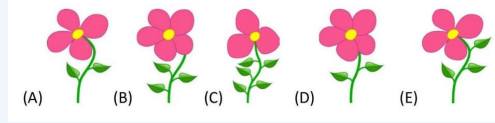


A. Full Responses for Multimodal Example

SYSTEM: You are a helpful AI Assistant that provides well-reasoned and detailed responses. You first think about the reasoning process as an internal monologue and then provide the user with the answer. Respond in the following format: <think> </think>\n<answer> </answer>

USER: The ladybird would like to sit on his flower. The flower has five petals and the stem has three leaves. On which flower should the ladybird sit?



ASSISTANT: <think> Let me inspect each flower carefully. Flower A appears to have exactly five petals, but its stem only sprouts two leaves. Flower B has a nice regular star of five petals—and I thought I spotted three little leaves tucked along its stem too. Flower C looks squashed, maybe only four broad petals and actually closer to four leaves as well. D has a perfect pentagon of petals but now that I look I remember only two leaves there. E seemed promising at first glance but I can’t quite recall its leaf count. On balance, my initial gut is that D stands out with a clear five-petal bloom and I vaguely thought three leaves peeked out—but I must have misremembered. Still, I’ll go with D for now. </think>
<answer>D</answer>

<think> **Re-checking D**, I see exactly two leaves—one on each side of the stem—so D definitely fails the “three leaves” criterion. The next candidate is E: I recall E had a single top-right leaf, another mid-left leaf, and one lower-left leaf, giving three in total, and its petals looked evenly spaced to make five. So E fits “five petals, three leaves,” so my second guess will be E. </think>
<answer>E</answer>

<think> **Oops**—I’ve miscounted again. On closer inspection, E actually only has four fully visible petals because one is hidden behind the stem’s curve. Meanwhile, B clearly displays five distinct petals and exactly three leaves along its stem (one low-left, one mid-right, one upper-left). Therefore the only flower meeting both criteria is B. </think>
<answer>B</answer>

Figure 7. Qualitative multimodal example of recursive think–answer process. The model refines its answer over multiple reasoning cycles by correcting initial counting errors and verifying object-level visual details, ultimately identifying the only flower that meets both criteria.

B. Algorithm of Recursive Think-Answer Process

Algorithm 1 Recursive Think-Answer Process for LLMs and VLMs

Require: Pre-trained Confidence Generator \mathcal{C}_ϕ and Pre-trained LLMs/VLMs $\pi_{\theta_{\text{init}}}$

- 1: Set reference model $\pi_{\text{ref}} \leftarrow \pi_{\theta_{\text{init}}}$
 - 2: Set the training model $\pi_\theta \leftarrow \pi_{\theta_{\text{init}}}$
 - 3: **for** sample a batch \mathcal{B} in Dataset **do**
 - 4: Copy and freeze model $\pi_{\theta_{\text{old}}} \leftarrow \pi_\theta$
 - 5: Sample G outputs $\{o_i\}_{i=1}^G \sim \pi_{\theta_{\text{old}}}(\cdot|q)$, until M maximum recursive.
 - 6: Extract the responses G' in G outputs until the early correct responses.
 - 7: Replace the responses: $G \leftarrow G'$
 - 8: Compute rewards and advantages for G outputs by confidence generator & answer parsing
 - 9: **for** Confidence generator Updating iteration = 1, 2, \dots , μ **do**
 - 10: Update \mathcal{C}_ϕ by using Eq. (6)
 - 11: **end for**
 - 12: **for** LLMs/VLMs Updating iteration = 1, 2, \dots , μ **do**
 - 13: Update π_θ by using Equation Eq. (3)
 - 14: **end for**
 - 15: **end for**
-

C. Detailed Comparison between Performance and Computation Complexity

Comparison with Related Refinement and Verification Methods. We compare R-TAP against four closely related baselines: (a) Reflexion [87], (b) Self-Consistency [96], (c) Self-Refine [75], and (d) Self-Verification [69]. All methods are evaluated under the same output token budget. Unlike iterative refinement or verification-based approaches, which rely on explicit self-feedback loops, majority voting, or external verification during inference, R-TAP learns when to continue or terminate reasoning via reinforcement learning. As a result, it removes the need for repeated trial-and-error generation at deployment. Although the confidence generator in R-TAP is trained with binary supervision, it does not function as a hard 0/1 classifier at inference. Instead, it produces a continuous signal in $[0, 1]$, which enables the model to measure confidence improvement across recursive steps and regulate reasoning depth smoothly.

	MaxTokenLimit	AIME25	HMMT Feb25	OmniMath	GPQA	LCB	Avg	Oops-style count	Decoding token count	Training Time
Phi-4-reasoning-plus	32768	78.0	53.6	81.9	69.3	65.9	69.7	15.7	14509.7	-
w/ Reflexion [87]	32768	80.2	56.1	83.4	71.3	66.5	71.5	18.9	15230.4	51 hours
w/ Self-Consistency [96]	32768	81.1	55.8	84.0	72.2	66.4	71.9	16.8	16012.9	58 hours
w/ Self-Refine [75]	32768	78.5	54.9	83.1	71.0	68.5	71.2	14.3	13890.6	45 hours
w/ Self-Verification [69]	32768	82.4	57.6	85.7	73.5	64.8	72.8	17.5	14890.3	51 hours
R-TAP	32768	83.7	60.3	86.2	76.7	72.1	75.8	5.6	4378.8	28 hours

	MaxTokenLimit	MMMU	MathVista	OlympiadBench	MathVision	MMMU-Pro	Avg	Oops-style count	Output token count	Training Time
R1V2-38B	12000	73.6	74.0	62.6	49.0	52.0	62.2	17.2	9890.7	-
w/ Reflexion [87]	12000	74.5	74.8	63.0	49.5	53.7	63.1	17.5	10989.3	57 hours
w/ Self-Consistency [96]	12000	75.0	76.0	64.0	50.0	52.5	63.5	18.6	10678.5	60 hours
w/ Self-Refine [75]	12000	73.8	74.2	62.0	49.0	55.5	62.9	15.3	9938.2	59 hours
w/ Self-Verification [69]	12000	76.5	77.0	65.0	51.0	53.0	64.5	17.8	10183.8	58 hours
R-TAP	12000	78.2	82.3	69.4	56.8	59.2	69.2	8.5	5789.4	39 hours

Token Efficiency and Computational Cost. R-TAP does not increase total output tokens. On the contrary, it substantially reduces them. While recursive sampling is introduced during training, the learned policy discourages unnecessary self-corrections at inference time. As shown in this table, R-TAP reduces output tokens by approximately 2–3 \times compared to self-consistency, self-refine, and verification-based baselines under the same output token budget, while achieving higher accuracy.

T	G	AIME25	HMMT Feb25	OmniMath	GPQA	LiveCodeBench	Avg	Output token count	Training Time
1	4	76.1	52.1	80.4	67.9	64.4	68.2	15034.9	51 hours
1	8	77.2	52.9	81.2	68.6	65.2	69.0	15210.7	52 hours
1	12	78.0	53.6	81.9	69.3	65.9	69.7	14872.5	58 hours
2	12	80.9	57.1	84.1	73.3	68.6	72.6	9258.9	42 hours
3	12	82.3	58.9	85.0	75.1	70.3	74.9	6127.4	34 hours
4	12	83.7	60.3	86.2	76.7	72.1	75.8	4378.8	28 hours
4	8	83.0	59.7	85.7	76.0	71.4	75.2	4442.3	25 hours
4	4	82.2	58.9	85.0	75.2	70.6	74.4	4321.6	22 hours

T	G	MMMU	MathVista	OlympiadBench	MathVision	MMMU-Pro	Avg	Decoding token count	Training Time
1	4	72.4	72.6	61.2	47.8	50.6	60.9	10342.7	55 hours
1	8	73.1	73.4	62.0	48.5	51.4	61.7	9927.3	62 hours
1	12	73.6	74.0	62.6	49.0	52.0	62.2	10168.9	68 hours
2	12	76.1	77.0	65.3	52.7	55.0	64.1	8234.6	55 hours
3	12	77.5	79.4	67.2	54.9	57.8	67.4	6912.1	46 hours
4	12	78.2	82.3	69.4	56.8	59.2	69.2	5789.4	39 hours
4	8	77.8	81.7	68.8	56.4	58.7	68.7	5698.2	34 hours
4	4	77.1	80.8	67.9	55.6	57.8	67.8	5861.5	30 hours

Effect of Majority Voting. We further evaluate self-consistency with varying voting numbers (N), under identical output token budgets (32768 for Phi-4-reasoning-plus and 12000 for R1V2-38B). As expected, increasing N consistently improves performance across all models. However, R-TAP-trained models already achieve strong performance with $N = 1$, and additional voting yields only marginal gains. This indicates that R-TAP does not replace self-consistency; rather, it learns a more stable single-sample reasoning policy, thereby reducing reliance on majority voting during inference.

Voting N	AIME25	HMMT Feb25	OmniMath	GPQA	LiveCodeBench	Avg	Voting N	MMMU	MathVista	OlympiadBench	MathVision	MMMU-Pro	Avg
1	83.7	60.3	86.2	76.7	72.1	75.8	1	78.2	82.3	69.4	56.8	59.2	69.2
3	83.9	60.5	86.4	76.9	72.3	76.0	3	78.3	82.4	69.5	56.9	59.3	69.3
5	84.0	60.6	86.5	77.0	72.4	76.1	5	78.3	82.4	69.5	56.9	59.3	69.3
7	84.1	60.7	86.6	77.1	72.5	76.2	7	78.4	82.5	69.6	57	59.4	69.4
9	84.1	60.7	86.6	77.1	72.5	76.2	9	78.5	82.6	69.7	57.1	59.5	69.5
12	84.2	60.8	86.7	77.2	72.6	76.3	12	78.5	82.6	69.7	57.1	59.5	69.5
15	84.2	60.8	86.7	77.2	72.6	76.3	15	78.5	82.6	69.7	57.1	59.5	69.5
18	84.2	60.8	86.7	77.2	72.6	76.3	18	78.5	82.6	69.7	57.1	59.5	69.5

Recap. R-TAP consists of four key components: (a) GRPO-based reinforcement learning for Think-Answer trajectories, (b) a recursive reward mechanism derived from both the confidence generator and intermediate results, (c) suppression of unnecessary Oops-style refinement, and (d) improved inference efficiency resulting from reduced refinement steps.

Importantly, while prior uncertainty- or refinement-based approaches use confidence signals for reranking, filtering, or verification after generation, R-TAP integrates confidence as an internal reinforcement signal during training. This signal continuously modulates recursive reasoning depth, directly shaping the learned reasoning policy instead of performing post-hoc correction at inference time.

Future Works. We plan to further advance R-TAP from the perspective of efficiency-oriented model design [54–61] and multiple evaluation benchmark [62, 63]. In particular, we aim to develop adaptive recursion strategies that dynamically determine the necessity and depth of additional Think–Answer cycles, thereby minimizing redundant computation while preserving reasoning accuracy. Instead of relying on fixed recursion depth or static confidence thresholds, lightweight gating mechanisms [52] or early-exit policies could be learned to selectively allocate reasoning steps based on estimated uncertainty. We also intend to explore parameter-efficient training schemes—such as partial fine-tuning or modular confidence heads—to reduce memory and training overhead, making R-TAP more practical for resource-constrained environments. Finally, extending confidence-guided recursive reasoning to smaller-scale models while maintaining competitive performance will be a key direction, enabling efficient yet reliable and robust [46–48, 51, 53] systems suitable for real-world deployment.