriptive phrase, generic category, action, or bare quantity is not a valid bridge entity. The final hop need not support another extension: when the final-hop quantity option is enabled, it may instead end in an exact count, measurement, date, or other short factual answer.

The policy proposes the complete answer string a^h, but it does not create the two entity representations used by the next turn. After accepting a non-final hop, the fixed harness derives e^h, a short canonical form used inside the next subquestion, and $\tilde{e}^h$, a fully disambiguated form used inside the next search query. For example, an accepted answer "Window Rock, Arizona" is represented as $e^h =$ "Window Rock" and $\tilde{e}^h =$ "Window Rock, Arizona, the capital of the Navajo Nation." This separation keeps questions readable while making retrieval queries unambiguous.

At hop h, the policy state contains the seed image, the running merged question $\bar{q}^h$, e^h, $\tilde{e}^h$, the accepted answer history, the latest harness observation, and the remaining hop/turn budget. Its two actions and the corresponding fixed-harness responses are summarized below.

A.2 A Recorded QA Generation Trajectory

We report one completed 5-hop trajectory from a recorded QA Generator rollout. Its final question received $3/8$ correct Solver rollouts, making it challenging but not uniformly unsolved. Each displayed query, evidence excerpt, accepted model proposal, and harness merge is copied from the trace. We show only the accepted path; rejected candidates are omitted and do not receive QA policy credit. Construction-time source URLs are also omitted because the Solver does not receive them.

Table 7: QA Generator state transitions. Harness observations are appended as user turns and are always masked from the policy loss.

Stage	Trainable QA Generator output	Fixed harness observation
Search	`action`, `thought`, `tool=search`, and `query`; the query must contain $\tilde{e}^h$.	Retrieved and filtered evidence chunks, followed by a request to propose one supported next-hop QA pair.
Proposal	`action`, `thought`, `question`, `answer`, `evidence`, and `source`; the question must depend on e^h.	Either an acceptance containing $\bar{q}^{h+1}$, e^{h+1}, and $\tilde{e}^{h+1}$, or a rejection diagnosis that preserves the last accepted state.

A.2.1 Monument Valley → Window Rock Navajo Tribal Park



Seed image

Seed QA. "Which protected area is the place in the image located in?" → "Monument Valley Navajo Tribal Park."
Recorded Solver group. $3/8$ correct rollouts.

$h = 2$ **Query.** "Monument Valley Navajo Tribal Park, located in which U.S. state?" **Evidence** E_1**.** "The valley is considered sacred by the Navajo Nation, which administers the area as Monument Valley Navajo Tribal Park." **Accepted QA.** "Which Native American nation administers Monument Valley Navajo Tribal Park?" → "Navajo Nation." **Merge.** "Which Native American nation administers the protected area where the place in the image is located?"

$h = 3$ **Query.** "Navajo Nation, the Native American nation that administers Monument Valley Navajo Tribal Park, what is its official capital or administrative center?" **Evidence** E_1**.** "The seat of government is located in Window Rock, Arizona." **Accepted QA.** "What is the official capital or administrative center of the Navajo Nation?" → "Window Rock, Arizona." **Merge.** "What is the official capital or administrative center of the Native American nation that administers the protected area where the place in the image is located?"

$h = 4$ **Query.** "Window Rock, Arizona, the capital of the Navajo Nation, what is the name of its official tribal government building?" **Evidence** E_4**.** "This small park established sometime around 1936, features the graceful red sandstone arch for which the capital is named Tséghåhoodzání (Window Rock)." **Accepted QA.** "What is the name of the red sandstone arch for which the capital of Window Rock, Arizona, is named?" → "Tséghåhoodzání." **Merge.** "What is the name of the red sandstone arch for which the capital of the Native American nation that administers the protected area where the place in the image is located is named?"

$h = 5$ **Query.** "Tséghåhoodzání, the red sandstone arch in Window Rock, Arizona, capital of the Navajo Nation, what is the official name of the park where it is located?" **Evidence** E_1**.** "This small park established sometime around 1936, features the graceful red sandstone arch for which the capital is named Tséghåhoodzání (Window Rock). The park, officially known as Window Rock Navajo Tribal Park & Veterans Memorial, is located adjacent to the Navajo Nation government offices in Window Rock, Arizona." **Accepted QA.** "What is the official name of the park where the red sandstone arch Tséghåhoodzání is located?" → "Window Rock Navajo Tribal Park & Veterans Memorial." **Final merged task.** "What is the official name of the park where the red sandstone arch, for which the capital of the Native American nation that administers the protected area where the place in the image is located is named, is itself located?" → "Window Rock Navajo Tribal Park & Veterans Memorial."

Table 8: Fixed QA-harness execution order. Model-call failure behavior is reported explicitly; verifier availability is monitored during training.

#	Stage	Operation	Failure behavior
1	Retrieve and filter	Text search returns at most 6 results. The filter may read up to 6,000 characters per result and exposes at most 6 evidence chunks of 1,200 characters each to the QA policy; raw text retained internally is capped at 8,000 characters per result.	An empty/failed search rejects the attempt. If the filter call is unavailable or malformed, bounded raw snippets are used; an explicit empty filtered set rejects the attempt.
2	Structural and rarity gates	Require nonempty question, answer, and evidence; reject a repeated answer or an answer appearing in more than 3 retrieved chunks.	Deterministic rejection.
3	Entity continuity	Compare the proposed question against both the short current entity e^h and its disambiguated form $\tilde{e}^h$, checking that the question truly depends on the same referent.	A valid negative verdict rejects; an unavailable or malformed model verdict does not veto the proposal.
4	Proposal quality	Rule-based and model-assisted checks test the proposal's supplied evidence for support, and test uniqueness, temporal stability, entity necessity, stand-alone wording, and non-final answer linkability.	Rule violations reject deterministically; an unavailable or malformed model verdict does not veto the proposal.
5	Merge and normalize	Replace the previous explicit entity in the new question with the running description; return the merged question together with e^{h+1} and $\tilde{e}^{h+1}$.	Missing or unparsable merged output rejects the attempt.
6	Merged-QA coherence	Check whether the merged question remains self-contained and has the proposed answer as its unique intended resolution.	A valid negative verdict rejects; an unavailable or malformed verdict does not veto the proposal.
7	Parametric direct-answer gate	Ask the frozen helper to answer the merged question without web access. The proposal is rejected when its expected answer appears in that response under case-insensitive substring matching.	A failed helper call does not veto the proposal.

B QA Harness Checks and Feedback

The QA harness is fixed throughout co-training. Its validation and merging stages follow MTA-Agent (Peng et al., 2026); our adaptation exposes structured accept/reject feedback to the online QA policy and masks rejected action spans. The harness uses retrieval together with fixed programmatic and model-based checks, but its parameters are not optimized and no gradient flows through it. The main run uses Serper [1] for text retrieval and routes all model-assisted harness calls through the same frozen Claude Sonnet 5 endpoint on AWS Bedrock. Calls use a 60-second timeout and at most 3 API attempts. The ordered checks are detailed in Table 8.

For an accepted proposal, the harness appends a user-turn observation containing $\bar{q}^h$, a^h, and an acceptance message. This is the state used for the next Generator decision. A rejected proposal instead produces diagnostic feedback, allowing another attempt while preserving the previous accepted state. Both types of feedback remain visible in the context. However, only the search action and QA proposal associated with an accepted hop receive the QA policy-gradient signal. Tokens from rejected attempts are masked from the QA loss.

B.1 Recorded Harness Feedback

The following boxes reproduce the decision-bearing content returned to the QA Generator in recorded rollouts, with field names normalized for clarity. We omit generic action-schema and budget instructions, but retain the query, evidence, validation decision, and updated state. Each message is appended as a user turn: it is context for the next policy decision rather than a training target.

Search feedback. The harness returns filtered evidence and switches the policy to the next-hop proposal stage. The following is from the recorded trajectory in Appendix A.2.

[1] https://serper.dev/

Table 9: External QA-quality audit of generated tasks. Claude 4.8 Opus audits 512 tasks from each QA checkpoint under the same fixed rubric. Values are the percentage of tasks judged to satisfy each criterion; higher is better.

Audit criterion	Q_0	Q_{50}
Valid generated task	90.0%	92.0%
Evidence / chain support	70.3%	80.3%
Answer unique and stable	81.3%	83.0%
Self-contained question	90.2%	93.5%
No answer or evidence leakage	87.9%	89.8%
Requires the seed image	73.8%	73.0%

`[TOOL_FEEDBACK] search` succeeded for Hop 3/5.
Query: *Navajo Nation, the Native American nation that administers Monument Valley Navajo Tribal Park, what is its official capital or administrative center?*
Filtered evidence (excerpt): *The seat of government is located in Window Rock, Arizona.*
`Now propose ONE supported sub-question using only this evidence.`

Accepted-hop feedback. Once a proposal passes validation and merging, the harness returns the accepted QA pair, the merged question, and the entity to explore next.

`[SUCCESS]` Hop 3/5 completed.
Used result: Evidence 1
Accepted QA proposal: "What is the official capital or administrative center of the Navajo Nation?" → "Window Rock, Arizona."
Merged question (after this hop): "What is the official capital or administrative center of the Native American nation that administers the protected area where the place in the image is located?"
Next entity to explore: "Window Rock, Arizona."

Rejected-hop feedback. If validation fails, the harness preserves the last accepted state, provides a specific diagnosis, and requests a different search angle. The following recorded White Tower proposal is rejected as too generic.

`[FAILED]` Hop 3/5 – your sub-question proposal failed verification.
Root cause: `qa_verification_failed: vague_or_generic_question`
Recommendation: Try a different search angle and propose a more specific, evidence-supported question.
Rejected proposal: "What architectural style does the White Tower at the Tower of London, England, exemplify?" → "Norman style architecture."
Reason: Asking "what architectural style" is a generic category question, and the answer is not unique to the White Tower.
Preserved state: The current entity is still "White Tower." Choose a different query.

The failed search–proposal span remains visible in context but is masked from the QA loss. Only a search action and proposal that form an accepted hop contribute to the QA GRPO update.

B.2 External Quality Audit of Generated QA Tasks

The rollout-success target alone does not guarantee that a QA Generator avoids ambiguous or otherwise shortcut tasks. We therefore conduct an external LLM-based quality audit with Claude 4.8 Opus, separately from the QA harness and Solver. For each of Q_0 and Q_{50}, the audit assesses 512 generated tasks under the same fixed rubric. The evaluator inspects each final task and its recorded construction trace, and returns a binary judgment for every criterion below.

Audit criteria. *Valid generated task* requires a usable final question–answer pair. *Evidence / chain support* requires that the cited evidence supports each accepted hop and that the merged chain preserves the intended answer. *Answer unique and stable* rejects plausible alternative answers or time-sensitive answers without an explicit date. *Self-contained* checks that the final question can be

Table 10: Learning boundary in the online QA construction loop.

Component	Input → output	Learning signal
QA Generator	State → tool and arguments, or proposed (q^h, a^h) with evidence citation	QA GRPO with scalar r_i^{Q}
Fixed harness	Tool request → bounded evidence; proposal → accept/reject feedback and merged question	Fixed; no updated parameters, reward, or gradient
Solver	Seed image and final task → K tool-use rollouts	Solver GRPO with rollout correctness c_{ij}

interpreted without the hidden construction state. *No answer or evidence leakage* checks that intermediate answers and minting evidence do not appear in the final task. Finally, *requires the seed image* tests whether the image remains necessary to answer the final question.

As Table 9 shows, all five structural quality checks increase from Q_0 to Q_{50}: validity, evidence/chain support, answer uniqueness, self-containedness, and anti-leakage. In particular, evidence/chain support rises by 10.0 percentage points and self-containedness by 3.3 points. Image dependence is nearly unchanged, at 73.8% for Q_0 and 73.0% for Q_{50}. This audit does not rule out every possible shortcut or establish that every mixed-outcome group tests the intended reasoning capability. It does, however, find no detectable quality degradation from QA training in the direct signatures of ambiguity, unsupported chains, answer leakage, or image irrelevance.

B.3 Learning Boundary of the Fixed Harness

The harness is a fixed task-construction environment, not a second policy that generates the training questions. It contains retrieval backends, programmatic filters, and fixed model-assisted checks, all held unchanged throughout a run. In particular, it never selects the next search action or its arguments, and it does not author the candidate single-hop QA pair: these are outputs of the trainable QA Generator. Table 10 separates the responsibilities and learning signals.

Token-level QA credit assignment. The harness may return filtered evidence, a diagnostic rejection, or a merged question, but each is appended as an observation rather than a target completion and is always masked from the QA loss. For an accepted hop, the entire Generator-produced assistant span—reasoning, tool selection, tool arguments, and the proposed question, answer, and evidence citation—remains trainable. If the hop is rejected, the corresponding span is masked starting from that attempt's tool selection; this masks its reasoning, arguments, and proposal together, while keeping the content visible as context for the next attempt. A later rejection never retroactively masks earlier accepted hops. If generation stops before the target depth but has accepted earlier hops, the current merged partial QA task is retained and those accepted spans remain trainable.

Rewards and update order. For each generated task, the $K = 8$ binary Solver outcomes are first reduced to p_i. With the main target $\tau = 0.5$, the QA Generator receives the single scalar Goldilocks reward

$$r_i^{\mathrm{Q}} = \begin{cases} 1 - 2|p_i - 0.5|, & 0 < p_i < 1, \\ 0, & p_i \in \{0, 1\}. \end{cases}$$

The $N_{\mathrm{QA}} = 8$ candidate trajectories from one seed form one QA GRPO group. Consequently, a candidate with reward zero is not automatically dropped: it can receive a negative relative advantage when other candidates from that seed have higher reward. The main 8B run keeps only Solver groups with $0 < p_i < 1$ and uses the individual binary correctness values c_{ij} as their terminal rewards; it enables neither a function-call format gate nor hop-level partial credit. Each iteration executes QA rollout, harness construction, K Solver rollouts per task, correctness and QA-reward calculation, then the QA update and the Solver update. Both updates therefore use the same pre-update rollout batch; the QA update does not regenerate the Solver data within that iteration.

Information isolation. The Solver receives the seed image and final merged question, but not the Generator's intermediate QA pairs, their answers, minting evidence, or mint source URLs. Answer-bearing mint URLs are recorded only as blocked retrieval URLs. The image URL is used solely

as a handle for image tools such as reverse image search, OCR, and cropping; no source HTML URL or other seed provenance metadata is provided to the Solver. Thus, model-assisted checks can control task quality, but they do not provide the Solver's answer or share a trainable representation, optimizer, or gradient with either policy. The QA Generator sees only its harness feedback and the scalar r_i^{Q}, never Solver tokens, hidden states, or gradients. The two policies have separate actor parameters and optimizers; the main run does not enable a KL/reference-policy loss for either policy.

Table 11: Solver tool-call interfaces and representative responses appended to the next Solver turn. The first five response excerpts are from recorded validation trajectories. The cropping response is from a successful live tool call and includes an image block in addition to its text metadata.

Tool	Arguments	Representative returned response
Web search	`query`	`Tool: web_search` `Response: The building was constructed to display and promote local products, commerce, and industry.`
Web image search	`query`	`Tool: image_search` `Response: Search returned web image results about Payette Lake near McCall, Idaho. A geographic source identifies its outflow as the Payette River.`

Web-page reader	`url`	`Tool: content_extraction`
		`URL: en.wikipedia.org/wiki/Parc_de_Belleville`
		`Response: Page topic: Parc de Belleville, an urban park in Paris.`

Reverse image search	`image_url`; optional `query`	`Tool: reverse_image_search` `Response: The image most strongly matches the Petit Trianon at the Château de Versailles. Visual matches include photos and pages associated with this landmark.`

OCR	`image_url`	`Tool: ocr_tool`
		`Response: Text found in image:`
		`Sign: KOMORA WARSZAWA XIX W. THE WARSZAWA CHAMBER (19TH CENTURY)`

Image cropping	`image_url`, `crop_area`	`Tool: image_cropping`
		`Output: Reasoning: Crop tightly around the central ruined dome-shaped building.`
		`Original size: (800, 564); Crops (1): ruined dome building | bbox=[...]`
		`[cropped image attached]`

C Solver Interaction and Tool Interfaces

For each Solver input, a rollout starts from its seed image and final merged question. At every ReAct step, the Solver emits a reasoning trace, an action, a `should_stop` decision, and a confidence score. An action specifies a tool, its arguments, a short action description, and the expected observation. The observation server executes the tool call and appends the result to the interaction history before the next decision. The rollout finishes when the Solver returns its final answer, when `should_stop` is true with confidence above 0.7, or when it reaches the $T = 10$-step budget. The final answer is emitted in a designated boxed-answer field for evaluation.

For every question q_i, we sample $K = 8$ complete rollouts. We first apply answer normalization and matching, and use a fixed semantic judge only for unresolved aliases or paraphrases. This yields the binary terminal correctness c_{ij} of rollout j. Its group rate p_i determines the Goldilocks score used by the QA Generator and whether the Solver group is retained for GRPO. Tool observations are part of the rollout context and are masked from the policy loss.

C.1 Tool Palette

The Solver uses the following 6 callable tools in the reported multi-tool environment.

Table 12: Main 8B co-training configuration.

Setting	Value
Seed contexts per iteration	8
QA trajectories per seed	$N_{\text{QA}} = 8$
Target construction depth	5 hops (1 seed + 4 extensions)
Solver rollouts per QA input	$K = 8$
Configured QA / Solver `max_turns`	16 / 10
Search attempts / hop	3
Prompt / response token limits	8,192 / 24,576
Co-training iterations	50
Learning rate	10^{-6}
QA sampling temperature / top-p	1.0 / 0.95
Default QA target	$\tau = 0.5$

D Training and Evaluation Details

D.1 Seed-Pool Construction

We select the one-hop starting contexts using the seed-filtering protocol of MTA-Agent (Peng et al., 2026). A source item must have an image-dependent question, a visually correct answer, and a single specific answer entity. We discard the previously generated extensions, evidence, and tool trajectory and retain only (I, q^1, a^1).

The fixed pool contains 1,019 contexts: 398 from InfoSeek, 300 from FVQA-train, 190 from InfoVQA, and 131 from OK-VQA. Although the source records also store a disambiguated answer field, the online data loader passes only I, q^1, and a^1 to the QA Generator; no source chain or answer metadata is used. Consequently, the comparison to MTA-Agent starts from its filtered one-hop contexts while requiring `MDS-Zero` to construct every training-time multi-hop task online.

D.2 Main Co-training Configuration

Both policies begin from separate copies of `Qwen3-VL-8B-Instruct`. The fixed seed pool combines the four public one-hop VQA sources listed in Section 4.1.1. Each seed supplies an image and one question–answer pair. It contains neither multi-hop questions nor retrieved evidence. The following configuration is used for the main 8B co-training run.

Each iteration therefore contains $8 \times 8 = 64$ QA-generation trajectories and schedules $8 \times 8 \times 8 = 512$ Solver rollout inputs. The QA Generator and Solver use separate parameters, optimizers, and GRPO updates; the main run does not enable a KL/reference-policy loss. The only Solver-to-Generator signal is the scalar rollout-derived score; no trajectory, hidden state, or gradient is shared between the two policies.

D.3 Online Data and Workload Accounting

To make the online training budget explicit, we distinguish the fixed one-hop seed pool from the QA tasks constructed during co-training. The main 8B run samples 400 seed-context draws from a fixed pool of 1,019 public image-grounded VQA contexts: 398 InfoSeek, 300 FVQA-train, 190 InfoVQA, and 131 OK-VQA examples. Each source item contains an image and one-hop QA pair, with no retrieved evidence, multi-hop question, or tool trajectory.

Compute and service footprint. Table 14 reports the end-to-end resource use of this same main 8B run.

D.4 Answer Scoring and LLM Adjudication

We evaluate the final Solver using the same ReAct interaction protocol as at training time and report accuracy. For each benchmark item, we extract the final boxed answer when present (otherwise

Table 13: Online data and workload accounting for the main 8B run.

Quantity	Main 8B run
Seed-context draws	$50 \times 8 = 400$
QA-generation trajectories attempted	$50 \times 8 \times 8 = 3{,}200$
Scheduled Solver rollout inputs	$3{,}200 \times 8 = 25{,}600$
Pre-generated multi-hop QA corpus	0
Additional SFT updates	0

Table 14: Verified allocation, timing, and external-service workload for the main 8B run.

Run allocation			
Metric	Value	Metric	Value
Completed iterations	50	Hardware	2 nodes, 16 NVIDIA H200 GPUs
End-to-end wall-clock	16:21:48	Allocated compute	32.727 node-h; 261.813 H200 GPU-h
Device memory	140.401 GiB/GPU	Peak allocated memory	QA: 128.757 GiB; Solver: 130.512 GiB
		Peak reserved memory	QA: 133.992 GiB; Solver: 133.996 GiB
Exclusive top-level timing		**External-service workload**	
Rollout phase	14.048 h	QA search requests	17,000 (16,299 succeeded; 701 failed)
Reward and group filtering	0.017 h	Recorded QA verifier calls	41,411 (4 fail-open)
QA / Solver optimization	0.624 h / 1.298 h	Solver tool calls	79,103
Initialization, checkpoints, and other overhead	0.376 h	Solver LLM-judge calls	20,758 (20,599 returned; 159 failed)

the final response text) and first apply a deterministic, normalized match against the set of acceptable reference answers. This rule stage handles differences such as capitalization, whitespace, and standard multiple-choice formatting.

Only when this match fails do we invoke the semantic judge, `gpt-5-mini`. The judge receives the research question, all accepted reference answers, and the model's extracted final answer. It is instructed to decide whether the candidate conveys the same core information as a reference, allowing harmless aliases, paraphrases, and formatting variation, and returns a JSON verdict with a Boolean correctness field and a brief reason. A response is counted as correct if either the rule stage or the semantic judge returns a positive verdict.

Human audit. To assess the reliability of LLM adjudication, we manually re-evaluated 100 LLM-judged answers and found 98% agreement between the semantic judge and human correctness labels.