BEYOND THE CAPABILITY BOUNDARY: ZEROTH-ORDER OPTIMIZATION FOR SELF-EVOLVING LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Self-evolving methods improve the capabilities of LLM agents by sampling trajectories from the underlying LLMs and learning from these trajectories. However, these methods struggle to learn beyond the inherent capability boundary of the agents, since successful trajectories can be rare on difficult examples for further improvements. In this paper, we propose a zeroth-order self-evolution framework that enables agents to learn beyond their capability boundary by perturbing LLM parameters to adapt to difficult examples without any trajectory annotations. Specifically, we perturb LoRA parameters of LLMs, run the agent, compute the losses under the perturbed and original parameters, and use the loss difference to estimate gradients and further update the LoRA parameters. We sample trajectories using the updated LLMs for supervised fine-tuning to break through the capability boundary of the agents, forming a closed self-evolution loop. We introduce a parallel perturbation inference mechanism and an adaptive lookup mechanism to reduce time consumption in zeroth-order optimization, with an answer perplexity loss that provides smooth and stable zeroth-order loss values. Experiments on multiple deep research benchmarks show that our method obtains substantially more successful trajectories and consistently outperforms strong baselines, especially on difficult examples.

1 INTRODUCTION

Large language model (LLM) agents have shown strong potential in complex information-seeking and reasoning tasks, such as Deep Research (Yao et al., 2023; Schick et al., 2023; Qin et al., 2024). This task requires agents to retrieve information, integrate evidence, reason over intermediate results, and generate a final answer (Nakano et al., 2021; Yao et al., 2023). Since annotating high-quality intermediate reasoning trajectories is expensive for training, self-evolving LLM agents have emerged and rapidly attracted increasing attention (Shinn et al., 2023; Madaan et al., 2023; Gou et al., 2024), which sample reasoning trajectories by themselves and use the trajectories for further training via supervised fine-tuning (SFT) or reinforcement learning (RL) (Wu et al., 2025a; Sun et al., 2025). However, existing self-evolving LLM agents struggle to learn beyond their inherent capability boundary. On difficult examples, the agents may struggle to sample correct trajectories for further improvement.

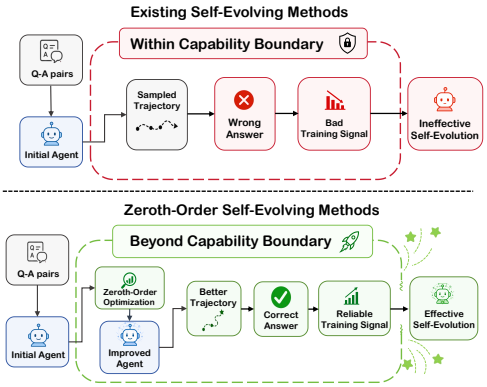


Figure 1: Comparison between existing self-evolving methods and our zeroth-order self-evolving method.

In this paper, we propose a zeroth-order self-evolution framework that enables agents to learn beyond their capability boundary without annotated trajectories. Specifically, for each difficult instance, we attach instance-specific LoRA modules and optimize them using zeroth-order estimation. We randomly perturb the LoRA parameters, compute the final-answer losses under both the original and perturbed parameters, and use the resulting loss differences to construct gradient estimates for

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.

updating the LoRA modules. Since this process only relies on answers and requires no annotated trajectories, it enables optimization without process supervision. The agent equipped with the optimized LoRA module is capable of generating successful trajectories. These newly discovered successful trajectories then serve as high-quality data for subsequent SFT to push the agent beyond its capability boundary, forming a closed self-evolution loop, as shown in Figure 1.

To realize this goal, we need to address two key challenges. First, agentic trajectory sampling is highly expensive. Trajectories involve multiple rounds of tool use and environment interaction, such as retrieval, browsing, filtering, and synthesis. These operations are time-consuming and sensitive to tool instability, latency, and environmental variation. Moreover, since zeroth-order optimization requires evaluating multiple random perturbations, it requires repeatedly sampling full agent trajectories, further introducing prohibitive sampling overhead. Second, without intermediate process (token-level) supervision, direct final-answer rewards, such as answer similarity or correctness signals, are often noisy and sparse, leading to unstable optimization.

To address the first challenge, we design a parallel perturbation inference mechanism. Since different perturbation branches share the same backbone computation, the expensive backbone output only needs to be computed once, while multiple lightweight LoRA perturbations can be evaluated in parallel. This significantly accelerates zeroth-order optimization with limited extra memory cost. We further design an adaptive lookup mechanism that caches recent tool-call results and adaptively uses exact or similarity-based matching according to query type, reducing repeated tool calls and compressing sampling time.

To address the second challenge, we introduce an answer perplexity loss as a process-level surrogate signal derived only from query and answer pairs. After the model generates a reasoning trajectory, we condition on the trajectory to compute the negative log-likelihood of the reference answer, and use it as the scalar loss for zeroth-order optimization. Compared with sparse final-answer rewards, this loss provides smoother and more discriminative feedback for different trajectories, improving optimization stability.

Experiments on multiple deep research benchmarks show that our method consistently improves final task performance over strong baselines. Compared with existing deep-research agents that rely on trajectories annotated by stronger models such as GPT, our agent achieves better performance, demonstrating that our method can break through the capability boundary of the agent. Further analysis shows that our method discovers substantially more successful trajectories, particularly on difficult examples where the original model struggles, further validating its effectiveness. Our contributions are summarized as follows.

- We propose a zeroth-order self-evolution method for LLM agents, which enables agents to learn beyond their capability boundary by perturbing LLM parameters to adapt to difficult examples without any trajectory annotations.
- We introduce a parallel perturbation inference mechanism, an adaptive lookup mechanism, and an answer perplexity loss to improve efficiency and optimization stability.
- We conduct systematic experiments to verify the effectiveness of our method and release the sampled high-quality trajectories and trained agents to facilitate future research.

2 Related Work

2.1 Self-Evolving Agent

Self-evolution refers to an iterative process in which a system improves by collecting, refining, and reusing its own experience (Tao et al., 2024). Reflexion first developed this idea to LLM agents by using task feedback to guide later agent trials (Shinn et al., 2023). Subsequently, self-evolving methods are developed along two lines, *i.e.*, non-parametric and training-based methods. Non-parametric methods improve the agent during inference by reusing feedback, memory, or tool-based corrections, such as Self-Refine (Madaan et al., 2023) using self-generated feedback to revise model outputs, CRITIC (Gou et al., 2024) correcting responses with external tools, and ExpeL (Zhao et al., 2024) storing past task experience as reusable knowledge. Training-based methods further convert self-generated experience into model updates. WebDancer (Wu et al., 2025a), SimpleDeepSearcher (Sun et al., 2025), and SE-Agent (Guo et al., 2026) construct and improve self-generated trajectories for SFT or RL, and WebEvolver (Fang et al., 2025) introduces a co-evolving world model to provide

richer experience. Existing methods depend on the initial agent to sample trajectories, which is bounded by its capability boundary. Our method uses zeroth-order optimization to reshape the policy, enabling the model to discover trajectories beyond its capability boundary.

2.2 Zeroth-order Optimization

Zeroth-order optimization estimates gradients from loss values, making it suitable when gradients are inaccessible or costly (Sun et al., 2022). Zeroth-order methods mainly follow two directions. Variance-reduction methods aim to stabilize gradient estimations, such as SZVR-G Liu et al. (2018a) and ZO-SVRG (Liu et al., 2018b). Acceleration methods aim to improve query efficiency and convergence speed, such as random gradient-free minimization Nesterov & Spokoiny (2017) and ZO-AdaMM (Chen et al., 2019).

Recently, zeroth-order optimization has been extended to LLMs to reduce training time and memory cost by avoiding backpropagation (Malladi et al., 2023).Subsequent studies aim to improve the practicality for LLMs. ZO-LLM (Zhang et al., 2024) introduces block-wise zeroth-order updates to improve stability, and LOZO (Chen et al., 2025) exploits low-rank structures to reduce the cost and variance of zeroth-order estimation. However, existing zeroth-order methods have not been studied for LLM agents, which is challenging since both the trajectory generation and sparse supervision in LLM agents result in expensive trajectory sampling and unstable optimization. To fill this gap, we design the adaptive lookup mechanism and answer perplexity loss for efficiency and stability.

3 Method

We present the background of the proposed method. Then, we present the formulation of the proposed zeroth-order method for self-evolving LLM agents, which can learn beyond the capability boundary of agents. We introduce a parallel perturbation inference mechanism and an adaptive lookup mechanism to reduce time consumption, and an answer perplexity loss to improve optimization stability. The overall process is summarized in Figure 2.

3.1 Background

LLM Agents. We denote the LLM agent as a policy $\boldsymbol{\pi_\theta}$, where $\boldsymbol{\theta}$ denotes parameters. The agent interacts with a tool environment $\mathcal{T}$ consisting of search, webpage visiting, and code execution. At step t, the interaction history is

$$h_t = (q, r_1, a_1, o_1, \ldots, r_{t-1}, a_{t-1}, o_{t-1}), \tag{1}$$

where r_t is the reasoning text, a_t is an action, and o_t is the returned observation. For a query q, a complete rollout is denoted as

$$\tau = (r_1, a_1, o_1, \ldots, r_T, a_T, o_T, \hat{y}), \tag{2}$$

where $\hat{y}$ is the model-generated final answer. The trajectory distribution is computed as

$$p_{\boldsymbol{\theta}}(\tau \mid q, \mathcal{T}) = \prod_{t=1}^{T} \boldsymbol{\pi_\theta}(r_t, a_t \mid h_t) p_{\mathcal{T}}(o_t \mid a_t, h_t). \tag{3}$$

The environment transition $p_{\mathcal{T}}$ includes multiple tool actions. Since actions are discrete and the environment is non-differentiable, the mapping from $\boldsymbol{\theta}$ to τ is a black-box stochastic process.

Self-Evolution for LLM Agents. A self-evolving LLM agent updates itself solely based on query q and answer pairs y (QA pairs), without intermediate reasoning trajectories. The agent should acquire trajectories τ from its own interaction with $p_{\mathcal{T}}$. By denoting the training set as $\mathcal{D} = \{(q_i, y_i)\}_{i=1}^{N}$, the objective is

$$F(\boldsymbol{\theta}) = \mathbb{E}_{(q,y)\sim\mathcal{D}} \mathbb{E}_{\tau\sim p_{\boldsymbol{\theta}}(\cdot|q,\mathcal{T})} \left[\ell(\boldsymbol{\theta}; q, \tau, y)\right], \tag{4}$$

where ℓ denotes the loss of SFT or RL. In SFT, self-sampled trajectories are used to supervise intermediate processes and compute gradients. In RL, these are used to estimate rewards and update the policy. Regardless of the update paradigm, the agent must sample successful trajectories to improve its capability; failed trajectories may instead introduce harmful signals and degrade the model. Existing self-evolving methods can only sample trajectories from the agent's own policy, which restricts the generated trajectories to the model's current capability boundary.

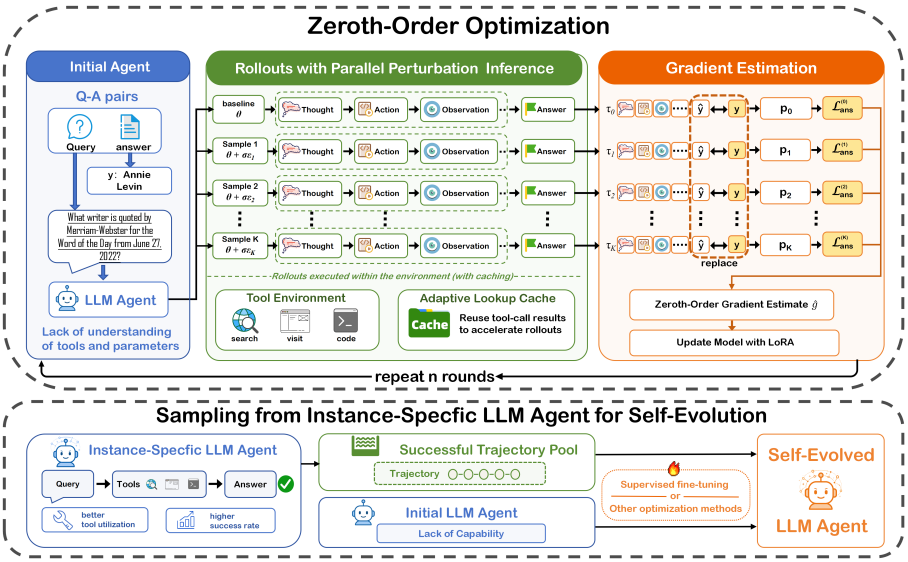


Figure 2: Overview of the proposed zeroth-order self-evolution pipeline. The method actively searches for successful agent trajectories around the current policy by perturbing LoRA parameters and evaluating complete rollouts with answer-only supervision.

Zeroth-order Optimization. Zeroth-order optimization aims to optimize a loss function when gradients are impractical to compute. Instead of relying on backpropagation, it treats the loss as a black box and estimates the gradient only from function evaluations. The core idea is to perturb $\boldsymbol{\theta}$ along a random direction $\boldsymbol{u}$, observe the change in the loss values, and use this finite-difference signal to construct an approximate gradient for updating the model, where the gradient is computed as

$$\hat{g}(\boldsymbol{\theta}) = \frac{F(\boldsymbol{\theta} + \mu \boldsymbol{u}) - F(\boldsymbol{\theta})}{\mu} \boldsymbol{u}, \tag{5}$$

where $F(\cdot)$ denotes a loss function, $\boldsymbol{u}$ is sampled from a standard Gaussian, and μ is the smoothing radius. With such estimated gradients and given a learning rate η, parameters $\boldsymbol{\theta}$ is updated as,

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \hat{\boldsymbol{g}}(\boldsymbol{\theta}_t), \tag{6}$$

thereby enabling optimization under black-box settings where only loss values are required.

3.2 Formulation

We use query and answer pairs as feedback and optimize an instance-specific LoRA module for each instance. QA-only supervision can not provide gradients, making direct parameter updates infeasible. We thus adopt zeroth-order optimization to perform black-box parameter updates based solely on loss values.

For each instance (q_i, y_i), the instance-specific LoRA module is denoted as $\boldsymbol{w}_i = [\boldsymbol{A}_i, \boldsymbol{B}_i]$. To optimize $\boldsymbol{w}_i$, we sample K random perturbation directions. For the k-th perturbation, the perturbed policy is denoted as $p_{\boldsymbol{\theta}^p_{i,k}}(\cdot)$, where the perturbed parameters $\boldsymbol{\theta}^p_{i,k}$ is computed as

$$\boldsymbol{\theta}^p_{i,k} = \boldsymbol{\theta} + (\boldsymbol{B}_i + \sigma \boldsymbol{\epsilon}^B_{i,k})(\boldsymbol{A}_i + \sigma \boldsymbol{\epsilon}^A_{i,k}), \tag{7}$$

where σ controls the perturbation scale. $\boldsymbol{\epsilon}^B_{i,k}$ and $\boldsymbol{\epsilon}^A_{i,k}$ are perturbations for parameters $\boldsymbol{B}_i$ and $\boldsymbol{A}_i$, respectively. The perturbed policy interacts with the same tool environment and generates a complete trajectory via

$$\tau^+_{i,k} \sim p_{\boldsymbol{\theta}^p_{i,k}}(\cdot \mid q_i, \mathcal{T}). \tag{8}$$

The corresponding loss of the perturbed model is computed by

$$\ell^+_{i,k} = \ell_{\text{ans}}(\boldsymbol{\theta}^p_{i,k}; q_i, \tau^+_{i,k}, y_i), \tag{9}$$

where ℓ_{ans} denotes a loss function computed solely from the final answer. We also need to utilize the current policy $\boldsymbol{\pi}_{\boldsymbol{\theta}}$ performs a complete rollout and produces a baseline trajectory $\tau^0_i \sim p_{\boldsymbol{\theta}}(\cdot \mid q_i, \mathcal{T})$,

and we compute the baseline loss on the un-perturbed policy as

$$\ell_i^0 = \ell_{\text{ans}}(\boldsymbol{\theta}; q_i, \tau_i^0, y_i). \tag{10}$$

The zeroth-order gradient of w_i is estimated by comparing the perturbed loss with the baseline loss,

$$\begin{aligned} \hat{g}(\boldsymbol{A}_i) &= \frac{1}{BK} \sum_{i=1}^{B} \sum_{k=1}^{K} \frac{\ell_{i,k}^+ - \ell_i^0}{\sigma} \boldsymbol{\epsilon}_{i,k}^A, \\ \hat{g}(\boldsymbol{B}_i) &= \frac{1}{BK} \sum_{i=1}^{B} \sum_{k=1}^{K} \frac{\ell_{i,k}^+ - \ell_i^0}{\sigma} \boldsymbol{\epsilon}_{i,k}^B. \end{aligned} \tag{11}$$

When a perturbation leads to a lower loss, the update moves the LoRA parameters toward that direction. When a perturbation increases the loss, the update moves away from it. We then update the LoRA parameters $\boldsymbol{w}_i$ with Adam. By the zeroth-order optimization process, the agent equipped with the optimized LoRA module is encouraged to generate successful reasoning trajectories that are beyond the capability boundary. Trajectories with correct final answers or sufficiently low answer loss are added to a high-quality trajectory buffer. This buffer provides the training data for the subsequent SFT, forming a closed self-evolution loop.

3.3 Parallel Perturbation Inference

In zeroth-order optimization, the model usually needs to perform forward evaluation under multiple perturbation directions. If we directly perturb the full model parameters, each perturbation requires a separate forward pass through the whole backbone, causing the computation cost to grow linearly with the number of perturbations.

For improving efficiency, we design a parallel perturbation inference that perturbs LoRA parameters in a parallel way. With the perturbed parameters in Eq. (7), as to the input $\boldsymbol{x}$, for the k-th perturbation, the model output can be written as

$$\boldsymbol{h}_k = \boldsymbol{\theta}\boldsymbol{x} + (\boldsymbol{B} + \boldsymbol{\epsilon}_k^B)(\boldsymbol{A} + \boldsymbol{\epsilon}_k^A)\boldsymbol{x}. \tag{12}$$

We observe that when perturbations only have effects in the LoRA branch, different perturbed models share the same backbone computation $\boldsymbol{\theta}\boldsymbol{x}$. Therefore, we only compute the backbone output $\boldsymbol{\theta}\boldsymbol{x}$ once and share it across all perturbation branches, which is illustrated in Figure 3.

In detail, for $1-k$ perturbations, we in parallel compute the right term in Eq. (12), *i.e.*, $(\boldsymbol{B} + \boldsymbol{\epsilon}_1^B)(\boldsymbol{A} + \boldsymbol{\epsilon}_1^A)\boldsymbol{x}, ..., (\boldsymbol{B} + \boldsymbol{\epsilon}_K^B)(\boldsymbol{A} + \boldsymbol{\epsilon}_K^A)\boldsymbol{x}$ are computed in parallel. Then, we add $\boldsymbol{\theta}\boldsymbol{x}$ to these terms as $\boldsymbol{\theta}\boldsymbol{x} + (\boldsymbol{B}+\boldsymbol{\epsilon}_1^B)(\boldsymbol{A}+\boldsymbol{\epsilon}_1^A)\boldsymbol{x}, ..., \boldsymbol{\theta}\boldsymbol{x}+(\boldsymbol{B}+\boldsymbol{\epsilon}_K^B)(\boldsymbol{A}+\boldsymbol{\epsilon}_K^A)\boldsymbol{x}$.

Due to the low-rank structures of $\boldsymbol{A}, \boldsymbol{B}$, their computation and memory cost are much smaller. In this way, we can evaluate multiple perturbations on top of a single backbone forward computation in parallel, which greatly reduces the sampling time cost of zeroth-order optimization.

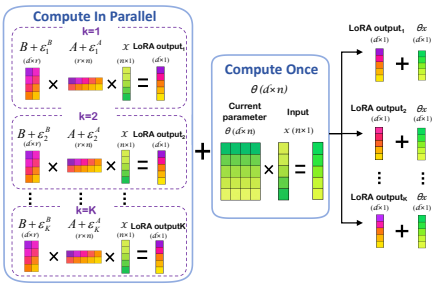


Figure 3: Illustration of parallel perturbation inference. The backbone output is computed once and shared across perturbation branches, while multiple LoRA perturbation branches are computed in parallel.

3.4 Adaptive Lookup Mechanism

Due to environmental and network conditions, tool calls can be quite time-consuming. Evaluating multiple perturbations for the same question often leads to repeated tool calls, such as similar `search` queries or visit the same webpages, which makes direct execution of every tool action unnecessarily expensive.

We therefore introduce an adaptive lookup mechanism to reuse tool observations. We maintain a shared pool of recent tool calls and returned results. For `search`, the lookup uses semantic query matching to handle surface-level variations. For `visit`, it uses stricter matching based on the URL and visit goal. Matched requests are served from the lookup pool, while unmatched requests are executed externally and registered for future reuse.

This design reduces redundant external interactions without changing the agent rollout format. It substantially lowers the time cost of multi-perturbation rollouts and improves the practicality of our zeroth-order self-evolution pipeline.

3.5 Answer Perplexity Loss

Zeroth-order gradient estimation relies on the loss values evaluated under perturbed parameters, making the design of the loss function critical for stability. Binary loss and BERT-based similarity loss are discontinuous, leading to noisy gradient estimation that causes the optimization process to oscillate and fail to converge.

We introduce an answer perplexity loss that uses trajectory-conditioned answer likelihood as dense supervision. Given a rollout τ for query q, the final answer is required to appear in the format `<answer>` $\hat{y}$ `</answer>`. We locate this answer span and replace the generated answer with the ground-truth answer. We denote the trajectory context generated by our agent as $\tau_{\backslash \text{ans}}$. The ground-truth answer is tokenized as $y = (y_1, \ldots, y_M)$. The introduced answer perplexity loss is

$$\ell_{\text{ans}}(\boldsymbol{\theta}; q, \tau, y) = -\frac{1}{M} \sum_{j=1}^{M} \log \boldsymbol{\pi}_{\boldsymbol{\theta}} \left(y_j \mid q, \tau_{\backslash \text{ans}}, y_{<j} \right). \tag{13}$$

The loss in Eq. (13) is computed only on ground-truth answer tokens. We do not supervise the generated reasoning, actions and observations.

Instead, the loss evaluates whether the sampled trajectory provides conditions under which the ground-truth answer is easy to predict.

Strictly, Eq. (13) is a token-normalized negative log-likelihood or cross-entropy loss. The corresponding answer perplexity is

$$\text{PPL}_{\text{ans}}(\boldsymbol{\theta}; q, \tau, y) = \exp\left(\ell_{\text{ans}}(\boldsymbol{\theta}; q, \tau, y) \right). \tag{14}$$

We optimize the log-space form because it is numerically stable and directly usable for loss-difference estimation. A lower loss means that the current trajectory makes the reference answer less surprising to the model. Therefore, even an incorrect rollout can still receive useful credit if it retrieves evidence that increases the likelihood of the ground-truth answer.

4 Experiments

4.1 Experimental Setup

Evaluation Benchmarks. We evaluate our method on two deep research benchmarks: GAIA (Mialon et al., 2024) and WebWalkerQA (Wu et al., 2025b). GAIA tests general assistant abilities including reasoning, browsing, multimodal handling, and tool use. WebWalkerQA focuses on website traversal and multi-page information extraction. These benchmarks are well suited to our study because their multi-step search difficulty provides a direct test of whether our discovered trajectories can produce stronger agents.

Training data. Starting from 304 high-quality QA pairs from the open-source WebShaper data (Tao et al., 2026).

Metrics. We report answer accuracy under the official evaluation protocol.

Comparison. We compare our method with the baseline ReAct framework (Yao et al., 2023). For controlled comparisons, all internal methods use the same backbone family and the same system configuration, including the search API, webpage parser, decoding setting, and tool budget. We further compare our method with existing deep-research agents, including Search-o1 (Li et al., 2025b), WebThinker (Li et al., 2026), WebDancer (Wu et al., 2025a), SimpleDeepSearcher (Sun et al., 2025), WebSailor (Li et al., 2025a), GRPO (Shao et al., 2024), and ARPO (Dong et al., 2026).

4.2 Main Results

Table 1 reports the main controlled results and contextual paper-reported comparisons. Results show that our method surpasses the baseline method, demonstrating the effectiveness of our method. Because the compared deep-research agents rely on more large-scale models (*e.g.* GPT) to generate

trajectories, these agents are easier to obtain strong results than self-evolution agents. Despite this disadvantage, our method still outperforms these agents, demonstrating its superiority and showing that it can break through the capability boundary of self-evolving agents.

Table 1: Main results under controlled evaluation and contextual paper-reported comparisons with recent deep-research agents. Paper-reported rows are grouped by framework/search scaffolds and training-based methods.

Backbone	Method	GAIA				WebWalkerQA			
		Level 1	Level 2	Level 3	Avg.	Easy	Medium	Hard	Avg.
Paper-reported framework/search scaffolds									
Qwen-2.5-7B	ReAct (Yao et al., 2023)	28.2	15.3	0.0	18.4	28.1	31.2	16.0	24.2
Qwen-2.5-7B	Search-o1 (Li et al., 2025b)	23.1	17.3	0.0	17.5	–	–	–	–
Qwen-3-8B	Search-o1	35.9	15.4	0.0	21.4	6.7	15.5	9.7	11.5
Qwen-3-8B	WebThinker (Li et al., 2026)	43.6	11.5	0.0	22.3	6.7	13.1	16.9	13.0
Paper-reported training-based methods									
Qwen-2.5-7B	WebDancer (Wu et al., 2025a)	41.0	30.7	0.0	31.0	40.6	44.1	28.2	36.0
Qwen-2.5-7B	SimpleDeepSearcher† (Sun et al., 2025)	–	–	–	36.9	–	–	–	–
WebSailor-7B	WebSailor† (Li et al., 2025a)	–	–	–	33.0	–	–	–	–
Qwen-3-8B	GRPO (Shao et al., 2024)	48.7	25.0	8.3	32.0	24.4	33.3	26.8	29.0
Qwen-3-8B	ARPO (Dong et al., 2026)	53.9	32.7	16.7	38.8	26.7	33.3	29.6	30.5
Controlled setting									
Qwen-3-4B	ReAct	33.3	15.1	5.0	19.6	25.0	17.5	18.3	19.5
	Ours-SFT	**42.8**	**24.2**	**10.0**	**28.3**	**30.0**	**26.4**	**29.2**	**28.2**
Qwen-3-8B	ReAct	35.9	17.3	8.3	23.3	8.9	16.7	18.3	15.5
	Ours-SFT	**56.4**	**42.3**	**41.6**	**47.5**	**31.4**	**36.8**	**35.2**	**34.8**

4.3 Trajectory Discovery and Loss Design

We first examine whether zeroth-order discovery produces useful supervision beyond the trajectories already available to the initial policy. We then analyze the loss used to guide this discovery.

4.3.1 Effect on difficult QA pairs

Among the 304 training examples, the baseline model correctly answers only 67 examples under Pass@1 and fails on the remaining 237, resulting in an initial success rate of 22.0%. After training with the proposed zeroth-order optimization method, the optimized agent correctly answers 164 examples under Pass@1, increasing the success rate to 53.9% and yielding an improvement of 31.9%. Results demonstrate that agents trained with the proposed method can effectively break through the capability boundary of the initial agent.

We further train the initial agent using the 67 successful trajectories sampled from the initial agent and evaluate the resulting self-evolved agent on the GAIA benchmark. As shown in Table 2, training on trajectories sampled from the initial model improves over the baseline agent, demonstrating the effectiveness of self-evolution. More importantly, our method significantly outperforms the two compared agents, demonstrating that our method can help sample high-quality trajectories and validating its effectiveness.

Table 2: GAIA accuracy comparison of the agents (Qwen-3-4B as the backbone) trained with different trajectory sources.

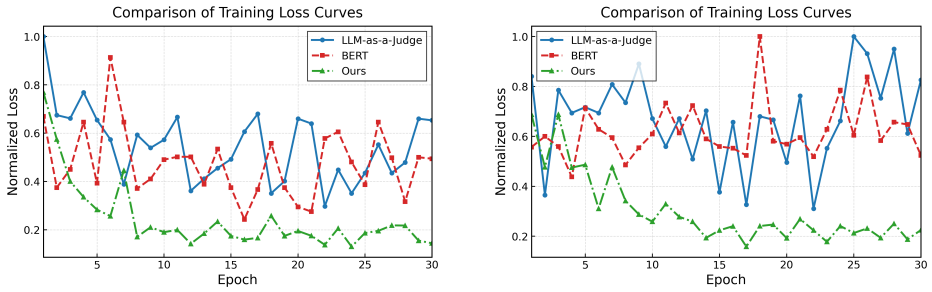
Agent.	Level 1	Level 2	Level 3	Avg.
Agent w/o training	33.3	15.1	5.0	19.6
Agent trained on initial trajectories	38.1	13.6	5.0	20.4
Agent trained on our trajectories	**42.8**	**24.2**	**10.0**	**28.3**

4.3.2 Loss convergence

We compare three loss functions: BERT-based semantic loss, LLM-as-a-Judge loss, and the proposed answer perplexity loss. The comparison is conducted on an easy example and a hard example with the same question under the same zeroth-order setting. Since these three losses have different scales, we normalize their values to the range of 0 to 1 for a fair comparison of their convergence.

Results are shown in Figure 4. On the easy example, all three losses can decrease during zeroth-order optimization, but our loss achieves a lower final loss value and leads to a more stable optimization

process. On difficult samples, both the LLM-as-a-Judge loss and the BERT-based loss fail to decrease, whereas our loss still converges normally with stable optimization dynamics. These results demonstrate that the proposed loss is well suited for zeroth-order optimization and is effective in providing reliable optimization signals.



(a) Results on the easy sample.

(b) Results on the hard sample.

Figure 4: Convergence comparison of different loss functions on easy and hard examples.

4.4 Capability Boundary Analysis

We separate the effect of loss-guided parameter updates from simply drawing more trajectories, with parameter-path diagnostics provided in Appendix A.3.

4.4.1 Best-of-N trajectory search

We further compare trajectory-discovery methods under matched complete-rollout budgets. The initial-policy baseline directly samples N trajectories. Random LoRA perturbation uses the same perturbation scale and perturbed rollouts as our method but does not estimate gradients or update parameters. Answer-conditioned trajectory search provides the reference answer as additional context and searches for a supporting trajectory without parameter updates. An example is counted as solved if at least one of the trajectories obtained within the budget is correct. As shown in Table 3, our method achieves the highest success rate for every evaluated budget. The advantage becomes larger as the budget increases, indicating that iterative loss-guided updates are more effective than additional sampling or unguided perturbation.

4.4.2 Pass@N scaling

We evaluate Pass@N for $N \in \{1, 2, 4, 6, 8\}$, where an example is considered solved if at least one of the N sampled trajectories is correct. On the trajectory-discovery training set, we compare the initial policy with our optimized policy. On GAIA, we additionally include the RL-trained policy. As shown in Table 4, our method obtains the best result at every sampling budget. The gap remains when N increases, showing that the improvement is not restricted to Pass@1.

We further match the full rollout budget of zeroth-order optimization with $N_{\text{total}} = 45$ samples from the unchanged initial policy. On the training set, the initial policy solves 102 examples, while our method solves 194. Among the 97 examples newly solved by our method over the initial Pass@1 result, only 35 examples (36.1%) can be recovered by Best-of-N_{total} sampling from the initial policy. The remaining 62 examples are still unsolved after the same sampling budget. Moreover, the initial-policy result already plateaus at Pass@8, with Pass@8 identical to Pass@N_{total}. These results show that additional sampling alone does not account for the newly discovered trajectories under the evaluated budget. Additional parameter-path diagnostics are reported in Appendix A.3.

First-order and RL alternatives. On 50 randomly selected difficult examples, we compare fixed-trajectory first-order (FO) optimization, which backpropagates the answer loss through an initial failed rollout, and PPL-reward RL, which regenerates trajectories, with our ZO method. All use the same Qwen-3-4B backbone, tools, and complete-rollout budget. ZO solves 23 examples versus 16 for RL and 7 for FO (Table 5), supporting loss-guided updates across regenerated trajectories over these alternatives.

Table 3: Best-of-N trajectory-discovery success rate (%) under matched rollout budgets. All methods start from the same initial policy.

N	Initial policy	Random LoRA perturbation	Answer-conditioned trajectory search	ZO (Ours)
1	22.04	21.38	24.67	**26.97**
2	26.97	27.63	27.63	**30.26**
4	29.93	31.57	32.56	**41.19**
6	31.57	32.56	36.51	**50.32**
8	32.23	33.55	37.82	**53.61**

Table 4: Pass@N (%) on the trajectory-discovery training set and GAIA.

	Training		GAIA		
N	Initial policy	Ours	Initial policy	RL	Ours
1	22.0	**53.9**	19.6	25.2	**28.3**
2	26.9	**58.5**	25.2	32.0	**34.9**
4	30.2	**61.5**	28.1	36.9	**37.8**
6	32.5	**63.1**	30.1	36.9	**39.8**
8	33.5	**63.8**	31.1	39.8	**41.7**

Table 5: Difficult examples solved under a matched rollout budget ($n = 50$).

Initial-policy sampling	Fixed-trajectory FO	PPL-reward RL	ZO (Ours)
6 / 50	7 / 50	16 / 50	**23 / 50**

4.5 Efficiency

We evaluate the efficiency of the parallel perturbation inference and adaptive lookup mechanism.

Parallel perturbation inference. The maximum trajectory context length is fixed to 20k tokens. We set $K = 2$ and $K = 4$ for comparison. We measure the wall-clock time of unperturbed rollout, perturbed rollout sampling without the parallel mechanism, and the perturbed rollout sampling with our parallel mechanism. Results are shown in Table 6.

Results show that parallel perturbation inference reduces the required computational time. Although perturbed sampling is still more expensive than a single unperturbed rollout, the extra cost is much smaller than running each perturbation independently. This makes multi-direction zeroth-order estimation feasible under limited hardware.

Adaptive lookup mechanism. We then analyze the time saved by the adaptive lookup mechanism. Since different perturbations often lead to similar `search` queries or visit overlapping webpages, the lookup pool can reuse previous tool observations. We report the cache hit counts for `search` and `visit` under different perturbation numbers and optimization rounds. We estimate the saved time using an average latency of 3 seconds for each `search` call and 10 seconds for each `visit` call. Results are shown in Table 7. Results indicate that adaptive lookup reduces redundant external interactions while preserving exploration.

Table 6: Efficiency of parallel perturbation inference. Sequential time is estimated by multiplying the single-rollout time by K.

K	Single rollout	Seq. estimate	Parallel time	Reduction
2	164s	328s	232s	29.3%
4	164s	656s	305s	53.5%

Table 7: Cache hits and estimated saved time from the adaptive lookup mechanism.

K	Rounds	Search hits	Visit hits	Saved time
2	15	46	27	408s
2	30	78	45	684s
4	15	86	50	758s
4	30	150	84	1290s

5 Conclusion

We have presented a zeroth-order optimization method for self-evolving LLM agents that enables agents to learn beyond their capability boundaries by perturbing LLM parameters to fit difficult examples. The proposed method computes the losses under both perturbed and original parameters, uses the loss difference to estimate gradients, and updates the parameters accordingly. The updated LLM agent is then used to sample trajectories for supervised fine-tuning, forming a closed self-evolution loop. The introduced adaptive lookup mechanism and parallel perturbation inference mechanism reduce the time cost of zeroth-order optimization, while the proposed answer perplexity loss provides smooth and stable loss values for zeroth-order gradient estimation. Experiments on multiple deep research benchmarks demonstrate that our method obtains substantially more successful trajectories and consistently outperforms strong baselines. The comparison with existing methods demonstrates that our method can break through the inherent capability boundary of the agent itself.

AI USE STATEMENT

We used GPT-4o (`gpt-4o-2024-08-06`), together with rule-based checks, to filter 500 question–answer pairs from the open-source WebShaper dataset and retain 304 high-quality pairs for training. The agent trajectories used in our method were obtained through the zeroth-order optimization and rollout procedure described in the paper. We also used OpenAI Codex to review the manuscript, check consistency among text, tables, and figures, and assist with English wording and LaTeX revisions. The authors take responsibility for the research methodology, experimental results, and the final content of this work, including all text and artifacts prepared with assistance from generative AI.

REFERENCES

Xiangyi Chen, Sijia Liu, Kaidi Xu, Xingguo Li, Xue Lin, Mingyi Hong, and David Cox. ZO-AdaMM: Zeroth-order adaptive momentum method for black-box optimization. *Advances in neural information processing systems*, 32, 2019.

Yiming Chen, Yuan Zhang, Liyuan Cao, Kun Yuan, and Zaiwen Wen. Enhancing zeroth-order fine-tuning for language models with low-rank structures. In *International Conference on Learning Representations*, volume 2025, pp. 62581–62607, 2025.

Guanting Dong, Hangyu Mao, Kai Ma, Licheng Bao, Yifei Chen, Zhongyuan Wang, Zhongxia Chen, Jiazhen Du, Huiyang Wang, Fuzheng Zhang, Guorui Zhou, Yutao Zhu, Ji-Rong Wen, and Zhicheng Dou. Agentic reinforced policy optimization. In *The Fourteenth International Conference on Learning Representations*, 2026. URL `https://openreview.net/forum?id=TX4k7BF6aO`.

Tianqing Fang, Hongming Zhang, Zhisong Zhang, Kaixin Ma, Wenhao Yu, Haitao Mi, and Dong Yu. WebEvolver: Enhancing web agent self-improvement with co-evolving world model. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 8970–8986, 2025.

Katelyn Gao and Ozan Sener. Generalizing gaussian smoothing for random search. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, 2022. URL `https://proceedings.mlr.press/v162/gao22f.html`.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yujiu Yang, Nan Duan, Weizhu Chen, et al. Critic: Large language models can self-correct with tool-interactive critiquing. In *International Conference on Learning Representations*, volume 2024, pp. 57734–57811, 2024.

Yifu Guo, Jiaye Lin, Huacan Wang, Yuzhen Han, Sen Hu, Ziyi Ni, Licheng Wang, and Mingguang Chen. SE-agent: Self-evolution trajectory optimization in multi-step reasoning with llm-based agents. *Advances in Neural Information Processing Systems*, 38:116314–116341, 2026.

Kuan Li, Zhongwang Zhang, Huifeng Yin, Liwen Zhang, Litu Ou, Jialong Wu, Wenbiao Yin, Baixuan Li, Zhengwei Tao, Xinyu Wang, et al. WebSailor: Navigating super-human reasoning for web agent. *arXiv preprint arXiv:2507.02592*, 2025a.

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

Xiaoxi Li, Jiajie Jin, Guanting Dong, Hongjin Qian, Yongkang Wu, Ji-Rong Wen, Yutao Zhu, and Zhicheng Dou. Webthinker: Empowering large reasoning models with deep research capability. *Advances in Neural Information Processing Systems*, 38:120091–120131, 2026.

Liu Liu, Minhao Cheng, Cho-Jui Hsieh, and Dacheng Tao. Stochastic zeroth-order optimization via variance reduction method. In *Proceedings of the 35th International Conference on Machine Learning*, 2018a.

Sijia Liu, Bhavya Kailkhura, Pin-Yu Chen, Paishun Ting, Shiyu Chang, and Lisa Amini. Zeroth-order stochastic variance reduction for nonconvex optimization. *Advances in neural information processing systems*, 31, 2018b.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594, 2023.

Sadhika Malladi, Tianyu Gao, Eshaan Nichani, Alex Damian, Jason D Lee, Danqi Chen, and Sanjeev Arora. Fine-tuning language models with just forward passes. *Advances in Neural Information Processing Systems*, 36:53038–53075, 2023.

Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. Gaia: a benchmark for general ai assistants. In *International Conference on Learning Representations*, volume 2024, pp. 9025–9049, 2024.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*, 2021.

Yurii Nesterov and Vladimir Spokoiny. Random gradient-free minimization of convex functions. *Foundations of Computational Mathematics*, 17(2):527–566, 2017.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *International Conference on Learning Representations*, volume 2024, pp. 9695–9717, 2024.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36:68539–68551, 2023.

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

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Shuang Sun, Huatong Song, Yuhao Wang, Ruiyang Ren, Jinhao Jiang, Junjie Zhang, Fei Bai, Jia Deng, Wayne Xin Zhao, Zheng Liu, Lei Fang, Zhongyuan Wang, and Ji-Rong Wen. SimpleDeepSearcher: Deep information seeking via web-powered reasoning trajectory synthesis. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 13705–13720, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-335-7. doi: 10.18653/v1/2025.findings-emnlp.739. URL https://aclanthology.org/2025.findings-emnlp.739/.

Tianxiang Sun, Yunfan Shao, Hong Qian, Xuanjing Huang, and Xipeng Qiu. Black-box tuning for language-model-as-a-service. In *International Conference on Machine Learning*, pp. 20841–20855. PMLR, 2022.

Zhengwei Tao, Ting-En Lin, Xiancai Chen, Hangyu Li, Yuchuan Wu, Yongbin Li, Zhi Jin, Fei Huang, Dacheng Tao, and Jingren Zhou. A survey on self-evolution of large language models. *arXiv preprint arXiv:2404.14387*, 2024.

Zhengwei Tao, Jialong Wu, Wenbiao Yin, Pu Wu, Junkai Zhang, Baixuan Li, Haiyang SHEN, Kuan Li, Liwen Zhang, Xinyu Wang, Wentao Zhang, Yong Jiang, Pengjun Xie, Fei Huang, and Jingren Zhou. Webshaper: Agentically data synthesizing via information-seeking formalization. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=hld4TzJsnD.

Jialong Wu, Baixuan Li, Runnan Fang, Wenbiao Yin, Liwen Zhang, Zhenglin Wang, Zhengwei Tao, Ding-Chu Zhang, Zekun Xi, Robert Tang, et al. WebDancer: Towards autonomous information seeking agency. *Advances in Neural Information Processing Systems*, 38:120957–120985, 2025a.

Jialong Wu, Wenbiao Yin, Yong Jiang, Zhenglin Wang, Zekun Xi, Runnan Fang, Linhai Zhang, Yulan He, Deyu Zhou, Pengjun Xie, et al. Webwalker: Benchmarking llms in web traversal. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 10290–10305, 2025b.

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*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Yihua Zhang, Pingzhi Li, Junyuan Hong, Jiaxiang Li, Yimeng Zhang, Wenqing Zheng, Pin-Yu Chen, Jason D. Lee, Wotao Yin, Mingyi Hong, Zhangyang Wang, Sijia Liu, and Tianlong Chen. Revisiting zeroth-order optimization for memory-efficient LLM fine-tuning: A benchmark. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 59173–59190. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/zhang24ad.html.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024.

A Training Details and Supplementary Evaluation

This appendix complements the main experimental section with the training protocol, sensitivity and estimator choices, alternative optimization baselines, and stability and transfer checks of the trained policy.

A.1 Training Pipeline and Implementation

We conduct computational experiments on deep-research LLM agents. The evaluation benchmarks include GAIA and WebWalkerQA, and the training data is constructed from 304 filtered high-quality QA pairs from WebShaper. We compare our method with ReAct under controlled settings and also report contextual comparisons with existing deep-research agents. We evaluate answer accuracy following the official evaluation protocol of each benchmark.

The main model backbones used in our controlled experiments are Qwen-3-4B and Qwen-3-8B. All controlled methods use the same backbone family and the same system configuration, including the search API, webpage parser, decoding setting, and tool budget.

For zeroth-order optimization, we optimize instance-specific LoRA modules by sampling random perturbation directions and estimating gradients from answer-loss differences. We evaluate perturbation numbers $K = 2$ and $K = 4$ in the efficiency analysis. The maximum trajectory context length is fixed to 20k tokens. The answer perplexity loss is used as the main loss for zeroth-order trajectory discovery. For supervised fine-tuning, sampled high-quality trajectories are used as training data for the self-evolved agent.

We use existing software packages and tools for model inference, LoRA fine-tuning, benchmark evaluation, search, and webpage parsing. Specifically, we use vLLM for LLM inference, Modified LLaMA-Factory for LoRA-based fine-tuning, and the official evaluation scripts or protocols of GAIA and WebWalkerQA for answer accuracy evaluation. We use Google Serper as the search tool and Jina.visit for webpage parsing. All methods in the controlled comparison use the same implementation settings for these tools.

The experiments were run on NVIDIA RTX3090 GPUs. The total computational budget was approximately 1152 GPU hours, including zeroth-order trajectory discovery, supervised fine-tuning, and benchmark evaluation. The main training and inference infrastructure consisted of 8 NVIDIA RTX3090 24GB GPUs. We will release additional implementation details with the code and artifacts.

We report benchmark accuracies as single-run results unless otherwise specified. For trajectory discovery, we report the number of successful examples under Pass@1. Among the 304 training examples, the baseline model correctly answers 67 examples and fails on 237 examples, while the optimized agent correctly answers 164 examples. For efficiency analysis, we report rollout wall-clock time, cache hit counts, and estimated saved time.

A.2 Training Objective and Update Analysis

Analysis of Self-Evolution methods for LLM Agents. We distinguish a policy's finite-budget discovery boundary from the support of its trajectory distribution. The analysis below concerns instance-specific trajectory discovery, before the subsequent SFT stage. It establishes sufficient conditions for improved discovery, rather than a universal guarantee that a lower answer loss implies a correct rollout.

A.2.1 A finite-budget notion of capability

Fix a QA pair (q, y), a stationary tool environment, and a rollout horizon. Let $w \in \mathbb{R}^d$ collect the trainable LoRA factors, with the backbone frozen. Write $p_w(\tau)$ for the complete-rollout distribution and let $\mathcal{S}$ be the set of trajectories with a correct final answer. Define

$$s(w) = \Pr_{\tau \sim p_w}(\tau \in \mathcal{S}), \qquad D_n(w) = 1 - (1 - s(w))^n. \tag{15}$$

Here D_n is the probability of discovering at least one success in n independent rollouts from a fixed policy. For a target confidence $1 - \delta$, the instance is inside the (n, δ) discovery boundary precisely when

$$s(w) \geq s_\star(n, \delta) := 1 - \delta^{1/n}. \tag{16}$$

Thus, "beyond the initial capability boundary" means $s(w_0) < s_\star$, not necessarily $s(w_0) = 0$. Observing no success in a finite batch does not establish zero probability. Parameter updates can change $s(w)$, whereas extra sampling at fixed w only changes n. This definition formalizes discovery at a stated budget and confidence; it is not an estimate of $s(w_0)$ from the reported experiments.

A.2.2 Why complete-rollout perturbations capture policy changes

Let $f(w, \tau)$ be the answer loss in Eq. (13) and $F(w) = \mathbb{E}_{\tau \sim p_w} f(w, \tau)$. Assume differentiability, locally common support, and domination sufficient to interchange differentiation and expectation. Then

$$\begin{aligned} \nabla F(w) &= g_{\text{fixed}}(w) + g_{\text{traj}}(w), \\ g_{\text{fixed}}(w) &= \mathbb{E}_{\tau \sim p_w}[\nabla_w f(w, \tau)], \\ g_{\text{traj}}(w) &= \mathbb{E}_{\tau \sim p_w}[f(w, \tau) \nabla_w \log p_w(\tau)]. \end{aligned} \tag{17}$$

This identity follows by differentiating $\int p_w(\tau) f(w, \tau)\, d\tau$ with the product rule. Differentiation through a tool's internal implementation is unnecessary: its stationary transition law is part of p_w, while the policy controls which actions are taken.

A fixed-trajectory first-order update on a failed rollout τ^- uses

$$g_{\text{FO}} = \nabla_w f(w, \tau^-), \tag{18}$$

and omits the explicit distribution term in Eq. (17). Such an update can still change future behavior through shared parameters; the omission does not prove that first-order learning cannot discover new trajectories. Similarly, the correctness-reward estimator

$$\widehat{g}_{\text{RL}} = \frac{1}{M} \sum_{i=1}^{M} (R(\tau_i) - b) \nabla_w \log p_w(\tau_i) \tag{19}$$

is zero when every sampled reward equals the baseline b. This describes a sparse-feedback failure mode, not an impossibility result for RL with richer rewards or exploration.

Lemma 1 (unbiased gradient of the smoothed rollout objective). Let $u_k \sim \mathcal{N}(0, I_d)$ and obtain unbiased scalar evaluations Y_k^+ at $w + \sigma u_k$ and Y_k^0 at w, such that $\mathbb{E}[Y_k^+ \mid u_k] = F(w + \sigma u_k)$ and $\mathbb{E}[Y_k^0 \mid u_k] = F(w)$. Under Gaussian integrability and differentiation conditions, define

$$\widehat{g} = \frac{1}{K} \sum_{k=1}^{K} \frac{Y_k^+ - Y_k^0}{\sigma} u_k, \qquad F_\sigma(w) = \mathbb{E}_u F(w + \sigma u). \tag{20}$$

Then $\mathbb{E}\widehat{g} = \nabla F_\sigma(w)$. If ∇F is globally L_F-Lipschitz, then

$$\|\nabla F_\sigma(w) - \nabla F(w)\| \leq L_F \sigma \sqrt{d} =: b_\sigma. \tag{21}$$

Proof. Conditional unbiasedness and $\mathbb{E}u = 0$ remove the baseline term. Gaussian integration by parts gives $\sigma^{-1}\mathbb{E}[F(w+\sigma u)u] = \nabla F_\sigma(w)$, the standard smoothing identity (Nesterov & Spokoiny, 2017; Gao & Sener, 2022). The bias bound follows from $\mathbb{E}\|\nabla F(w+\sigma u) - \nabla F(w)\| \le L_F\sigma\mathbb{E}\|u\| \le L_F\sigma\sqrt{d}$. □

Crucially, the perturbed evaluation reruns the agent and therefore samples from $p_{w+\sigma u}$, rather than rescoring only a fixed trajectory. It consequently captures both terms of Eq. (17) in the small-smoothing limit. The identity does not require a successful rollout in every update batch. It is an unconditional expectation statement, not a claim of unbiasedness after conditioning on an all-failure batch. A shared baseline is allowed if the conditional-mean condition holds; any resulting correlation must be included in the estimator variance.

A.2.3 A SUFFICIENT CONDITION FOR CROSSING THE DISCOVERY BOUNDARY

Decreasing F alone is insufficient. The missing link is alignment between the surrogate-loss gradient and success probability. A useful sufficient condition in a region of interest is

$$\nabla F(w) = -a\nabla s(w) + r(w), \qquad a > 0, \quad \|r(w)\| \le \varepsilon. \tag{22}$$

This is an explicit assumption, not a consequence of answer perplexity. For example, it holds exactly with $\varepsilon = 0$ if expected loss equals $c - as(w)$, and approximately if its gradient differs by a bounded residual. The residual includes changes in answer likelihood that do not improve trajectory success. The theorem also requires a reachable improving direction in the LoRA parameterization.

Theorem 1 (finite-budget boundary crossing under aligned ZO updates). Consider T updates $w_{t+1} = w_t - \eta\widehat{g}_t$ and a failure probability $\alpha \in (0,1)$. Conditional on the history before each update, assume Lemma 1, Eq. (22), and

$$\|\nabla s(w_t)\| \ge m > 0, \qquad \mathbb{E}[\|\widehat{g}_t - \nabla F_\sigma(w_t)\|^2 \mid \mathcal{H}_t] \le V. \tag{23}$$

Assume these conditions hold throughout the region reached over these T steps, s has an L_s-Lipschitz gradient on each update segment, and $\|\nabla F_\sigma(w_t)\| + e \le G$, where $e = \sqrt{TV/\alpha}$. If

$$\varepsilon + b_\sigma + e \le \frac{am}{2}, \qquad 0 < \eta \le \frac{am^2}{2L_sG^2}, \tag{24}$$

then, with probability at least $1-\alpha$ over the optimization randomness,

$$s(w_T) \ge s(w_0) + \frac{T\eta am^2}{4}. \tag{25}$$

Consequently, an instance initially outside the (n,δ) boundary crosses it whenever

$$T\eta am^2/4 \ge s_\star(n,\delta) - s(w_0). \tag{26}$$

For n fresh, independent rollouts from the resulting fixed policy, the joint probability of reaching such a policy and discovering a success is at least $(1-\alpha)(1-\delta)$.

Proof. For $V > 0$, conditional Markov's inequality applied to the squared estimation error bounds the probability of $\|\widehat{g}_t - \nabla F_\sigma(w_t)\| > e$ by α/T; for $V = 0$, the error is zero almost surely. A union bound yields simultaneous error bounds for all T steps with probability at least $1-\alpha$, without requiring independent optimization steps. On this event, set $v_t = \nabla s(w_t)$ and write $\widehat{g}_t = -av_t + r(w_t) + b_t + z_t$, where $\|b_t\| \le b_\sigma$ and $\|z_t\| \le e$. Thus

$$\begin{aligned} -\langle v_t, \widehat{g}_t\rangle &\ge a\|v_t\|^2 - (\varepsilon + b_\sigma + e)\|v_t\| \\ &\ge \tfrac{a}{2}\|v_t\|^2 \ge \tfrac{am^2}{2}. \end{aligned} \tag{27}$$

Smoothness and $\|\widehat{g}_t\| \le G$ imply

$$\begin{aligned} s(w_{t+1}) - s(w_t) &\ge -\eta\langle v_t, \widehat{g}_t\rangle - \tfrac{L_s\eta^2}{2}\|\widehat{g}_t\|^2 \\ &\ge \tfrac{\eta am^2}{2} - \tfrac{L_s\eta^2G^2}{2} \ge \tfrac{\eta am^2}{4}. \end{aligned} \tag{28}$$

Summing over t proves Eq. (25). Equation (16) then gives the crossing criterion. Conditional on the optimization event, the fresh-rollout discovery probability is at least $1-\delta$, proving the joint bound.□

Interpretation and cost. The gain comes from changing the policy's probability of successful interaction sequences, rather than increasing the number of draws from an unchanged policy. The sufficient condition balances useful surrogate alignment against misalignment, smoothing bias, and rollout noise. It is deliberately conservative: the uniform gradient lower bound is needed only along the finite path to the threshold and cannot hold indefinitely for a bounded success probability. All optimization rollouts must be counted when comparing total cost with fixed-policy sampling; the theorem does not prove a lower total rollout complexity. Independent estimator terms can reduce variance with K, but shared baselines and correlated tool responses do not justify an automatic $1/K$ claim.

A.2.4 Illustration, failure case, and applicability

An analytic two-decision example. Consider two independent binary decisions $A_1, A_2 \sim \mathrm{Bernoulli}(p(w))$, with $p(w) = (1 + e^{-w})^{-1}$, and success only when both equal one. Then $s(w) = p(w)^2$. An idealized graded surrogate $f = 2 - A_1 - A_2$ has $F(w) = 2 - 2p(w)$, so

$$\begin{aligned} s'(w) &= 2p(w)^2(1 - p(w)), \qquad F'(w) = -2p(w)(1 - p(w)), \\ -s'(w)F'(w) &= 4p(w)^3(1 - p(w))^2 > 0. \end{aligned} \tag{29}$$

For small p, full successes are rare, yet failed trajectories can still have distinct losses of one or two. Loss-guided updates increase w and hence success probability. In contrast, zero-baseline correctness-reward updates vanish on an all-failure batch. This is an analytic illustration of graded feedback, not an experiment or a claim that our answer loss observes intermediate correctness labels. It shows how discovery can improve before successes become frequent; it does not guarantee that every all-failure batch contains a useful signal.

Why the assumptions matter. Even an answer NLL need not align with success probability. Consider two equally likely trajectory contexts and a one-token reference answer. Let its conditional probabilities change from $(0.01, 0.99)$ to $(0.20, 0.60)$. Expected NLL falls from $-\frac{1}{2}\log(0.01 \cdot 0.99)$ to $-\frac{1}{2}\log(0.20 \cdot 0.60)$, yet sampled-answer success falls from 0.50 to 0.40. More strongly, along $p_1(w) = 0.01 + 0.19w$, $p_2(w) = 0.99 - 0.39w$ for $w \in [0, 1]$, the success probability is $s(w) = 0.50 - 0.10w$, while $F(w) = -\frac{1}{2}\log(p_1(w)p_2(w))$ decreases throughout the interval. NLL improves the geometric mean of answer probabilities here, whereas success depends on their arithmetic mean. This analytic counterexample shows why a falling answer loss or a smooth optimization path alone cannot establish capability expansion. If success is impossible for every reachable LoRA policy, no perturbation method can create a successful trajectory; a flat region with $\nabla s = 0$ is also outside Theorem 1.

Scope for the implemented pipeline. The theorem analyzes plain stochastic ZO descent, not Adam with data-dependent momentum and preconditioning. Extending it to the implementation requires alignment of the actual update direction and corresponding step/noise bounds. Stationary tools, unbiased rollout evaluations, smoothness, and surrogate alignment are analytical conditions, not verified properties of live web interaction, cache reuse, or the current answer loss. The results therefore explain a sufficient mechanism for instance-specific discovery; they do not guarantee monotonic empirical success, downstream SFT generalization, or that zeroth-order optimization is the only way to cross the boundary. The matched-budget experiments test discovery directly, while the parameter-path diagnostics below provide complementary evidence about optimization behavior.

A.3 Optimization Trajectory Analysis

For 50 difficult examples, we record the LoRA parameters and answer loss at every optimization round. For each example, we construct a fixed two-dimensional PCA basis from the parameter displacements of both our method and a matched-budget random-perturbation baseline, and project all checkpoints onto this shared basis. Table 8 reports a representative trajectory and aggregate diagnostics. On the representative example, our trajectory moves along a consistent direction while the answer loss decreases from 4.333 to 0.024. In contrast, the random baseline ends with a higher loss than its initial value. Across all 50 examples, our updates have higher consecutive-direction cosine similarity and trajectory efficiency, together with a larger loss decrease and more successful examples. These results are consistent with loss-guided optimization rather than unguided random exploration.

Table 8: Low-dimensional analysis of the parameter optimization trajectory. The PCA basis is shared by our method and the matched-budget random-perturbation baseline for each example.

(a) Shared-PCA projection on a representative difficult example.

	Ours			Random perturbation		
Round	PC1	PC2	Answer loss	PC1	PC2	Answer loss
0	0.000	0.000	4.333	0.000	0.000	4.333
4	-0.354	0.059	2.541	-0.173	0.314	3.486
8	-1.630	0.526	0.391	0.375	-0.929	6.925
12	-2.819	0.928	0.454	1.339	-1.733	7.376
15	-7.008	1.617	0.024	-0.564	-3.811	5.197

(b) Aggregate diagnostics over 50 difficult examples.

Diagnostic	Ours	Random perturbation
Mean answer-loss decrease from round 0 to 15	74.8%	13.4%
Checkpoints with loss below the initial value	62.3%	46.3%
Mean cosine similarity between consecutive updates	0.85	0.02
Mean trajectory efficiency $\frac{\|w_R - w_0\|_2}{\sum_t \|w_{t+1} - w_t\|_2}$	0.64	0.27
Successful examples after 15 rounds	23	5

A.4 Hyperparameter sensitivity

We randomly select 50 difficult examples and vary one hyperparameter at a time while keeping the remaining settings fixed. We evaluate the number of perturbation directions K, perturbation scale σ, LoRA rank r, and learning rate η. Table 9 shows that the number of successful examples changes only slightly over most tested settings. The default configuration is not the only effective choice, suggesting that the trajectory-discovery results are not narrowly tuned to a single hyperparameter setting.

Table 9: Hyperparameter sensitivity on 50 difficult examples. We vary one factor at a time; default settings are bolded.

Factor	Tested value	Successful instances
K	1	20 / 50
	2	**23 / 50**
	4	24 / 50
	6	24 / 50
σ	5×10^{-3}	22 / 50
	$\mathbf{1 \times 10^{-3}}$	**23 / 50**
	5×10^{-4}	21 / 50
r	8	20 / 50
	16	**23 / 50**
	64	23 / 50
η	5×10^{-6}	22 / 50
	$\mathbf{1 \times 10^{-6}}$	**23 / 50**
	5×10^{-7}	19 / 50

A.5 One-sided versus two-sided estimation

The two-sided estimator of zero-order gradients evaluates positive and negative perturbations for each direction, whereas the one-sided estimator reuses a shared unperturbed rollout. We compare the two estimators under the same four-rollout budget per round. As shown in Table 10, the two-sided estimator solves 24 examples and the one-sided estimator solves 23. Since their trajectory-discovery performance is comparable, we use the one-sided estimator in the main experiments because it supports more perturbation directions with a shared baseline rollout.

Table 10: Comparison of one-sided and two-sided estimators on 50 difficult examples under the same rollout budget.

Estimator	Directions	Rollouts / direction	Rollouts / round	Successful
One-sided (Ours)	3	1 + shared baseline	4	23 / 50
Two-sided SPSA	2	2	4	24 / 50

A.6 Memory and Wall-Clock Comparison

For the alternatives compared in Table 5, fixed-trajectory FO requires 21–24 GB of GPU memory and 2.5–3.5 hours of wall-clock time under the same Qwen-3-4B configuration, whereas our ZO

implementation requires 12–15 GB and 1.7–2.4 hours. These measurements show the practical benefit of avoiding backpropagation in this configuration, rather than implying that ZO is uniformly faster than first-order optimization.

A.7 Training Stability Across Random Seeds

We repeat the main controlled evaluation with five independent random seeds and report the mean, standard deviation, and 95% confidence interval. As shown in Table 11, the gains remain consistent across seeds. We also repeat the Pass@8 trajectory-discovery evaluation and obtain success rates of $63.5 \pm 0.4\%$ on the training set and $41.9 \pm 0.8\%$ on the validation set.

Table 11: Main results over five independent random seeds. Results are reported as mean $\pm$ standard deviation with 95% confidence intervals.

Method	GAIA (%)	WebWalkerQA (%)
Initial agent	19.1 ± 1.9 (16.8, 21.5)	19.7 ± 1.9 (17.3, 22.0)
Ours	$\mathbf{28.6 \pm 0.6}$ (27.7, 29.4)	$\mathbf{28.3 \pm 0.8}$ (27.3, 29.3)

A.8 Transfer of the Trained Policy

A.8.1 Non-retrieval and non-verbatim subsets

We identify 56 GAIA questions that do not require information retrieval and instead mainly involve multi-step reasoning, computation, or direct problem solving. The initial policy solves 5 examples, the RL baseline solves 11, and our method solves 16, corresponding to accuracies of 8.93%, 19.64%, and 28.57%, respectively. We further evaluate 68 examples for which the gold answer does not appear verbatim in any retrieved observation. The initial policy solves 7 examples, while our method solves 21. This result shows that the answer loss remains useful when the agent cannot obtain the answer through direct copying from retrieved text.

A.8.2 Performance on harder benchmarks

We evaluate the final evolved agent on BrowseComp-en and BrowseComp-zh without benchmark-specific zeroth-order trajectory discovery. As shown in Table 12, our method improves over both the initial policy and RL on the two benchmarks. Together with the non-retrieval and non-verbatim analyses, these results show that the improvement is not limited to direct retrieval on the original evaluation benchmarks.

Table 12: Results on non-retrieval and non-verbatim GAIA subsets and direct transfer to BrowseComp-en and BrowseComp-zh. The first two rows report accuracy, and the last two rows report Pass@8.

Evaluation	Initial	RL	Ours
GAIA non-retrieval (56)	8.93	19.64	**28.57**
GAIA non-verbatim (68)	10.29	–	**30.88**
BrowseComp-en	2.29	3.40	**4.11**
BrowseComp-zh	17.64	21.45	**23.53**

B Tools, Prompts, and Case Studies

This appendix documents the agent's tool interface and shows three paired evaluation rollouts. The retained *Think* passages and tool observations are excerpts from the supplied records; `[...]` marks omitted material. Tool calls show their function names and selected original arguments, with long URLs shortened for display.

B.1 Tool Interfaces

Our tools refer to the implementation released with WebDancer (Wu et al., 2025a). The cards show the fields, types, and required status advertised in the evaluation logs; descriptive prose has been shortened for display.

search

query *required*
Array of strings; at least one query.
Returns web search results and snippets.

visit

url *required*
Array of URL strings.
String specifying the fact to extract.

google_scholar

query *required*
Array of strings; at least one query.
Returns scholarly search results.

parse_file

files *required*
Array of file-path strings.
Returns extracted file content.

PythonInterpreter

JSON arguments *none required*
The logged wrapper advertises an empty argument object. Executable code is supplied separately in a `<code>` block.

In the logged wrapper, `visit.url` is an array, whereas the current upstream implementation also accepts a single URL. The upstream Python interface declares a `code` argument; the logged wrapper uses the separate block above. A final answer ends a rollout and is not an external tool.

B.2 Prompt and Interaction Protocol

The following presents the agent prompt used in our experiments. The tool definitions are reformatted for readability while retaining their advertised arguments and requirements.

Agent prompt

You are a deep research assistant. Your core function is to conduct thorough, multi-source investigations into any topic. You must handle both broad, open-domain inquiries and queries within specialized academic fields. For every request, synthesize information from credible, diverse sources to deliver a comprehensive, accurate, and objective response. When you have gathered sufficient information and are ready to provide the definitive response, you must enclose the entire final answer within `<answer></answer>` tags.

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

Example of a correct call:

```
<tool_call>
{"name": "PythonInterpreter", "arguments": {}}
<code>
import numpy as np
## Your code here
print(f"The result is: {np.mean([1,2,3])}")
</code>
</tool_call>
```

google_scholar. Leverage Google Scholar to retrieve relevant information from academic publications. Accepts multiple queries. This tool will also return results from Google Search. The required argument `query` is an array of strings containing at least one search query.

parse_file. Parse multiple user-uploaded local files, such as PDF, DOCX, PPTX, TXT, CSV, XLSX, DOC, ZIP, MP4, and MP3 files. The required argument `files` is an array of uploaded file names.

For each function call, return a JSON object with the function name and arguments within `<tool_call></tool_call>` tags.

`Current date:`

B.3 Case Study: Observed Behavioral Changes

The three paired rollouts exhibit complementary changes:

- **Query refinement and multi-hop linking.** After an uninformative APOD calendar visit, the optimized agent searches for the specific image, identifies Marquette, and follows the namesake to the building and firm.
- **Unit-aware calculation.** The optimized agent keeps individuals and breeding pairs distinct, converts the latter before subtraction, and returns the requested scale.
- **Evidence-based answer revision.** Rather than excluding countries by an assumed diplomatic date, the optimized agent visits the CIIE page and retains all three countries named there.

The excerpts below preserve the wording of the shown *Think* spans and tool evidence, including uncertainty and errors. Other messages and portions of long responses are omitted. These are selected evaluation rollouts, not zero-order update traces or an isolated causal test of training; the GAIA post-training instructions changed and tool responses were not held fixed. The three WebWalkerQA post-training records for the last question are identical, not independent replications.

B.4 Case 1: Following an Entity Chain

Question

During the first week of August 2015, one of the NASA Astronomy Pictures of the Day shows the lights of a city on the horizon. The namesake of this city also has a landmark building in Chicago named after him. What is the name of the architectural firm that designed this landmark building? Give the first name appearing in the name of the firm as of June 2023.

Base model

Tool Call: `search`: "NASA Astronomy Picture of the Day August 2015 city lights"; "namesake city Chicago landmark".
Tool Response: "Marina City - Wikipedia"; "Portions of the complex were designated a Chicago Landmark in 2016." `[...]`
Think: Marina City is a landmark in Chicago designed by Bertrand Goldberg. It's a twin-tower complex with a distinctive "corncob" shape. `[...]`
Think: Wait, the firm that designed Marina City is "Bertrand Goldberg Architects", so the first name in the firm's name is "Bertrand". `[...]`
Answer: `Bertrand`

Optimized model

Tool Call: `search`: "NASA Astronomy Pictures of the Day August 2015"; `visit`: `ca1508.html`, goal: "Find the NASA APOD from August 2015 showing city lights".
Think: Okay, so the calendar page lists all August dates but doesn't describe what's in each image. So I need to check one of those specific dates to find the city lights APOD. `[...]`
Tool Call: `search`: "City Lights in NASA APOD August 2015".
Tool Response: "In the featured image, the yellow lights on the horizon are the city lights of Marquette, Michigan, USA."
Think: Okay, result 1 mentions Marquette, Michigan as having city lights in an August 3 APOD. Let's check that page to confirm.
Tool Call: `visit`: `ap150803.html`, goal: "Get the description of the APOD from August 3, 2015, to identify the city".
Think: The namesake of Marquette would be French missionary Jacques Marquette. His name is used to name the city. `[...]`
Tool Call: `search`: "Chicago building named after Jacques Marquette"; `visit`: Marquette Building page, goal: "Find the architect of the Marquette Building in Chicago".
Tool Response: "Architect[Holabird & Roche]" `[...]`
Think: Alright, the architectural firm that designed the Marquette Building in Chicago is Holabird & Roche. The question asks for the first name appearing in the firm's name as of June 2023.
Answer: `Holabird`

B.5 Case 2: Keeping Population Units Straight

Question

What is the absolute difference in tens of thousands between the population of chinstrap penguins on the Wikipedia page for penguin species populations as of the end of 2018 and the population recorded in the Nature.com "global population assessment of the Chinstrap penguin" article from 2020, assuming two penguins per breeding pair?

Base model

Tool Call: `search`: "Wikipedia Chinstrap penguin population 2018"; `[... repeated searches omitted]`
Tool Response: "In 2018, the IUCN estimated the chinstrap penguin population at around 8 million individuals." `[...]`
Think: Given the information from the tool response (Wikipedia search), the population is 8 million individuals. Let's assume the Nature.com article from 2020 reports a different number. `[...]`
Think: Assume the Nature.com article (2020) states 7 million individuals. `[...]` Difference: 1,000,000 penguins. `[...]` 1,000,000 / 10,000 = 100.
Answer: `100`

Optimized model

Tool Call: `search`: "Wikipedia Chinstrap penguin population 2018".
Tool Response: "Chinstrap penguin, Pygoscelis antarcticus, 8,000,000" `[...]`
Think: Result 1 says Chinstrap penguin population is 8,000,000 (8 million). `[...]`
Tool Call: `search`: "Nature global population assessment chinstrap penguin 2020".
Tool Response: "one recent comprehensive assessment (2020) estimated ~3.42 million breeding pairs" `[...]`
Think: Nature's study (2020) states 3.42 million breeding pairs. Multiply by 2 gives 6.84 million individuals. Difference: 8,000,000 - 6,840,000 = 1,160,000. Absolute value. In tens of thousands: 116.
Answer: `116`

B.6 Case 3: Checking the Complete Country List

Question

Which African countries participated in the 7th China International Import Expo with the theme of the 60th anniversary of diplomatic relations with China?

Base model

Tool Call: `search`: "7th China International Import Expo African countries participating"; `[...]`
Tool Response: "In particular, Tanzania, Tunisia and Zambia will be key components ..." `[...]`
Think: Result 1 mentions: "Tanzania, Tunisia and Zambia will be key components..." (Oct 24, 2024) `[...]`
Think: Zambia is the only country with a diplomatic relation starting in 1964. Thus, the answer is **Zambia**. `[...]`
Answer: `Zambia`

Optimized model

Tool Call: `search`: "China International Import Expo 7th theme diplomatic relations 60th anniversary".
Think: However, I should verify if there are others. Maybe another country like Ethiopia or Kenya? But the search results don't mention them here. The search results provided by Google are limited, so perhaps Zambia is the primary one identified.
Tool Call: `visit`: `ciie.org/.../20241024/46201.html`, goal: "List of African countries participating in the 7th CIIE with the 60th anniversary of diplomatic relations theme".
Tool Response: "In particular, Tanzania, Tunisia and Zambia will be key components of events themed on the Year of Bilateral Diplomatic Relations at the CIIE, as this year marks the 60th anniversary of China establishing diplomatic relations with them."
Think: The search results initially hinted at Zambia, but after visiting the official CIIE page, it's confirmed that Tanzania, Tunisia, and Zambia are the three African countries that participated in the 7th China International Import Expo with the theme tied to their 60th anniversary of diplomatic relations with China.
Answer: `Tanzania, Tunisia, Zambia`