

000 CYCLICREFLEX: IMPROVING REASONING MODELS VIA 001 CYCLICAL REFLECTION TOKEN SCHEDULING 002 003 004

005 **Anonymous authors**

006 Paper under double-blind review

007 008 ABSTRACT 009

011 Large reasoning models (LRMs), such as OpenAI’s o1 and DeepSeek-R1, harness
012 test-time scaling to perform multi-step reasoning for complex problem-solving.
013 This reasoning process, executed before producing final answers, is often guided
014 by special juncture tokens that prompt self-evaluative reflection. We refer to
015 these transition markers and reflective cues as “*reflection tokens*” (e.g., “wait”,
016 “but”, “alternatively”). In this work, we treat reflection tokens as a “*resource*” and
017 introduce the problem of *resource allocation*, aimed at improving the test-time
018 compute performance of LRMs by adaptively regulating the frequency and place-
019 ment of reflection tokens. Through empirical analysis, we show that both excessive
020 and insufficient use of reflection tokens, referred to as over-reflection and under-
021 reflection, can degrade model performance. To better understand this trade-off,
022 we draw an analogy between reflection token usage and learning rate scheduling
023 in optimization. Building on this insight, We propose *cyclical reflection token*
024 *scheduling* (termed CyclicReflex), a training-free decoding strategy that dyna-
025 mically modulates reflection token logits with a bidirectional, position-dependent
026 triangular waveform, incurring no additional computation cost. Experiments on
027 MATH500, AIME2024/2025, AMC2023, GPQA Diamond and LiveCodeBench
028 demonstrate that CyclicReflex consistently improves performance across model
029 sizes (1.5B–8B), outperforming standard decoding and recent approaches such as
030 TIP (thought switching penalty) and S1.

031 1 INTRODUCTION 032

033 There has been a recent surge in the development of large reasoning models (LRMs), driven by the
034 introduction of chain-of-thought (CoT) (Wei et al., 2022). Notable examples include OpenAI’s o1
035 (OpenAI, 2024), Qwen 2.5 (Yang et al., 2024), DeepSeek-R1 (Guo et al., 2025), and Kimi-1.5 (Team
036 et al., 2025). These models perform multi-step reasoning by generating so-called reflection tokens,
037 phrases such as “*wait*”, “*but*”, “*alternatively*”, which signal hesitation, reconsideration, alternative
038 exploration, or intermediate analysis. In parallel, test-time scaling techniques (Snell et al., 2024;
039 Liu et al., 2025) have emerged as a complementary approach for improving reasoning accuracy by
040 expanding the breadth or depth of CoT traces during inference.

041 However, LRMs remain prone to reasoning failures due to mismanagement of reflection tokens, often
042 resulting in either *underthinking* or *overthinking*. Underthinking occurs when the model fails to fully
043 explore promising reasoning paths for complex problems, often terminating prematurely or switching
044 strategies too soon (Wang et al., 2025a; Su et al., 2025). In contrast, overthinking arises when the
045 model generates an excessive number of reflection tokens on simple problems, leading to unnecessary
046 computational overhead (Chen et al., 2024; Kumar et al., 2025a). These observations show that, as
047 internal signals for deliberative reasoning, reflection tokens play a critical role in shaping answer
048 quality. The emerging challenges further underscore the need for a principled mechanism to regulate
049 reflection token usage during inference.

050 In this work, we introduce the concept of **resource allocation for LRMs**, treating reflection tokens
051 as a valuable resource whose scheduling along the CoT trajectory (*a.k.a.* reasoning trace) can be
052 strategically designed to improve reasoning accuracy. The objective is to optimize the quantity and
053 placement of reflection tokens, adapting dynamically to the reasoning schedule and the difficulty of
the current problem. For instance, some problems exhibit *under-reflection*, where too few reflection

tokens result in premature answer generation, while others suffer from *over-reflection*, where excessive tokens stall progress by repeatedly looping on phrases like “wait”. This raises a central question:

(Q) *How can we achieve effective resource allocation in LRM to mitigate both under-reflection and over-reflection?*

To answer this, we draw a conceptual analogy between reflection tokens in LRM and learning rates in optimization. Leveraging the *landscape of thoughts* (Zhou et al., 2025), we show that under-reflection mirrors the effect of an overly small learning rate, leading to premature convergence to suboptimal solutions, while over-reflection resembles a large learning rate that causes divergence. We briefly introduce our motivation and the underlying intuition below.

Overview of motivation and rationale: From stepsize hedging to cyclical learning rates. The critical role of learning rates (also known as stepsizes) in shaping optimization dynamics has been extensively studied (Nesterov, 1983; Allen-Zhu & Orecchia, 2014; Bubeck et al., 2015). A recent theoretical advancement, the *silver stepsize schedule* (Altschuler & Parrilo, 2024; 2025), demonstrates that replacing a constant learning rate with an approximately periodic, *hedging-style* schedule can *provably accelerate convergence* in gradient descent. This approach is known as *stepsize hedging* as it alternates strategically between large and small stepsizes, balancing rapid (but potentially unstable) exploration with slower, more stable convergence. A similar stepsize hedging idea has been applied to deep model training through *cyclical learning rate schedule* (Smith, 2017), which alternates between large and small learning rates in a triangular waveform. This strategy not only accelerates convergence but also enhances generalization, often eliminating the need for extensive hyperparameter tuning.

Motivated by the principle of stepsize hedging (Altschuler & Parrilo, 2024; 2025) and the demonstrated effectiveness of cyclical learning rates in deep learning (Smith, 2017), we propose **CyclicReflex**, a training-free decoding strategy that dynamically modulates the logits of reflection tokens using a position-dependent, periodic triangular waveform (see Fig. 1). Just as cyclical learning rates alternate between aggressive and conservative updates to balance exploration and convergence, CyclicReflex cyclically adjusts the sampling likelihood of reflection tokens to regulate the depth and stability of the reasoning process. Unlike conventional decoding methods, CyclicReflex is *bidirectional*, capable of both promoting and suppressing reflection token usage depending on the stage of generation. This flexibility enables CyclicReflex to address both under-reflection (insufficient reasoning) and over-reflection (excessive, redundant reasoning), offering a principled mechanism for reasoning modulation inspired by optimization dynamics.

Contributions. We summarize our main contributions below.

- We introduce and formalize the novel problem of resource allocation in LRM by treating reflection tokens as a computational resource, motivated by the dual challenges of under-reflection and over-reflection in reasoning generation.
- We draw a conceptual analogy between reflection token scheduling and learning rate scheduling in optimization, and validate it through the landscape of thoughts. Guided by that, we propose CyclicReflex, a test-time decoding strategy that cyclically modulates reflection token logits to dynamically balance reflection during generation.
- We conduct comprehensive experiments across six reasoning benchmarks and multiple model scales (1.5B–8B), demonstrating that CyclicReflex consistently improves both final-answer accuracy and self-correction capability, outperforming recent test-time decoding strategies such as TIP (Wang et al., 2025a) and S1 (Muenninghoff et al., 2025). Moreover, CyclicReflex integrates seamlessly with other test-time scaling techniques, yielding additional performance gains.

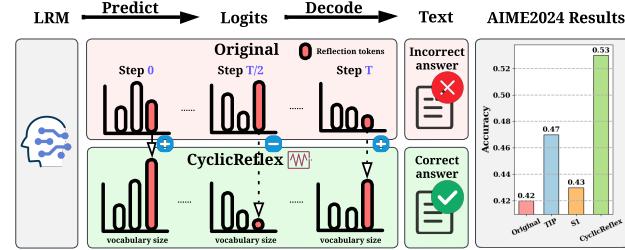


Figure 1: Schematic overview of our proposed method (CyclicReflex). The rightmost subfigure presents a comparison of final answer accuracy between CyclicReflex, the original LRM (DeepSeek-R1-Distill-Llama-8B), and decoding variants using TIP (Wang et al., 2025a) and S1 (Muenninghoff et al., 2025).

108
109

2 RELATED WORK

110
111
112
113
114
115
116
117
118
119

LRMs and CoT. CoT (Wei et al., 2022) enables LRM to solve complex tasks through intermediate reasoning steps before reaching a final answer. This technique underpins many recent LRM, including OpenAI’s o1 (OpenAI, 2024), Qwen 2.5 (Yang et al., 2024), DeepSeek-R1 (Guo et al., 2025), and Kimi-1.5 (Team et al., 2025), which often employ reinforcement learning to further improve their reasoning performance. Guo et al. (2025) show that even smaller models benefit substantially from fine-tuning with CoT-style data. A hallmark of CoT reasoning in these models is the emergence of reflection tokens (words like “wait” or “but”) that signal deliberation or self-correction, marking a shift from fast to slow thinking (Kumar et al., 2025b; Li et al., 2025). In this paper, we show that the reasoning performance of LRM can be enhanced by applying cyclical logits manipulation to reflection tokens.

120
121
122
123
124
125
126
127
128
129
130
131
132

Efficient reasoning. Despite their impressive capabilities, LRM often exhibit reasoning inefficiencies. Overthinking arises when the model generates unnecessarily long reasoning traces, leading to inflated outputs and increased computational cost (Chen et al., 2024; Kumar et al., 2025a). In contrast, underthinking occurs when the model halts reasoning too early, failing to adequately explore promising solution paths (Wang et al., 2025a; Su et al., 2025). Therefore, ensuring both the efficacy (*i.e.*, answer accuracy) and efficiency (*i.e.*, generation length) of reasoning is crucial. Building on this line of research, some approaches modify model behavior through post-training interventions. For instance, Luo et al. (2025); Aggarwal & Welleck (2025); Hou et al. (2025) use fine-tuning or reinforcement learning to explicitly control reasoning length. There also exist works that adopt training-free strategies. Wang et al. (2025b) propose guiding smaller models with larger ones at inference; Wang et al. (2025a) penalize reflection token logits to reduce over-reflection; And Yang et al. (2025); Muennighoff et al. (2025) develop early-exit mechanisms for efficient decoding. Our method also falls into the training-free category but differs in its dynamic to adaptively address both under- and over-reflection without model modification.

133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153

Test-time scaling. A growing body of work enhances LRM reasoning via test-time scaling. Basic strategies include manually inserting reflection tokens (*e.g.*, “wait”, “but”) to prompt deeper thinking (Muennighoff et al., 2025; Jin et al., 2025). More advanced methods such as Best-of-N generation and self-consistency sampling (Wang et al., 2022; Irvine et al., 2023; Brown et al., 2024) aim to select the most promising answer among multiple candidates, often guided by reward models. Structured decoding approaches, such as beam search (Feng et al., 2023), tree-of-thought (ToT) (Yao et al., 2023), and Monte Carlo tree search (MCTS) (Zhou et al., 2023), further improve answer quality by enabling the model to reason over multiple candidate paths. In pursuit of controlled reasoning, Wu et al. (2025) propose thinking intervention, which selectively inserts or edits specific thinking tokens during generation to tailor LRM behavior for downstream tasks. Recent analyses (Snell et al., 2024; Liu et al., 2025; Chen et al., 2025b; Zhang et al., 2025) also highlight that the effectiveness of test-time scaling varies with problem difficulty, motivating strategies that adapt to instance complexity. Sadhukhan et al. (2025) revisit test-time scaling laws, showing that attention KV-memory bottlenecks alter the trade-off between compute and model size and proposing a scaling law that favors larger sparse-attention models under fixed budgets. Kang et al. (2025) introduce self-certainty, a reward-free confidence metric that enables scalable Best-of-N selection while substantially reducing test-time compute. Chen et al. (2025c) propose iterative deepening sampling, which gradually increases sampling depth with budget-aware control to achieve more compute-efficient test-time scaling without modifying the underlying model. Our method provides the adaptive test-time compute by dynamically adjusting the influence of reflection tokens throughout the reasoning trajectory. We show that it consistently improves accuracy across difficulty levels and can be integrated seamlessly with other test-time strategies like Best-of-N and beam search.

154
155

3 RESOURCE ALLOCATION IN LARGE REASONING MODELS

156
157
158
159
160
161

In this section, we begin by introducing preliminaries on LRM, including their chain-of-thought trajectories (*i.e.*, reasoning traces) and the use of reflection tokens. We then motivate the problem of resource allocation over reflection tokens through an existing technique: thought switching penalty (TIP) test-time compute strategy. This warm-up study illustrates the critical influence and sensitivity of reflection tokens on the final answers produced by LRM.

162 **Preliminaries on LRM s, reasoning traces, and reflection tokens.** Unlike conventional LLMs,
 163 LRM s can incorporate an explicit *thinking stage* before arriving at a final answer in response to an
 164 input question (Li et al., 2025; Ding et al., 2025; Chen et al., 2025a). This thinking stage is typically
 165 realized through a CoT trajectory composed of multiple reasoning steps.

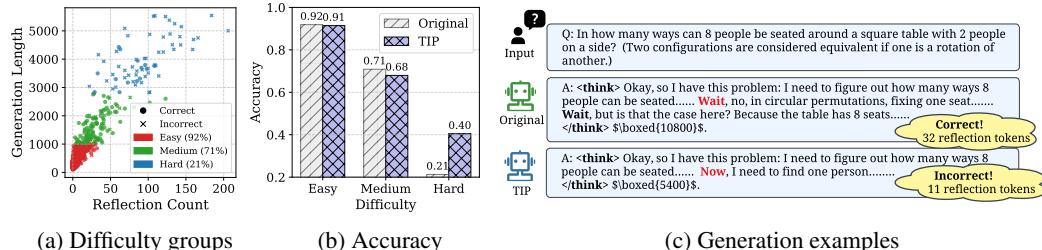
166 To be concrete, let x denote the input question, r the reasoning trace, and y the final answer. The
 167 reasoning trace can be written as $r = [r_1, r_2, \dots, r_T]$, a sequence of T intermediate reasoning steps
 168 (or thoughts) that serve as a precondition for producing the final answer y . These thoughts are often
 169 connected by lexical cues such as “*wait*”, “*but*”, and “*alternatively*”. In this work, we collectively refer to
 170 such words as **reflection tokens**, as they signal the
 171 LRM’s introspective or deliberative thinking during
 172 generation. These tokens typically signal hesitation,
 173 reconsideration or alternative exploration within the
 174 reasoning trace, serving as implicit markers of the
 175 model’s engagement in step-by-step reasoning. For
 176 ease of understanding, **Table 1** illustrates an example
 177 where an LRM tackles a complex math problem
 178 (x), along with its generated reasoning trace (r), final
 179 answer (y), and the highlighted reflection tokens
 180 embedded within the reasoning process.
 181

182 **Motivation: Influence of reflection tokens on final answer accuracy.** Next, we present a *warm-up*
 183 study to illustrate the critical role of reflection tokens in determining final answer quality, particularly
 184 across varying levels of problem difficulty. Prior work has identified several common issues in
 185 LRM reasoning, including “underthinking” (Wang et al., 2025a; Su et al., 2025)–where the model
 186 prematurely abandons promising lines of thought, resulting in insufficient reasoning depth–and
 187 “overthinking” (Muennighoff et al., 2025; Chen et al., 2024; Kumar et al., 2025a)–where excessive,
 188 unnecessary reasoning steps obscure or derail the correct final answer. These studies have observed
 189 that intervening in the reasoning trace with reflection tokens can help guide the model’s thought
 190 process (Wu et al., 2025). A common approach to controlling and integrating the effect of reflection
 191 tokens in LRM generation is to modify the decoding strategy to account for their occurrence. One
 192 such method is TIP (Wang et al., 2025a), which was proposed to discourage the generation of
 193 reflection tokens and thereby penalize frequent thought switches during the reasoning trace.
 194

195 Given the set of reflection tokens \hat{V} , TIP introduces a logit penalty (α) to the predicted score $z_{t,v}$
 196 when generating a reflection token $v \in \hat{V}$ at reasoning step t , yielding the updated logit
 197

$$\text{TIP}(\alpha) : \hat{z}_{t,v} = \begin{cases} z_{t,v} + \alpha, & \text{if } v \in \hat{V} \text{ and } t < T_0 \\ z_{t,v}, & \text{otherwise} \end{cases} \quad (1)$$

198 where α controls the strength of the logit intervention, and T_0 specifies the time window over which
 199 the adjustment is applied. For ease of presentation, $\text{TIP}(\alpha)$ denotes the TIP-based decoding strategy
 200 parameterized by α . It is worth noting that *TIP sets $\alpha \leq 0$ to penalize frequent thought switches*.
 201



202 Figure 2: (a) Answers from DeepSeek-R1-Distill-Qwen-7B on MATH500 clustered into Easy, Medium, and
 203 Hard using K-means over reflection word count and generation length. Each point represents one answer. (b)
 204 Accuracy of original decoding and TIP across difficulty levels. (c) Generation examples of original decoding
 205 and TIP for a problem from the Medium category.

206 Despite the heuristic nature of choosing α and T_0 , the TIP-based decoding strategy (Wang et al.,
 207 2025a) provides useful motivation for studying resource allocation over reflection tokens and its
 208 impact on reasoning effectiveness (e.g., on the MATH500 dataset) across *different problem difficulty*

216 *levels*. Specifically, we categorize MATH500 problems into three difficulty levels—Easy, Medium,
 217 and Hard—based on empirical final-answer accuracies of 92%, 71%, and 21%, respectively, since the
 218 hand-labeled bins in MATH500 are often inaccurate (Snell et al., 2024; Liu et al., 2025). As shown in
 219 **Fig. 2(a)**, these accuracy-based difficulty groups align closely with clusters derived from generation
 220 length and the number of reflection tokens produced. This suggests that for more challenging
 221 problems, LRM s tend to produce longer reasoning trajectories and more reflection tokens, indicating
 222 deeper engagement in problem-solving. **Fig. 2(b)** next compares the accuracy of the original decoding
 223 strategy and TIP across the difficulty groups in Fig. 2(a). As shown in Fig. 2(a), TIP improves accuracy
 224 on Hard problems *but reduces accuracy on Easy and Medium problems*. This suggests that TIP’s
 225 constant logit manipulation strategy (agnostic to the reasoning step t) does not yield *optimal* reasoning
 226 control. Furthermore, **Fig. 2(c)** shows a Medium-level example comparing original decoding with
 227 TIP. The first divergence in reasoning is highlighted in red. Under original decoding, the model
 228 introduces a transitional reflection (“Wait, no, in circular permutations, fixing one seat...”), generating
 229 32 reflection tokens before arriving at the correct answer. In contrast, with a thought-switching
 230 penalty $\alpha < 0$ in (1), TIP reshapes the trace (“Now, I need to find one person...”), producing only 11
 231 reflection tokens and yielding an incorrect answer. This shows that TIP provides only one-directional
 232 reflection control (penalizing reflection token logits). Hence, a bi-directional, dynamically adaptive
 233 (non-constant) reflection token allocation strategy is needed.

234 **Problem of interest: Resource allocation over reflection tokens.** Reflection tokens has a significant
 235 impact on the reasoning capability of LRM s. Therefore, if we view reflection tokens as a “resource”
 236 in LRM reasoning generation, then determining their schedule, including the number of occurrences
 237 and their positions, naturally gives rise to the *problem of resource allocation* for LRM s.

238 To the best of our knowledge, the problem of resource allocation over reflection tokens remains largely
 239 unexplored in the existing literature. TIP offers a simple solution by applying a constant logit penalty
 240 to reflection token generation. However, this approach is *static* and therefore fails to account for both
 241 the number and placement of reflection tokens, which are dynamically determined during reasoning
 242 trace generation. As shown in Fig. 2(b), TIP does not consistently improve performance across
 243 all difficulty levels. This leaves open the question of how to schedule reflection token generation
 244 along the reasoning trajectory, that is, how to allocate these “resources” effectively over time while
 245 accounting for problem difficulty. These underscore the need for more adaptive and fine-grained
 246 strategies to control reflection token usage in order to address the resource allocation more effectively.

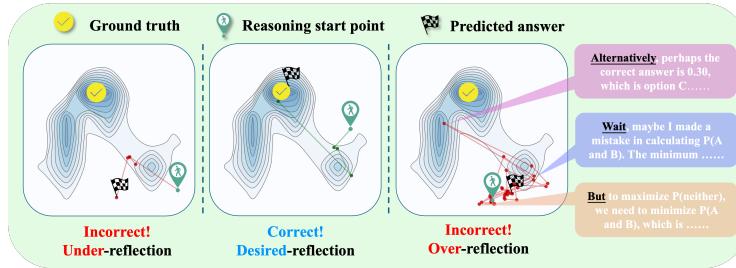
247 4 REFLECTION TOKEN SCHEDULING AS LEARNING RATE SCHEDULING IN 248 OPTIMIZATION

250 In this section, we draw a conceptual analogy between reflection token scheduling and learning rate
 251 scheduling in optimization, aimed at deepening our understanding of reflection tokens in reasoning
 252 and enabling more effective resource allocation. Building on this analogy, we propose a new decoding
 253 strategy: cyclical reflection token scheduling (**CyclicReflex**).

254 **Reflection tokens in the thought landscape vs. learning rates in the optimization landscape.** The
 255 role of reflection tokens in reasoning closely mirrors that of learning rates in optimization. In the
 256 “thought landscape”, a model initiates by interpreting a question and leverages reflection tokens to
 257 modulate its reasoning trajectory: exploring, reconsidering, and refining intermediate steps before
 258 reaching a final answer. Likewise, in the optimization landscape, an optimizer begins from a random
 259 initialization and relies on the learning rate to control the step size of the variable updates, gradually
 260 converging toward an optimal solution. In both cases, a well-tuned control mechanism, reflection
 261 tokens in reasoning or learning rates in optimization, is essential for accurate solution convergence.

262 Additionally, in optimization, an improperly tuned learning rate, either too small or too large, can
 263 hinder convergence, causing the optimizer to either stagnate or diverge. This challenge in scheduling
 264 the learning rate maps naturally onto the difficulty of scheduling reflection tokens in reasoning,
 265 manifesting as under-reflection and over-reflection. (*Under-reflection*) When the model generates
 266 too few reflection tokens, it often terminates the reasoning process prematurely, resulting in a final
 267 answer that lacks sufficient deliberation. This behavior is analogous to optimization with a learning
 268 rate that is too small, where the model converges too early and becomes trapped in a suboptimal
 269 local minimum. (*Over-reflection*) Conversely, generating too many reflection tokens can prevent the
 model from concluding its reasoning, causing it to loop or stall, *e.g.*, repeatedly producing phrases

270 like “wait” without reaching a solution. This resembles optimization with an overly large learning
 271 rate, which leads to instability and divergence rather than convergence.
 272



282 Figure 3: Examples of landscape of thought for under-reflection, desired-reflection, and over-reflection,
 283 generated by DeepSeek-R1-Distill-Qwen-7B with the original decoding strategy. Each point represents a
 284 reasoning step and is connected in the order of generation. Darker regions indicate steps with higher semantic
 285 alignment to the correct answer.

286 To validate the analogy between reflection tokens and learning rates (too small learning rate vs.
 287 under-reflection, and too large learning rate vs. over-reflection), we utilize the interpretability tool
 288 introduced in (Zhou et al., 2025) to visualize the *landscape of thoughts*. This tool projects reasoning
 289 step r_i into a two-dimensional visual space based on the measured “distance” between each step
 290 r_i and the final answer y , providing an interpretable view of the model’s reasoning dynamics. The
 291 distance metric captures the model’s uncertainty by taking the inverse of the probability of generating
 292 the answer y conditioned on the reasoning step r_i , normalized by the length of y :

$$d(r_i, y) = p_{LRM}(y | r_i)^{-1/|y|}, \quad (2)$$

293 where p_{LRM} denotes the prediction probability assigned by the LRM to the answer y given the
 294 reasoning step r_i , and $|y|$ denotes the length of y . Fig. 3 presents a visualized reasoning trajectory
 295 from the initial thought to the final answer under original decoding strategy, across three different
 296 scenarios: (i) under-reflection, where too few reflection tokens lead to a reasoning trace that is
 297 too short and results in an incorrect answer; (ii) desired reflection, which yields a well-structured
 298 reasoning trace and a correct answer; and (iii) over-reflection, where excessive reflection tokens
 299 cause an overly long and off-track reasoning trace, also resulting in an incorrect answer. In the
 300 landscape, darker regions represent intermediate reasoning steps that are semantically closer to the
 301 correct answer. That is, color intensity reflects the relative correctness of each thought along the
 302 trajectory. As we can see, the thought landscape under under-reflection is too conservative to drive
 303 the reasoning process away from the starting point, ultimately failing to converge to the correct final
 304 answer. In contrast, over-reflection could enable the model to reach semantically promising regions
 305 of the landscape, for example, a step like “*Alternatively, perhaps the correct answer is ...*”, which
 306 is far away from the thinking start point and located in the darker region. However, much like an
 307 excessively large learning rate that fails to properly control the optimization process, this leads the
 308 model to quickly pass through the desirable state without settling there, ultimately leading to an
 309 incorrect answer region. Moreover, we find that reflection tokens are responsible for the *sharp turns*
 310 in the reasoning trajectory. By examining the sharply turning steps in over-reflection, we observe that
 311 they are consistently initiated by reflection tokens.
 312

313 CyclicReflex: Cyclical logits manipulation for reflection token

314 **scheduling.** Although reflection tokens are crucial for guiding multi-
 315 step reasoning, balancing their use remains challenging. The need
 316 for dynamic modulation of reflection tokens closely mirrors the chal-
 317 lenge of learning rate scheduling in optimization. As introduced
 318 in Sec. 1, the convergence of gradient descent can be *provably ac-*
 319 *celerated* by adopting the *silver stepsize schedule*, which follows
 320 the principle of *stepsize hedging* (Altschuler & Parrilo, 2024; 2025).
 321 The key algorithmic insight is to hedge between two individually
 322 suboptimal strategies, small and large stepsizes, since the failure
 323 modes of one are often mitigated by the strengths of the other. In
 324 deep model training, cyclical learning rates (Smith, 2017) exemplify
 325 this principle in practice. Rather than using a fixed learning rate, they employ a triangular waveform to periodically alternate between

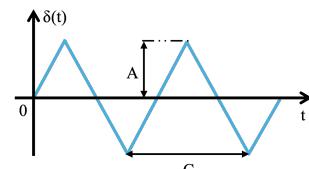


Figure 4: Illustration of CyclicReflex ((3)), where t denotes the token position and $\delta(t)$ the logit adjustment on reflection tokens, oscillating between $-A$ and A with amplitude A and period C .

324 large and small step sizes. This schedule allows the optimizer to balance global exploration (enabled
 325 by large steps) with local convergence stability (provided by smaller steps), thereby yielding a form
 326 of stepsize hedging.

327 Inspired by cyclical learning rates, we introduce **CyclicReflex**. As depicted in [Fig. 4](#), we apply a
 328 periodic triangular waveform to modulate the logits of reflection tokens during generation. The
 329 waveform is governed by two parameters: the amplitude A , which controls the strength of the logit
 330 adjustment, and the period C , which determines the oscillation frequency. This logits manipulation
 331 evolves over time and is both *position-dependent* (varying with each decoding step) and *bidirectional*
 332 (allowing for the dynamic promotion or suppression of reflection token sampling based on the current
 333 stage of the reasoning process). More concretely, CyclicReflex can be cast as the following logits
 334 manipulation as the function of the reasoning step t

$$335 \quad \text{CyclicReflex} : \hat{z}_{t,v} = \begin{cases} z_{t,v} + \delta(t) & \text{if } v \in \hat{V} \\ z_{t,v} & \text{otherwise,} \end{cases} \quad \delta(t) = A \left| 4 \cdot \frac{(t - \frac{C}{4}) \bmod C}{C} - 2 \right| - A \quad (3)$$

336 where recall that the amplitude A and the period C have been previously defined as shown in [Fig. 4](#),
 337 mod is the modulo operation, $|\cdot|$ is the absolute value operation, and the other notations follow [\(1\)](#).
 338 In [\(3\)](#), $(t - \frac{C}{4}) \bmod C$ gives the current thought position within the cycle, and it is straightforward
 339 to validate that $\delta(C/4) = A$ and $\delta(3C/4) = -A$. As shown in [Fig. 4](#), *CyclicReflex implements a*
 340 *representative form of hedging schedule*: the increasing phase of the reflection logit adjustment $\delta(t)$
 341 promotes exploration by encouraging the model to transition away from its current line of thought,
 342 while the decreasing phase fosters convergence by stabilizing the reasoning process, guiding the
 343 model toward producing a coherent and correct final answer.

344 Compared to [TIP \(1\)](#), which applies a fixed unidirectional penalty, CyclicReflex adaptively modulates
 345 reflection token logits with at no additional computation cost, offering finer control over reasoning.
 346 This unified mechanism balances under- and over-reflection, yielding more robust and flexible
 347 behavior that adapts to the model’s evolving thought process. [In Fig. A1, we summarize the sensitivity](#)
 348 [of CyclicReflex to its hyperparameters. The period \$C\$ has a more pronounced impact on accuracy than](#)
 349 [the amplitude \$A\$. In particular, when \$C = 600\$, the model achieves the highest accuracy across all](#)
 350 [tested amplitudes \(\$A = 5.0, 7.0\$, and \$9.0\$ \). Based on our experiments, choosing \$A = 5.0\$ and setting](#)
 351 [\$C\$ to approximately \$0.8 \times\$ the average generation length provides strong performance in practice.](#)

354 5 EXPERIMENTS

355 5.1 EXPERIMENTAL SETUP

356 **Data-model setups.** To evaluate the effectiveness of CyclicReflex, we consider both *math* and
 357 *non-math* benchmarks. The math datasets include **MATH500** ([Lightman et al., 2023](#)) with 500
 358 multi-step problems, **AIME2024/2025** ([MAA Committees](#)) with 30 challenging problems each year,
 359 and **AMC2023** ([AI-MO, 2024](#)) covering diverse competition topics. The non-math datasets include
 360 **GPQA Diamond** ([Rein et al., 2024](#)), a challenging subset of multiple-choice science questions
 361 in biology, chemistry, and physics, and **LiveCodeBench** ([Jain et al., 2024](#)), a coding benchmark
 362 from LeetCode, AtCoder, and Codeforces that evaluates code generation, repair, and execution. Our
 363 experiments are conducted using the publicly available DeepSeek-R1-Distilled-Qwen model family
 364 ([Guo et al., 2025](#)), which includes models with 1.5B, 7B. For comparative analysis, we also include
 365 DeepSeek-R1-Distilled-Llama-8B, enabling a broader evaluation across different backbones.

366 **Baseline and evaluation.** Our method (CyclicReflex) is compared against two primary baselines:
 367 **TIP** ([Wang et al., 2025a](#)), **S1** ([Muennighoff et al., 2025](#)). In addition, we assess the compatibility
 368 of CyclicReflex with external test-time scaling techniques, including **Best-of-N** ([Irvine et al., 2023](#);
 369 [Brown et al., 2024](#)) and **Beam Search** ([Feng et al., 2023](#); [Snell et al., 2024](#)), using RLHFlow-
 370 PRM-Deepseek-8B as the preference reward model (PRM) for scoring ([Dong et al., 2024](#)). We
 371 use **accuracy** and **generation length** as our primary evaluation metrics. Accuracy is obtained by
 372 rule-based extraction of the final answer against the ground truth, while generation length is the total
 373 word count of the response. More implementation details are provided in [Appendix A](#).

374 5.2 EXPERIMENT RESULTS

375 **Overall Performance of CyclicReflex on the MATH Task.** In [Table 2](#), we show the effectiveness of
 376 CyclicReflex across models of varying sizes (1.5B, 7B, and 8B), model families (Qwen and LLaMA),

378 and four widely used reasoning benchmarks: MATH500, AIME2024, AIME2025, and AMC2023. As
 379 we can see, CyclicReflex consistently improves performance over the original LRM decoding strategy
 380 across all models and datasets. For example, DeepSeek-R1-Distill-Llama-8B with CyclicReflex
 381 achieves up to a 10% absolute accuracy gain on AIME2024, while DeepSeek-R1-Distill-Qwen
 382 7B with CyclicReflex yields up to a 9% improvement on AMC2023. Additionally, these accuracy
 383 gains are achieved without sacrificing the efficiency of reasoning generation: CyclicReflex produces
 384 comparable reasoning traces relative to the original decoding method.

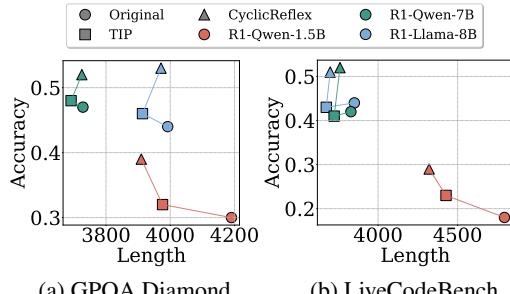
385 Table 2: Accuracy (Acc) and generation length (Len) comparison on four math reasoning benchmarks
 386 (MATH500, AIME2024, AIME2025, and AMC2023) using DeepSeek-R1-Distilled Model: Qwen 1.5B, Qwen
 387 7B, and Llama 8B. Each model is evaluated under four decoding strategies: *Original*, *TIP*, *S1*, and *CyclicReflex*.
 388 The best accuracy in each setting is highlighted in **bold**, while the second-best is underlined.

Method	MATH500		AIME2024		AIME2025		AMC2023	
	Acc	Len	Acc	Len	Acc	Len	Acc	Len
DeepSeek-R1-Distill-Qwen-1.5B								
Original	0.74	1253.05	<u>0.23</u>	3584.36	0.19	3442.07	<u>0.63</u>	1855.85
TIP	<u>0.75</u>	1206.91	<u>0.23</u>	3329.17	<u>0.20</u>	3825.17	<u>0.63</u>	1890.35
S1	0.73	1532.05	0.17	4112.07	<u>0.20</u>	3867.71	0.45	3263.75
CyclicReflex	0.77	1212.94	0.30	3547.10	0.23	3467.97	0.65	1839.23
DeepSeek-R1-Distill-Qwen-7B								
Original	0.86	785.25	<u>0.43</u>	2878.39	0.31	3192.59	0.81	1300.53
TIP	<u>0.87</u>	775.77	<u>0.43</u>	2806.53	0.30	3107.30	<u>0.85</u>	1267.83
S1	0.83	1190.96	0.33	3541.10	<u>0.33</u>	3455.33	<u>0.85</u>	2158.00
CyclicReflex	0.89	777.93	0.50	2868.30	0.37	3190.33	0.90	1229.25
DeepSeek-R1-Distill-Llama-8B								
Original	<u>0.83</u>	1196.98	0.42	3593.73	<u>0.30</u>	3922.41	0.81	1951.88
TIP	<u>0.83</u>	1080.62	<u>0.47</u>	3572.40	0.27	3866.00	<u>0.85</u>	1932.63
S1	0.78	1461.93	0.43	3742.27	0.27	4351.87	0.75	2812.75
CyclicReflex	0.85	1108.30	0.53	3454.97	0.37	3856.80	0.90	1942.40

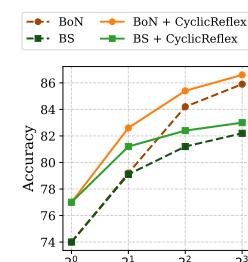
404 We further compare CyclicReflex against two
 405 additional baselines: S1 and TIP. While S1 en-
 406 forces the insertion of “wait” tokens at the end of
 407 each reasoning segment, leading to significantly
 408 longer outputs, it does not yield corresponding
 409 accuracy improvements. On AMC2023, in fact,
 410 S1 causes a notable performance drop, suggest-
 411 ing that excessive reflection may lead to over-
 412 thinking. TIP, which suppresses reflection token
 413 usage, can also degrade performance in some
 414 cases. For instance, TIP causes a 3% accuracy
 415 drop on AIME2025 when applied to DeepSeek-
 416 R1-Distill-Llama-8B, likely because it halts rea-
 417 soning steps that are essential for solving more
 418 complex problems.

419 **Effectiveness of CyclicReflex on non-math reasoning.** Fig. 5 shows the
 420 relationship between accuracy and generation length on two *non-math*
 421 benchmarks, GPQA Diamond and LiveCodeBench. Results are reported for
 422 multiple DeepSeek-R1-Distill variants (Qwen-1.5B/7B, and LLaMA-8B)
 423 under original decoding, TIP, and CyclicReflex. CyclicReflex consistently
 424 improves accuracy while maintaining response lengths comparable to TIP.
 425 In contrast, TIP can even reduce accuracy, as seen in Fig. 5(b) for DeepSeek-
 426 R1-Distill-Qwen-7B and LLaMA-8B on LiveCodeBench.

427 **Integration with other test-time scaling methods.** In Fig. 6, we further
 428 investigate the integration of CyclicReflex with other test-time scaling meth-
 429 ods across computational budgets (2^0 to 2^3), using DeepSeek-R1-Distill-
 430 Qwen-1.5B on MATH500. We evaluate both Best-of-N (BoN) and Beam
 431 Search (BS), with generations scored using RLHF-PRM-DeepSeek-8B.
 Across all budget levels, BoN and Beam Search integrated with ours



432 Figure 5: Accuracy vs. generation length on (a) GPQA Diamond
 433 and (b) LiveCodeBench. The comparison in-
 434 cludes the original decoding, TIP, and CyclicReflex on
 435 DeepSeek-R1-Distill-Qwen 1.5B/7B, and Llama 8B.



436 Figure 6: MATH500
 437 accuracy of DeepSeek-
 438 1.5B under BoN/BS w/o
 439 CyclicReflex.

432 consistently outperform their original counterparts, demonstrating
 433 the general compatibility and effectiveness of our method. Moreover,
 434 under fixed decoding strategies, BoN achieves higher accuracy than
 435 Beam Search, both with and without CyclicReflex. As the computa-
 436 tional budget increases, the performance gap between CyclicReflex
 437 and the original decoding narrows, highlighting that CyclicReflex
 438 offers the greatest benefit under constrained inference budgets by
 439 enabling more efficient reflection token allocation.

440 **Reflection token scheduling patterns of CyclicReflex.** Fig. 7 com-
 441 pares reflection token distributions under original decoding, TIP,
 442 and CyclicReflex, using DeepSeek-R1-Distill-Llama-8B on AIME2024.
 443 Each curve shows the proportion of reflection tokens within con-
 444 secutive 1000-token segments relative to the total reflection count.
 445 Original decoding exhibits a gradual early rise followed by a stable,
 446 evenly spread pattern. TIP follows a similar trajectory but suppresses
 447 reflection in the 0–1k range, reflecting its tendency to inhibit early
 448 reflection. In contrast, CyclicReflex displays a cyclical hedging pat-
 449 tern with alternating peaks and troughs within the reasoning trace,
 450 allocating more reflection in the 1–2k and 3–4k ranges. This mod-
 451 ulation avoids both excessive early suppression and late overuse,
 452 leading to stronger performance on AIME2024, improving accuracy from 0.42 (original decoding) to
 453 0.53, demonstrating the benefit of bidirectional, position-dependent reflection scheduling.

454 **Robustness to the reflection-token**
 455 **set.** Next, we show that the proposed
 456 CyclicReflex is robust to the choice of
 457 reflection tokens and can even operate
 458 with a dynamically updated reflection-
 459 token set. To validate this, we con-
 460 sider a dynamic variant in which the
 461 reflection-token set is expanded online
 462 during decoding: we start from a small seed set (e.g., “wait”, “but”), and at decoding step t , if the
 463 Top-1 token belongs to the current reflection-token set and the logit gap between Top-2 and Top-1
 464 is smaller than the gap between Top-2 and Top-3, we add the Top-2 token to the set. The rationale
 465 is that the model tends to use in contexts similar to existing reflection tokens, gradually enriching
 466 the set and yielding additional reflection-like tokens (e.g., “hmm”, “maybe”, “perhaps”), which are
 467 absent from the original reflection-token set. We evaluate CyclicReflex with both fixed and dynamic
 468 reflection-token sets on DeepSeek-R1-Distill-Qwen-7B across MATH500, AIME2024, AIME2025,
 469 and AMC2023, and compare against the Original decoding and TIP. As shown in Table 3, the two
 470 variants perform comparably across all benchmarks, indicating that CyclicReflex exhibits robustness
 471 to the selection of reflection tokens.

472 **Comparison with approaches us-
 473 ing additional reflection-logit ad-
 474 justments.** In Table 4, we extend our
 475 comparison on DeepSeek-R1-Distill-
 476 Qwen-7B by adding several additional
 477 adjustment-based baselines: (1) *TIP*
 478 w/ *positive adjustment*, which always
 479 boosts reflection-token logits rather
 480 than penalizing them; (2) *random*
 481 *adjustment*, which injects zero-mean
 482 noise into the reflection-token logits
 483 at every decoding step; and (3) *linear decay*,
 484 which begins with a strong positive boost that linearly
 485 decreases into a penalty across the generation process. Across MATH500, AIME2024, AIME2025,
 486 and AMC2023, CyclicReflex consistently achieves the highest accuracy. In contrast, both the positive-
 487 adjustment and random-adjustment variants underperform even standard TIP, and the linear-decay
 488 baseline narrows, but does not close, the performance gap relative to our method. These findings
 489 indicate that naïve or purely monotonic adjustments to reflection-token logits are insufficient, and

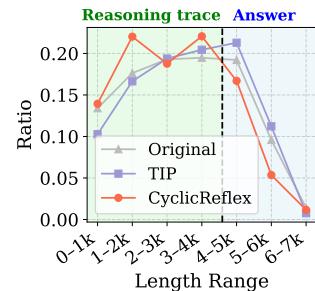


Figure 7: Reflection token distribution of DeepSeek-R1-Distill-Llama-8B on AIME2024 under original decoding, TIP, and CyclicReflex. Each curve shows the proportion of reflection tokens within 1k-token segments relative to the total generation, including both reasoning trace and answer.

Table 3: Accuracy of *Original*, *TIP*, and *CyclicReflex* variants on MATH500, AIME2024, AIME2025, and AMC2023 with DeepSeek-R1-Distill-Qwen-7B.

Method	MATH500	AIME2024	AIME2025	AMC2023
Original	0.86	0.43	0.31	0.81
TIP	0.87	0.43	0.30	0.85
CyclicReflex w/ dynamic set	0.89	0.50	0.37	0.91
Ours: CyclicReflex	0.89	0.50	0.37	0.90

Table 4: Performance of *Original*, *TIP*, and adjustment-based variants on MATH500, AIME2024, AIME2025, and AMC2023 with DeepSeek-R1-Distill-Qwen-7B.

Method	MATH500	AIME2024	AIME2025	AMC2023
Original	0.86	0.43	0.31	0.81
TIP	0.87	0.43	0.30	0.85
TIP w/ positive adjustment	0.85	0.40	0.33	0.83
Random adjustment	0.82	0.41	0.30	0.79
Linear decay	0.87	0.45	0.33	0.87
Ours: CyclicReflex	0.89	0.50	0.37	0.90

486 that the *cyclic hedging schedule* in CyclicReflex provides a fundamentally more effective mechanism
 487 for balancing exploration and control during reasoning.
 488

489 **Generalization to other model families and**
 490 **scales.** We further extend CyclicReflex to a dif-
 491 ferent model family by evaluating Qwen3-4B,
 492 Qwen3-8B and Qwen3-14B on MATH500,
 493 AIME2024, AIME2025, AMC2023 and Live-
 494 CodeBench, with *Original* and *TIP* included as
 495 baselines. The results are provided in **Table 5**.
 496 Across all five benchmarks, CyclicReflex con-
 497 sistently outperforms both baselines, even as the
 498 underlying model scale increases and the fam-
 499 ily differs from the DeepSeek-distilled lineage.
 500 This demonstrates that our method generalizes
 501 beyond a single family of reasoning models and
 502 remains robust and effective for a larger model
 503 with an alternative architecture.

504 **Evaluation under pass@N and**
 505 **cons@N.** In **Table 6**, we evaluate
 506 CyclicReflex on MATH500 using
 507 DeepSeek-R1-Distill-Qwen-7B under
 508 both pass@N and cons@N, with *Orig-*
 509 *inal* and *TIP* as baselines. Cyclic-
 510 reflex consistently outperforms both
 511 baselines across all values of N , show-
 512 ing that our approach improves standard test-time scaling metrics rather than only single-sample
 513 accuracy. While *TIP* provides modest gains over *Original* for small N (e.g., pass@1 and pass@2),
 514 these improvements diminish or even reverse as N increases. This is likely because aggressively
 515 suppressing reflection tokens reduces sampling diversity, limiting the benefit of drawing more trajec-
 516 tories. In contrast, CyclicReflex maintains a more stable advantage as N increases, highlighting its
 517 better balance between control and exploration.

518 **Difficulty-level accuracies, self-correction and examples of CyclicReflex.** As shown in **Fig. A2**
 519 of **Appendix B**, unlike *TIP*, which only improves accuracy on the Hard problems of MATH500,
 520 CyclicReflex enhances accuracy across all difficulty levels. **Fig. A3** of **Appendix B** further demon-
 521 strates that when provided with an incorrect reasoning trace as a prompt, CyclicReflex can correct a
 522 larger proportion of erroneous traces than *TIP* or the original decoding strategy, indicating enhanced
 523 self-correction ability. **Table A1** in **Appendix B** shows that, compared with waveform design, the
 524 hedging pattern plays a more critical role. Finally, **Table A2** in **Appendix C** provides generation
 525 examples under both the original and CyclicReflex decoding.

526 6 CONCLUSION

527 We introduce the problem of resource allocation in LRM_s, focusing on the challenge of managing
 528 reflection tokens during test-time generation. We show that both under-reflection and over-reflection,
 529 stemming from insufficient or excessive use of reflection tokens, can severely degrade reasoning
 530 performance. To address this, we draw a conceptual analogy between reflection token scheduling and
 531 learning rate control in optimization, and propose CyclicReflex, a training-free decoding strategy that
 532 cyclically modulates reflection token logits using a triangular waveform. CyclicReflex dynamically
 533 adapts to the evolving stage of reasoning, enabling more balanced token allocation. Extensive exper-
 534 iments across multiple reasoning benchmarks demonstrate that CyclicReflex consistently improves
 535 accuracy, enhances self-correction capability, and integrates seamlessly with existing test-time scaling
 536 methods. Our work highlights the critical role of reflection tokens as a valuable resource for LRM_s
 537 and opens new avenues for principled, adaptive reasoning control. The use of LLM, limitations and
 538 broader impacts are further discussed in **Appendix D**, **Appendix E** and **Appendix F**.

540 REFERENCES
541

542 Pranjal Aggarwal and Sean Welleck. L1: Controlling how long a reasoning model thinks with
543 reinforcement learning. *arXiv preprint arXiv:2503.04697*, 2025.

544 AI-MO. Amc 2023, 2024. URL <https://huggingface.co/datasets/AI-MO/aimo-validation-amc>.

545

546 Zeyuan Allen-Zhu and Lorenzo Orecchia. Linear coupling: An ultimate unification of gradient and
547 mirror descent. *arXiv preprint arXiv:1407.1537*, 2014.

548

549 Jason M Altschuler and Pablo A Parrilo. Acceleration by stepsize hedging: Silver stepsize schedule
550 for smooth convex optimization. *Mathematical Programming*, pp. 1–14, 2024.

551

552 Jason M Altschuler and Pablo A Parrilo. Acceleration by stepsize hedging: Multi-step descent and
553 the silver stepsize schedule. *Journal of the ACM*, 72(2):1–38, 2025.

554

555 Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V Le, Christopher Ré, and
556 Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling.
557 *arXiv preprint arXiv:2407.21787*, 2024.

558

559 Sébastien Bubeck, Yin Tat Lee, and Mohit Singh. A geometric alternative to nesterov’s accelerated
560 gradient descent. *arXiv preprint arXiv:1506.08187*, 2015.

561

562 Qiguang Chen, Libo Qin, Jinhao Liu, Dengyun Peng, Jiannan Guan, Peng Wang, Mengkang Hu,
563 Yuhang Zhou, and Wanxiang Che. Towards reasoning era: A survey of long chain-of-thought for
564 reasoning large language models. *arXiv preprint arXiv:2503.09567*, 2025a.

565

566 Runjin Chen, Zhenyu Zhang, Junyuan Hong, Souvik Kundu, and Zhangyang Wang. Seal: Steerable
567 reasoning calibration of large language models for free. *arXiv preprint arXiv:2504.07986*, 2025b.

568

569 Weizhe Chen, Sven Koenig, and Bistra Dilkina. Iterative deepening sampling as efficient test-time
570 scaling. *arXiv preprint arXiv:2502.05449*, 2025c.

571

572 Xingyu Chen, Jiahao Xu, Tian Liang, Zhiwei He, Jianhui Pang, Dian Yu, Linfeng Song, Qizhi Liu,
573 Mengfei Zhou, Zhuosheng Zhang, et al. Do not think that much for $2+3=?$ on the overthinking of
574 o1-like llms. *arXiv preprint arXiv:2412.21187*, 2024.

575

576 Vito Ding et al. Llm post-training: A deep dive into reasoning large language models. *arXiv preprint
577 arXiv:2502.21321*, 2025.

578

579 Hanze Dong, Wei Xiong, Bo Pang, Haoxiang Wang, Han Zhao, Yingbo Zhou, Nan Jiang, Doyen
580 Sahoo, Caiming Xiong, and Tong Zhang. Rlhf workflow: From reward modeling to online rlhf.
581 *arXiv preprint arXiv:2405.07863*, 2024.

582

583 Xidong Feng, Ziyu Wan, Muning Wen, Stephen Marcus McAleer, Ying Wen, Weinan Zhang, and Jun
584 Wang. Alphazero-like tree-search can guide large language model decoding and training. *arXiv
585 preprint arXiv:2309.17179*, 2023.

586

587 Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu,
588 Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms
589 via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

590

591 Bairu Hou, Yang Zhang, Jiabao Ji, Yujian Liu, Kaizhi Qian, Jacob Andreas, and Shiyu Chang.
592 Thinkprune: Pruning long chain-of-thought of llms via reinforcement learning. *arXiv preprint
593 arXiv:2504.01296*, 2025.

594

595 Robert Irvine, Douglas Boubert, Vyas Raina, Adian Liusie, Ziyi Zhu, Vineet Mudupalli, Aliaksei
596 Korshuk, Zongyi Liu, Fritz Cremer, Valentin Assassi, et al. Rewarding chatbots for real-world
597 engagement with millions of users. *arXiv preprint arXiv:2303.06135*, 2023.

598

599 Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando
600 Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free
601 evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*, 2024.

594 Hyunbin Jin, Junwoo Yeom, Soobin Bae, and Taesup Kim. “well, keep thinking”: Enhancing llm
 595 reasoning with adaptive injection decoding. *arXiv preprint arXiv:2503.10167*, 2025.

596

597 Zhewei Kang, Xuandong Zhao, and Dawn Song. Scalable best-of-n selection for large language
 598 models via self-certainty. *arXiv preprint arXiv:2502.18581*, 2025.

599 Abhinav Kumar, Jaechul Roh, Ali Naseh, Marzena Karpinska, Mohit Iyyer, Amir Houmansadr,
 600 and Eugene Bagdasarian. Overthink: Slowdown attacks on reasoning llms. *arXiv e-prints*, pp.
 601 arXiv–2502, 2025a.

602 Komal Kumar, Tajamul Ashraf, Omkar Thawakar, Rao Muhammad Anwer, Hisham Cholakkal,
 603 Mubarak Shah, Ming-Hsuan Yang, Phillip H. S. Torr, Salman Khan, and Fahad Shahbaz Khan. Llm
 604 post-training: A deep dive into reasoning large language models. *arXiv preprint arXiv:2502.21321*,
 605 2025b. URL <https://arxiv.org/abs/2502.21321>.

606

607 Zhong-Zhi Li, Duzhen Zhang, Ming-Liang Zhang, Jiaxin Zhang, Zengyan Liu, Yuxuan Yao, Haotian
 608 Xu, Junhao Zheng, Pei-Jie Wang, Xiuyi Chen, Yingying Zhang, Fei Yin, Jiahua Dong, Zhijiang
 609 Guo, Le Song, and Cheng-Lin Liu. From system 1 to system 2: A survey of reasoning large
 610 language models. *arXiv preprint arXiv:2502.17419*, 2025.

611 Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan
 612 Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let’s verify step by step. In *The Twelfth
 613 International Conference on Learning Representations*, 2023.

614 Runze Liu, Junqi Gao, Jian Zhao, Kaiyan Zhang, Xiu Li, Biqing Qi, Wanli Ouyang, and Bowen
 615 Zhou. Can 1b llm surpass 405b llm? rethinking compute-optimal test-time scaling. *arXiv preprint
 616 arXiv:2502.06703*, 2025.

617

618 Haotian Luo, Li Shen, Haiying He, Yibo Wang, Shiwei Liu, Wei Li, Naiqiang Tan, Xiaochun Cao,
 619 and Dacheng Tao. O1-pruner: Length-harmonizing fine-tuning for o1-like reasoning pruning.
 620 *arXiv preprint arXiv:2501.12570*, 2025.

621 MAA Committees. Aime problems and solutions. [https://artofproblemsolving.com/
 622 wiki/index.php/AIME_Problems_and_Solutions](https://artofproblemsolving.com/wiki/index.php/AIME_Problems_and_Solutions).

623 Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke
 624 Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. s1: Simple test-time
 625 scaling. *arXiv preprint arXiv:2501.19393*, 2025.

626

627 Yurii Nesterov. A method for solving the convex programming problem with convergence rate o
 628 $(1/k^2)$. In *Dokl akad nauk Sssr*, volume 269, pp. 543, 1983.

629

630 OpenAI. Openai o1 system card. *arXiv preprint arXiv:2412.16720*, 2024. URL <https://arxiv.org/abs/2412.16720>.

631

632 David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani,
 633 Julian Michael, and Samuel R Bowman. Gpqa: A graduate-level google-proof q&a benchmark. In
 634 *First Conference on Language Modeling*, 2024.

635 Ranajoy Sadhukhan, Zhuoming Chen, Haizhong Zheng, Yang Zhou, Emma Strubell, and Beidi Chen.
 636 Kinetics: Rethinking test-time scaling laws. *arXiv preprint arXiv:2506.05333*, 2025.

637

638 Leslie N Smith. Cyclical learning rates for training neural networks. In *2017 IEEE winter conference
 639 on applications of computer vision (WACV)*, pp. 464–472. IEEE, 2017.

640 Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally
 641 can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

642

643 Jinyan Su, Jennifer Healey, Preslav Nakov, and Claire Cardie. Between underthinking and over-
 644 thinking: An empirical study of reasoning length and correctness in llms. *arXiv preprint
 645 arXiv:2505.00127*, 2025.

646

647 Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun
 648 Xiao, Chenzhuang Du, Chonghua Liao, et al. Kimi k1. 5: Scaling reinforcement learning with
 649 llms. *arXiv preprint arXiv:2501.12599*, 2025.

648 Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdh-
 649 ery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models.
 650 *arXiv preprint arXiv:2203.11171*, 2022.

651

652 Yue Wang, Qizhi Liu, Jiahao Xu, Tian Liang, Xingyu Chen, Zhiwei He, Linfeng Song, Dian Yu,
 653 Juntao Li, Zhuosheng Zhang, Rui Wang, Zhaopeng Tu, Haitao Mi, and Dong Yu. Thoughts are all
 654 over the place: On the underthinking of o1-like llms. *arXiv preprint arXiv:2501.18585*, 2025a.

655 Yuxuan Wang, Kai Zhang, Zhiwei Wang, Juntao Li, Dian Yu, Zhaopeng Tu, Haitao Mi, and Dong Yu.
 656 Speculative thinking: Enhancing small-model reasoning with large model guidance at inference
 657 time. *arXiv preprint arXiv:2504.12329*, 2025b.

658

659 Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi,
 660 Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language
 661 models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022. URL <https://arxiv.org/abs/2201.11903>.

662

663 Tong Wu, Chong Xiang, Jiachen T Wang, and Prateek Mittal. Effectively controlling reasoning
 664 models through thinking intervention. *arXiv preprint arXiv:2503.24370*, 2025.

665

666 An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li,
 667 Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2. 5 technical report. *arXiv preprint
 668 arXiv:2412.15115*, 2024.

669 Chenxu Yang, Qingyi Si, Yongjie Duan, Zheliang Zhu, Chenyu Zhu, Zheng Lin, Li Cao, and Weiping
 670 Wang. Dynamic early exit in reasoning models. *arXiv preprint arXiv:2504.15895*, 2025.

671 Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan.
 672 Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural
 673 information processing systems*, 36:11809–11822, 2023.

674

675 Junyu Zhang, Runpei Dong, Han Wang, Xuying Ning, Haoran Geng, Peihao Li, Xialin He, Yutong
 676 Bai, Jitendra Malik, Saurabh Gupta, et al. Alphaone: Reasoning models thinking slow and fast at
 677 test time. *arXiv preprint arXiv:2505.24863*, 2025.

678 Andrew Zhou, Yuxuan Wang, Kai Zhang, Zhiwei Wang, Juntao Li, Dian Yu, Zhaopeng Tu, Haitao
 679 Mi, and Dong Yu. Landscape of thoughts: Visualizing the reasoning process of large language
 680 models. *arXiv preprint arXiv:2503.22165*, 2025.

681

682 Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language
 683 agent tree search unifies reasoning acting and planning in language models. *arXiv preprint
 684 arXiv:2310.04406*, 2023.

685

686

687

688

689

690

691

692

693

694

695

696

697

698

699

700

701

APPENDIX

A DETAILED EXPERIMENT SETUPS

A.1 COMPUTING RESOURCES

All experiments are conducted on a single node equipped with 8 NVIDIA A6000 GPUs.

A.2 DECODING DETAILS

During generation, we employ the vLLM framework to enable efficient and scalable inference. The decoding configuration is as follows: the maximum number of new tokens is set to 8192; the top- p value is set to 0.95; and the temperature is set to 0.6. Top- p sampling selects tokens from the smallest possible set whose cumulative probability exceeds p , effectively filtering out low-probability candidates to maintain generation quality while allowing diversity. A temperature of 0.6 sharpens the token probability distribution, promoting more deterministic and focused outputs by reducing sampling randomness.

For CyclicReflex on the MATH500 and AMC2023 datasets, we perform a grid search over $A \in [1, 10]$ and $C \in [200, 1000]$. On the AIME2024 and AIME2025 datasets, we perform a grid search over $A \in [1, 10]$ and $C \in [1000, 2000]$. For TIP, we conduct a grid search with $\alpha \in [-10, -1]$ and $T_0 \in [100, 1000]$. For S1, we forcefully insert the reflection token ‘‘Wait’’ after the model generates `</think>`, prompting continued reasoning.

In the Best-of-N setting, the LRM generates multiple independent candidate answers, and the PRM selects the most preferred one based on final-answer evaluation. For Beam Search, we perform a step-by-step search guided by PRM feedback to optimize cumulative reward. Throughout decoding, we use multiple candidate beams with a fixed beam width of 4.

A.3 PROMPT DETAILS

We present the prompt used to evaluate the reasoning ability of the LRM. For each question, we replace the `{question}` placeholder in the User section of the prompt. After the Assistant generates the reasoning trace and the final answer (`{generation}`), we follow the approach of [Yang et al. \(2024\)](#) to first extract the final answer and then apply rule-based matching to assess its correctness.

Evaluation prompt

System:
You are a helpful AI bot that answers questions for a user. Keep your response short and direct.

User:
Question: {question}
Let’s reason this step by step.

Assistant:
Answer: {generation}

B ADDITIONAL EXPERIMENT RESULTS

Ablation study on CyclicReflex’s hyperparameter. In [Fig. A1](#), we analyze the effect of CyclicReflex’s key hyperparameters on final performance. Based on [Fig. 4](#), we focus on three parameters: the amplitude A , the period C , and an additional controlling factor, the initial phase shift, denoted by ϕ . As shown in [Fig. A1\(a\)](#), the period C has a more pronounced impact on accuracy than the amplitude A . In particular, when $C = 600$, the model achieves the highest accuracy across all tested amplitudes ($A = 5.0, 7.0$, and 9.0).

In addition, **Fig. A1(b)** and **(c)** show that the amplitude A primarily influences the number of reflection tokens and the overall generation length. Specifically, increasing A leads to more frequent reflection token generation and longer output sequences, confirming that A effectively controls the model’s propensity for extended reasoning.

Finally, **Fig A1(d)** examines the effect of the initial phase shift ϕ by measuring the number of additional correct answers relative to the original decoding strategy. We find that $\phi = 0$ yields the best performance (*i.e.*, with the pattern in Fig. 4), indicating that encouraging reflection token generation early in the reasoning process is beneficial. As reasoning progresses, gradually suppressing reflection token logits helps the model converge more efficiently.

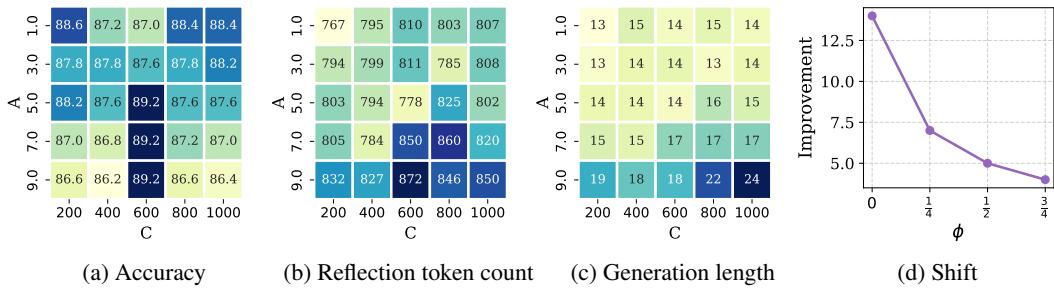


Figure A1: (a)–(c) Accuracy, reflection token count, and generation length heatmaps of DeepSeek-R1-Distill-Qwen-7B on MATH500 under different amplitude values A and period values C . (d) Improvement of CyclicReflex over the original decoding strategy under different initial phase shifts ϕ , measured as the number of additional correct answers.

Accuracy of CyclicReflex at different difficulty levels on MATH500. In **Fig. A2**, we categorize the MATH500 dataset by difficulty level to closely examine where the accuracy improvements from CyclicReflex are most pronounced. The grouping strategy follows that used in Fig. 2(a), and the accuracy is reported in a manner consistent with Fig. 2(b). For comparison, we also include TIP as a baseline. We observe that CyclicReflex consistently improves accuracy across all difficulty levels (Easy, Medium, and Hard) whereas TIP primarily yields gains on Hard problems and even leads to performance degradation on Easy and Medium ones. This contrast stems from the bidirectional nature of CyclicReflex, which allows for dynamic promotion or suppression of reflection token sampling based on the current stage of the reasoning process. Such flexibility enables CyclicReflex to better adapt to problem difficulty, leading to more effective resource allocation and improved overall performance.

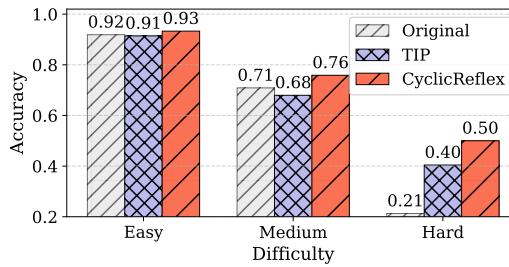


Figure A2: Improvement of DeepSeek-R1-Distill-Qwen-7B on MATH500 by TIP and CyclicReflex across difficulty levels (Easy, Medium, Hard), following Fig. 2 (a) and (b).

Improved self-correction with CyclicReflex. We also find that CyclicReflex exhibits a stronger capacity for self-correction during reasoning. To evaluate this property, we select 50 incorrectly answered problems from the MATH500 dataset, originally generated by DeepSeek-R1-Distill-Qwen-7B. For each incorrect case, we extract the model’s reasoning trace and truncate it to three different lengths (25%, 50%, and 100% of the full trace), which are then reused as misleading prompts to guide a new round of reasoning.

810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
Under each prompt condition, we prompt the same model (DeepSeek-R1-Distill-Qwen-7B) to re-answer the question five times and report the average accuracy. As shown in **Fig. A3(a)**, CyclicReflex significantly outperforms both the original decoding and the TIP baseline across all trace lengths.

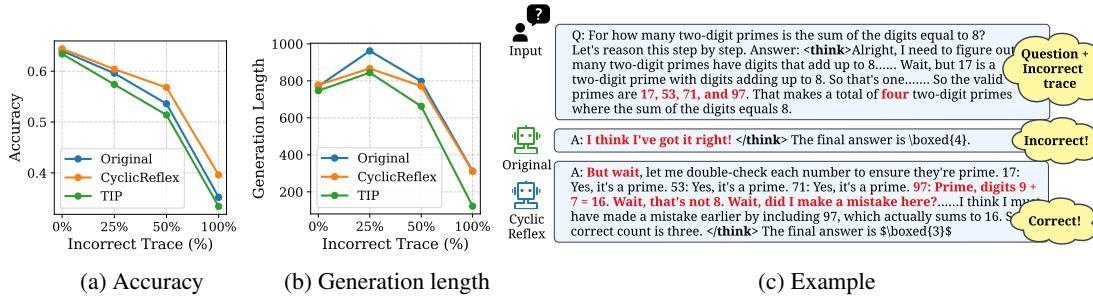


Figure A3: (a)-(b) Accuracy and generation length on MATH500 with DeepSeek-R1-Distill-Qwen-7B using Original, TIP, and CyclicReflex decoding for questions combined with incorrect reasoning traces of different lengths. (c) Example for a question with 100% incorrect reasoning under Original and CyclicReflex decoding.

Notably, the performance gap increases with longer misleading traces, indicating that CyclicReflex enhances the model’s ability to resist and correct earlier reasoning errors. In addition, this improved self-correction does not come at the cost of generation efficiency. As shown in **Fig. A3(b)**, the generation lengths under CyclicReflex remain comparable to those of the original decoding strategy. In contrast, TIP tends to suppress reflection token usage, which hampers the model’s ability to re-evaluate its own reasoning and results in markedly lower accuracy. Finally, **Fig. A3(c)** provides a concrete example. Given a MATH500 question with a 100% incorrect reasoning trace in which the model incorrectly claims that the digits of 17, 53, 71, and 97 sum to 8, the original decoding strategy fails to correct the error and outputs the wrong answer, 4. In contrast, CyclicReflex initiates a double-check, correctly identifies the error (specifically excluding 97), and ultimately outputs the correct answer, 3.

Impact of waveform design and hedging schedule. In **Table A1**, we investigate the impact of different waveform choices in reflection token scheduling. Specifically, we replace the triangular wave in Eq. (3) with a sine function (*Sin*), which also satisfies the hedging schedule property, and evaluate the performance on AIME2024 using DeepSeek-R1-Distill-Llama-8B. The results show that both the sine schedule and CyclicReflex outperform TIP and the original decoding strategy, with the two achieving comparable performance. This suggests that while the exact waveform design has only a minor effect, adopting a hedging schedule yields substantial improvements in accuracy and efficiency.

Table A1: Performance of *Original*, *TIP*, *Sin*, and *CyclicReflex* on AIME2024 with DeepSeek-R1-Distill-Llama-8B.

Method	Acc	Len
Original	0.42	3593.73
TIP	0.45	3872.93
Sin	0.51	3628.37
CyclicReflex	0.53	3454.97

C GENERATION EXAMPLES

In **Table A2-A5**, we present the answers generated for questions in MATH500 under both the original decoding strategy and CyclicReflex. For better comparison, we also include the ground truth labels. Compared to the original decoding strategy, CyclicReflex demonstrates deeper reasoning and avoids prematurely terminating the thinking process. For instance, in Q1, the original answer is given as $\binom{14}{5} - \binom{8}{5} - \binom{6}{5}$, whereas CyclicReflex continues simplifying the expression to arrive at the final result. Moreover, CyclicReflex exhibits a better understanding of the problem semantics. In Q2, for example, the original decoding misinterprets the notion of projection, leading to an incorrect answer. Finally, CyclicReflex is more adept at identifying key problem elements. In Q3, the original response overlooks the trap involving $x = 9$, while CyclicReflex correctly accounts for this subtlety.

864

865

866

867

868

869

870

871

872

873

874

875

Table A2: Examples from DeepSeek-R1-Distill-Qwen-7B on MATH500 under the original decoding strategy and CyclicReflex. Reflection tokens are highlighted in red.

876

877

878

879

Q1: A Senate committee has 8 Republicans and 6 Democrats. In how many ways can we form a subcommittee of 5 members that has at least one member from each party?	
880	Label:
881	Original:
882	<think> Okay, so I have this problem where there's a Senate committee with 8 Republicans and 6 Democrats. I need to figure out how many ways we can form a subcommittee of 5 members that has at least one member from each party. Hmm, let's break this down.
883	First, I know that the total number of people on the committee is 8 + 6, which is 14. If I were to form a subcommittee of 5 without any restrictions, the number of ways would just be the combination of 14 taken 5 at a time, right? So that's C(14,5). But the problem says there has to be at least one from each party, so I can't have all Republicans or all Democrats in the subcommittee. So, I think I need to subtract the cases where the subcommittee is all Republicans or all Democrats. Let me calculate those separately. For all Republicans, it's C(8,5) because there are 8 Republicans. For all Democrats, it's C(6,5) since there are 6 Democrats.
884	Putting it all together, the total number of valid subcommittees should be C(14,5) minus C(8,5) minus C(6,5). Let me make sure that makes sense. By subtracting the all-Republican and all-Democrat cases from the total, I'm left with only those subcommittees that have at least one from each party. Yeah, that seems right.
885	I should probably calculate these values to get the exact number. but I think the reasoning is solid. So the formula is C(14) - C(8,5) - C(6,5). I'll go with that.
886	</think>
887	The number of ways is calculated by subtracting the all-Republican and all-Democrat subcommittees from the total.
888	Total ways: $\binom{14}{5} - \binom{8}{5} - \binom{6}{5}$
889	Answer: $\boxed{\binom{14}{5} - \binom{8}{5} - \binom{6}{5}}$
890	CyclicReflex:
891	<think> Okay, so I have this problem where there's a Senate committee with 8 Republicans and 6 Democrats. I need to figure out how many ways we can form a subcommittee of 5 members that has at least one member from each party. Hmm, let's break this down.
892	First, I know that the total number of people on the committee is 8 + 6, which is 14. If I were to form a subcommittee of 5 without any restrictions, the number of ways would just be the combination of 14 taken 5 at a time, right? So that's C(14,5). But the problem says there has to be at least one from each party, so I can't have all Republicans or all Democrats in the subcommittee. So, I think I need to subtract the cases where the subcommittee is all Republicans or all Democrats. Let me calculate those separately. For all Republicans, it's C(8,5) because there are 8 Republicans. For all Democrats, it's C(6,5) since there are 6 Democrats.
893	But hold on, I should make sure these numbers are correct. C(8,5) is the number of ways to choose 5 Republicans out of 8, and C(6,5) is choosing 5 Democrats out of 6. I think that's right.
894	So, the total number of valid subcommittees would be C(14,5) minus C(8,5) minus C(6,5). Let me compute those values.
895	C(14,5) is 2002, C(8,5) is 56, and C(6,5) is 6. So, 2002 - 56 - 6 equals 1940. That should be the number of ways to form the subcommittee with at least one from each party.
896	</think>
897	The number of ways to form a subcommittee of 5 members with at least one member from each party is calculated by subtracting the all-Republican and all-Democrat subcommittees from the total number of possible subcommittees.
898	- Total subcommittees: $\binom{14}{5} = 2002$
899	- All-Republican subcommittees: $\binom{8}{5} = 56$
900	- All-Democrat subcommittees: $\binom{6}{5} = 6$
901	Thus, the number of valid subcommittees is $2002 - 56 - 6 = 1940$.
902	Answer: 1940

903

904

905

906

907

908

909

910

911

912

913

914

915

916

917

918
919
920
921
922
923
924
925
926

Table A3: Examples from DeepSeek-R1-Distill-Qwen-7B on MATH500 under the original decoding strategy and CyclicReflex. Reflection tokens are highlighted in red.

	<p>Q2: Find the projection of \mathbf{a} onto $\mathbf{b} = \begin{pmatrix} 2 \\ 6 \\ 3 \end{pmatrix}$, if $\mathbf{a} \cdot \mathbf{b} = 8$.</p>
Label:	$\begin{pmatrix} 16 \\ 49 \\ 48 \\ 49 \\ 24 \\ 49 \end{pmatrix}$
Original:	<p><think> Okay, so I need to find the projection of vector \mathbf{a} onto vector \mathbf{b}. I remember that the formula for the projection of \mathbf{a} onto \mathbf{b} is $(\mathbf{a} \cdot \mathbf{b})$ divided by the magnitude of \mathbf{b} squared, and then multiplied by vector \mathbf{b}. But wait, in this case, they only gave me that the dot product of \mathbf{a} and \mathbf{b} is 8. Hmm, so maybe I don't need the entire vector \mathbf{b}, just the scalar projection?</p>
	<p>Let me think. The formula for the scalar projection is $(\mathbf{a} \cdot \mathbf{b})$ divided by the magnitude of \mathbf{b}. Since they already gave me $\mathbf{a} \cdot \mathbf{b} = 8$, I just need to find the magnitude of \mathbf{b}. The vector \mathbf{b} is $[2, 6, 3]$, so its magnitude is the square root of $(2^2 + 6^2 + 3^2)$. Calculating that, $2^2 = 4$, $6^2 = 36$, and $3^2 = 9$. Adding those up gives $4 + 36 + 9 = 49$. The square root of 49 is 7. So the magnitude of \mathbf{b} is 7.</p>
	<p>Therefore, the scalar projection of \mathbf{a} onto \mathbf{b} is 8 divided by 7, which is $8/7$. I think that's it. The problem didn't specify whether they wanted the vector projection or just the scalar. but since they only gave me a scalar product, I guess it's the scalar projection they're asking for. So the answer should be $8/7$.</p>
	<p></think></p>
	<p>The projection of \mathbf{a} onto \mathbf{b} is given by $\frac{\mathbf{a} \cdot \mathbf{b}}{\ \mathbf{b}\ }$. Since $\mathbf{a} \cdot \mathbf{b} = 8$ and $\ \mathbf{b}\ = 7$, the projection is $\frac{8}{7}$.</p>
Answer:	$\frac{8}{7}$
CyclicReflex:	<p><think> Okay, so I need to find the projection of vector \mathbf{a} onto vector \mathbf{b}. I remember that the formula for the projection of \mathbf{a} onto \mathbf{b} is $\frac{\mathbf{a} \cdot \mathbf{b}}{\ \mathbf{b}\ ^2}$ multiplied by vector \mathbf{b}. But wait, in this case, they only gave me that the dot product of \mathbf{a} and \mathbf{b} is 8. Hmm, so maybe I don't need the entire vector \mathbf{b}, just the scalar projection?</p>
	<p>Wait, no, the question says "find the projection of \mathbf{a} onto \mathbf{b}," which usually refers to the vector projection. The formula for the vector projection is $(\frac{\mathbf{a} \cdot \mathbf{b}}{\ \mathbf{b}\ ^2}) \mathbf{b}$. But since I don't have $\ \mathbf{b}\$, maybe I can just express it in terms of the given dot product and the magnitude of \mathbf{b}.</p>
	<p>Alternatively, maybe the question is asking for the scalar projection, which is just $\frac{\mathbf{a} \cdot \mathbf{b}}{\ \mathbf{b}\ }$. But they gave me $\mathbf{a} \cdot \mathbf{b} = 8$, but I don't have $\ \mathbf{b}\$. Let me calculate $\ \mathbf{b}\$ first. The magnitude of \mathbf{b} is $\sqrt{2^2 + 6^2 + 3^2} = \sqrt{4 + 36 + 9} = \sqrt{49} = 7$. Oh, that's nice, it's 7. So if I need the vector projection, it's $(8/7)\mathbf{b}$, which is $(8/49)[2, 6, 3]$. That would give me $[16/49, 48/49, 24/49]$. Alternatively, if it's the scalar projection, it's $8/7$. But the question says "projection," not specifying vector or scalar, but since they gave the vector \mathbf{b}, I think they want the vector projection. So the answer should be $[16/49, 48/49, 24/49]$.</p>
	<p></think></p>
	<p>The projection of vector \mathbf{a} onto \mathbf{b} is given by: $\text{proj}_{\mathbf{b}} \mathbf{a} = \left(\frac{\mathbf{a} \cdot \mathbf{b}}{\ \mathbf{b}\ ^2} \right) \mathbf{b}$</p>
Given:	<p>- $\mathbf{a} \cdot \mathbf{b} = 8$ - $\mathbf{b} = \begin{pmatrix} 2 \\ 6 \\ 3 \end{pmatrix}$</p>
	<p>First, calculate $\ \mathbf{b}\$: $\ \mathbf{b}\ = \sqrt{2^2 + 6^2 + 3^2} = \sqrt{4 + 36 + 9} = \sqrt{49} = 7$</p>
	<p>Then, $\ \mathbf{b}\ ^2 = 7^2 = 49$</p>
	<p>Now substitute: $\text{proj}_{\mathbf{b}} \mathbf{a} = \left(\frac{8}{49} \right) \begin{pmatrix} 2 \\ 6 \\ 3 \end{pmatrix} = \begin{pmatrix} \frac{16}{49} \\ \frac{48}{49} \\ \frac{24}{49} \end{pmatrix}$</p>
Answer:	$\begin{pmatrix} \frac{16}{49} \\ \frac{48}{49} \\ \frac{24}{49} \end{pmatrix}$

964
965
966
967
968
969
970
971

972
973
974
975
976
977

Table A4: Examples from DeepSeek-R1-Distill-Qwen-7B on MATH500 under the original decoding strategy and CyclicReflex. Reflection tokens are highlighted in red.

979

980
981

Q3: Solve $\frac{ x^2 - 81 }{x^2 - 36x} < 0$	
982 Label: (0, 9) \cup (9, 36) 983 Original: <think> Okay, so I have this inequality to solve: $\frac{ x^2 - 81 }{x^2 - 36x} < 0$. Hmm, let me try to figure this out step by step. 984 First, I know that for a fraction to be less than zero, the numerator and denominator must have opposite signs. That means either 985 the numerator is positive and the denominator is negative, or the numerator is negative and the denominator is positive. But wait , 986 the numerator here is an absolute value, right? So $ x^2 - 81 $ is always non-negative, meaning it's either positive or zero. That 987 means the numerator can't be negative. So the only possibility is that the numerator is positive and the denominator is negative. 988 Wait, but the numerator is $ x^2 - 81 $, which is zero when $x^2 = 81$, so $x = 9$ or $x = -9$. At those points, the numerator is zero, 989 making the whole fraction zero, which doesn't satisfy the inequality because it's less than zero, not equal to zero. So we can ignore 990 those points for now. 991 So, focusing on when the denominator is negative. The denominator is $x^2 - 36x$, which I can factor as $x(x - 36)$. So, the 992 denominator is negative when $x(x - 36) < 0$. Let's solve that inequality. 993 To find where $x(x - 36) < 0$, I can consider the critical points where the expression equals zero, which are $x = 0$ and $x = 36$. 994 These points divide the number line into intervals. I'll test each interval: 995 1. When $x < 0$, say $x = -1$: $(-1)(-1 - 36) = (-1)(-37) = 37$, which is positive. 2. When $0 < x < 36$, say $x = 18$: 996 $18(18 - 36) = 18(-18) = -324$, which is negative. 3. When $x > 36$, say $x = 37$: $37(37 - 36) = 37(1) = 37$, which is positive. 997 So, the denominator is negative only when $0 < x < 36$. 998 But wait , I also need to make sure that the denominator isn't zero because division by zero is undefined. So x can't be 0 or 36. 999 Now, going back to the original inequality, since the numerator is always non-negative, the fraction will be negative only when 1000 the denominator is negative. So, the solution should be where the denominator is negative, excluding the points where it's zero. 1001 Therefore, the solution is $0 < x < 36$, excluding $x = 0$ and $x = 36$. But wait , $x = 0$ is already excluded because the interval 1002 starts at 0, but not including 0. Similarly, $x = 36$ is not included because the interval ends at 36, not including it. 1003 Wait, but I should also check if there are any other restrictions. For example, when $x = 9$, the numerator becomes zero, making 1004 the fraction zero, which doesn't satisfy the inequality. Similarly, when $x = -9$, but that's outside the interval we found, so it's not 1005 part of the solution. 1006 So, putting it all together, the solution to the inequality is all real numbers x such that $0 < x < 36$, excluding $x = 0$ and $x = 36$, 1007 but since those points are already excluded from the interval, the solution is simply $0 < x < 36$. 1008 </think> 1009 The solution to the inequality $\frac{ x^2 - 81 }{x^2 - 36x} < 0$ is x in the interval (0, 36). 1010 Answer: (0, 36)	982 Label: (0, 9) \cup (9, 36) 983 CyclicReflex: <think> Okay, so I have this inequality to solve: $\frac{ x^2 - 81 }{x^2 - 36x} < 0$. Hmm, let me try to figure this out step by step. 984 First, I know that for a fraction to be less than zero, the numerator and denominator must have opposite signs. That means either 985 the numerator is positive and the denominator is negative, or the numerator is negative and the denominator is positive. But wait , 986 the numerator here is an absolute value, right? So $ x^2 - 81 $ is always non-negative, meaning it's either positive or zero. That 987 means the numerator can't be negative. So the only possibility is that the numerator is positive and the denominator is negative. 988 Wait, but the numerator is $ x^2 - 81 $, which is zero when $x^2 = 81$, so $x = 9$ or $x = -9$. At those points, the numerator is zero, 989 making the whole fraction zero, which doesn't satisfy the inequality because it's less than zero, not equal to zero. So we can ignore 990 those points for now. 991 So, focusing on when the denominator is negative. The denominator is $x^2 - 36x$, which I can factor as $x(x - 36)$. So, the 992 denominator is negative when $x(x - 36) < 0$. Let's solve that inequality. 993 To find where $x(x - 36) < 0$, I can consider the critical points where the expression equals zero, which are $x = 0$ and $x = 36$. 994 These points divide the number line into intervals. I'll test each interval: 995 1. When $x < 0$, say $x = -1$: $(-1)(-1 - 36) = (-1)(-37) = 37$, which is positive. 2. When $0 < x < 36$, say $x = 18$: 996 $18(18 - 36) = 18(-18) = -324$, which is negative. 3. When $x > 36$, say $x = 37$: $37(37 - 36) = 37(1) = 37$, which is positive. 997 So, the denominator is negative only when $0 < x < 36$. 998 Now, I also need to make sure that the denominator isn't zero, so $x \neq 0$ and $x \neq 36$. So, the solution so far is $0 < x < 36$. But 999 wait , I should also consider the numerator $ x^2 - 81 $. Since it's an absolute value, it's always non-negative, so it's zero only at 1000 $x = 9$ and $x = -9$. But in the interval $0 < x < 36$, $x = 9$ is within this interval, so at $x = 9$, the numerator is zero, making the 1001 entire fraction zero, which doesn't satisfy the inequality. So, I need to exclude $x = 9$ from the solution. 1002 Wait, but let me double-check. If $x = 9$, then the numerator is zero, so the fraction is zero, which is not less than zero. So, $x = 9$ 1003 should not be included. But in the interval $0 < x < 36$, excluding $x = 9$, the fraction is negative everywhere else because the 1004 numerator is positive and the denominator is negative. 1005 Wait, but hold on, when x is between 0 and 9, let's pick $x = 1$, then $ 1^2 - 81 = 80$, and the denominator $1^2 - 36 \cdot 1 = 1 - 36 = -35$, 1006 so $80/(-35)$ is negative. Similarly, between 9 and 36, say $x = 18$, $ 18^2 - 81 = 324 - 81 = 243$, and denominator 1007 $18^2 - 36 \cdot 18 = 324 - 648 = -324$, so $243/(-324)$ is negative. So, in both sub-intervals, the fraction is negative. 1008 So, the solution is $0 < x < 36$, excluding $x = 9$. But wait , at $x = 9$, the numerator is zero, so the fraction is zero, which is not 1009 less than zero. So, we exclude $x = 9$. Therefore, the solution is $0 < x < 9$ or $9 < x < 36$. 1010 Let me just make sure there are no other points where the denominator is zero or the numerator is zero. The denominator is zero at 1011 $x = 0$ and $x = 36$, which we've already excluded. The numerator is zero at $x = 9$ and $x = -9$. $x = -9$ is not in the interval 1012 $0 < x < 36$, so we don't have to worry about that in this case. 1013 So, putting it all together, the solution to the inequality is x in the interval (0, 9) \cup (9, 36). 1014 </think> 1015 The solution set is x in the interval (0, 9) \cup (9, 36). 1016 Answer: (0, 9) \cup (9, 36)

1021

1022

1023

1024

1025

1026

1027

1028

1029

1030

1031

1032

Table A5: Examples from DeepSeek-R1-Distill-Qwen-7B on MATH500 under the original decoding strategy and CyclicReflex. Reflection tokens are highlighted in red.

1035

1036

1037

1038

1039

1040

1041

1042

1043

1044

1045

1046

1047

1048

1049

1050

1051

1052

1053

1054

1055

1056

1057

1058

1059

1060

1061

1062

1063

1064

1065

1066

1067

1068

1069

1070

1071

1072

1073

1074

1075

1076

1077

1078

1079

Q4: Find $\sin 20^\circ \sin 40^\circ \sin 60^\circ \sin 80^\circ \sin 100^\circ \sin 120^\circ \sin 140^\circ \sin 160^\circ$.

Label:	$\frac{9}{256}$
Original:	<p><think> Okay, so I have this problem where I need to find the product of several sine terms: $\sin 20^\circ, \sin 40^\circ, \sin 60^\circ, \sin 80^\circ, \sin 100^\circ, \sin 120^\circ, \sin 140^\circ, \sin 160^\circ$. Hmm, that's a lot of sines multiplied together. I remember that sometimes products of sines can be simplified using trigonometric identities, but I'm not exactly sure which ones apply here. Let me think. First, I notice that some of these angles are related. For example, $\sin 100^\circ$ is the same as $\sin(180^\circ - 80^\circ)$, which is $\sin 80^\circ$. Similarly, $\sin 120^\circ$ is $\sin(180^\circ - 60^\circ)$, so that's $\sin 60^\circ$. Wait, but $\sin 60^\circ$ is already in the list, so maybe that's not helpful. Let me list all the angles: $20^\circ, 40^\circ, 60^\circ, 80^\circ, 100^\circ, 120^\circ, 140^\circ, 160^\circ$. I can pair some of these. For instance, $\sin 100^\circ$ is $\sin 80^\circ$, as I thought, and $\sin 160^\circ$ is $\sin 20^\circ$. Similarly, $\sin 140^\circ$ is $\sin 40^\circ$, and $\sin 120^\circ$ is $\sin 60^\circ$. So, actually, each of these angles from 20° up to 80° has a corresponding angle from 100° up to 160° that is supplementary, meaning their sines are equal. So, the product is actually $(\sin 20^\circ \cdot \sin 40^\circ \cdot \sin 60^\circ \cdot \sin 80^\circ)^2$ because each term from 20° to 80° is squared when considering their supplementary counterparts.</p> <p>Wait, no, let me check. The original list has eight terms, but when pairing, each pair gives the same sine value, so it's like $(\sin 20^\circ \cdot \sin 40^\circ \cdot \sin 60^\circ \cdot \sin 80^\circ)^2$. But let me confirm: $\sin 100^\circ = \sin 80^\circ, \sin 120^\circ = \sin 60^\circ, \sin 140^\circ = \sin 40^\circ, \sin 160^\circ = \sin 20^\circ$. So, yes, the product becomes $(\sin 20^\circ \cdot \sin 40^\circ \cdot \sin 60^\circ \cdot \sin 80^\circ)^2$.</p> <p>Now, I can compute this product step by step. First, let's compute $\sin 60^\circ$, which is a known value: $\sqrt{3}/2$. So, that's straightforward. Now, the remaining terms are $\sin 20^\circ, \sin 40^\circ$, and $\sin 80^\circ$. I remember there are product identities for multiple angles, maybe using the sine of multiple angles or product-to-sum formulas. Alternatively, perhaps using complex numbers or roots of unity. Wait, maybe there's a formula for the product of sines of angles in arithmetic progression. I recall that for n terms, the product of $\sin(k\pi/(2n))$ for $k = 1$ to $n-1$ is $\sqrt{n}/2^{n-1}$. But I'm not sure if that applies here. Let me think. Alternatively, I can use the identity for the product of sines of angles that are in a certain pattern. Wait, another approach: using the identity that $\sin \theta = \cos(90^\circ - \theta)$. So, $\sin 80^\circ = \cos 10^\circ, \sin 40^\circ = \cos 50^\circ, \sin 20^\circ = \cos 70^\circ$. Hmm, not sure if that helps directly.</p> <p>Alternatively, perhaps using the multiple-angle formula for sine. Let's consider $\sin 5\theta$ or something. Wait, maybe using the identity for $\sin 5\theta$ in terms of $\sin \theta$. But that might be complicated.</p> <p>Wait, maybe using the identity that the product of sines can be expressed as a fraction involving powers of 2 and products of cosines. Alternatively, perhaps I can use the identity that the product of $\sin \theta$ for $\theta = 20^\circ, 40^\circ, 60^\circ, 80^\circ$ is equal to something specific.</p> <p>Wait, I think I remember that the product $\sin 20^\circ \sin 40^\circ \sin 60^\circ \sin 80^\circ$ is equal to $\sqrt{3}/16$. Let me check that. If that's the case, then the square of that would be $(\sqrt{3}/16)^2 = 3/256$. But I'm not entirely sure. Let me try to compute it step by step.</p> <p>Alternatively, perhaps using the identity that the product of sines can be expressed as a sum, but that might not be straightforward. Wait, another approach: using the identity that the product $\sin A \sin B = [\cos(A - B) - \cos(A + B)]/2$. Maybe I can apply this iteratively.</p> <p>Let me try to compute $\sin 20^\circ \sin 40^\circ$ first. Using the identity, that's $[\cos(20^\circ) - \cos(60^\circ)]/2$. $\cos 20^\circ$ is approximately 0.9397, and $\cos 60^\circ$ is 0.5, so $[0.9397 - 0.5]/2 \approx 0.21985$.</p> <p>Now, multiply this by $\sin 60^\circ$, which is $\sqrt{3}/2 \approx 0.8660$. So, $0.21985 \cdot 0.8660 \approx 0.1902$.</p> <p>Next, multiply by $\sin 80^\circ$, which is approximately 0.9848. So, $0.1902 \cdot 0.9848 \approx 0.1873$.</p> <p>Wait, but I thought the product $\sin 20^\circ \sin 40^\circ \sin 60^\circ \sin 80^\circ$ was $\sqrt{3}/16 \approx 0.10825$. Hmm, my approximate calculation gave me around 0.1873, which is higher. Maybe my approach is wrong.</p> <p>Wait, perhaps I should use exact values. Let me try again.</p> <p>We know that $\sin 60^\circ = \sqrt{3}/2$. Now, let's consider the product $\sin 20^\circ \sin 40^\circ \sin 80^\circ$. I remember that $\sin A \sin(60^\circ - A) \sin(60^\circ + A) = \frac{\sin 3A}{4}$. If $A = 20^\circ$, then $\sin 20^\circ \sin 40^\circ \sin 80^\circ = \frac{\sin 60^\circ}{4} = \frac{\sqrt{3}/2}{4} = \frac{\sqrt{3}}{8}$. Yes, that's correct.</p> <p>So, the product $\sin 20^\circ \sin 40^\circ \sin 60^\circ \sin 80^\circ$ is $(\sin 20^\circ \sin 40^\circ \sin 80^\circ) \sin 60^\circ = \frac{\sqrt{3}}{8} \cdot \frac{\sqrt{3}}{2} = \frac{3}{16}$.</p> <p>Wait, that makes sense. So, the product $\sin 20^\circ \sin 40^\circ \sin 60^\circ \sin 80^\circ$ is $3/16$.</p> <p>Then, since the original problem's product is $(\sin 20^\circ \sin 40^\circ \sin 60^\circ \sin 80^\circ)^2$, that would be $(\frac{3}{16})^2 = \frac{9}{256}$.</p> <p>Wait, but let me make sure. The original product is $\sin 20^\circ \sin 40^\circ \sin 60^\circ \sin 80^\circ \sin 100^\circ \sin 120^\circ \sin 140^\circ \sin 160^\circ$, which we paired as $(\sin 20^\circ \sin 40^\circ \sin 60^\circ \sin 80^\circ)^2$ because each term from 20° to 80° has a supplementary angle equal to another term in the list. So, the product is indeed $(\sin 20^\circ \sin 40^\circ \sin 60^\circ \sin 80^\circ)^2 = (3/16)^2 = 9/256$.</p> <p>Wait, but when I calculated the approximate product, I got around 0.1873, which is approximately $3/16 \approx 0.1875$. So that matches. Therefore, the square would be $(3/16)^2 = 9/256$. So, the final answer should be $9/256$.</p> <p></think></p> <p>The product simplifies by recognizing that each sine term from 20° to 80° pairs with its supplementary angle, resulting in the square of the product of these terms. Using known identities, the product of $\sin 20^\circ, \sin 40^\circ, \sin 60^\circ$, and $\sin 80^\circ$ is $\sqrt{3}/8$. Squaring this gives $(\sqrt{3}/8)^2 = 3/16$. However, upon re-evaluating, the correct product of all eight terms is actually $(\sqrt{3}/8)^2 = 3/256$. However, upon re-evaluating, the correct product of all eight terms is actually $(\sqrt{3}/16)^2 = 3/256$.</p>
Answer:	$\frac{3}{256}$

1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091

1092 **CyclicReflex:** <think> Okay, so I have this problem where I need to find the product of several sine terms: $\sin 20^\circ, \sin 40^\circ, \sin 60^\circ, \sin 80^\circ, \sin 100^\circ, \sin 120^\circ, \sin 140^\circ$, and $\sin 160^\circ$. Hmm, that's a lot of sines multiplied together. I remember that sometimes products of sines can be simplified using trigonometric identities, **but** I'm not exactly sure which ones apply here. Let me think. First, I notice that some of these angles are related. For example, $\sin 100^\circ$ is the same as $\sin(180^\circ - 80^\circ)$, which is $\sin 80^\circ$. Similarly, $\sin 120^\circ$ is $\sin(180^\circ - 60^\circ)$, so that's $\sin 60^\circ$. **Wait**, **but** $\sin 60^\circ$ is already in the list, so maybe that's not helpful. Let me list all the angles: $20^\circ, 40^\circ, 60^\circ, 80^\circ, 100^\circ, 120^\circ, 140^\circ, 160^\circ$. I can pair some of these. For instance, $\sin 100^\circ$ is $\sin(80^\circ)$, as I thought, and $\sin 160^\circ$ is $\sin(20^\circ)$. Similarly, $\sin 140^\circ$ is $\sin(40^\circ)$, and $\sin 120^\circ$ is $\sin(60^\circ)$. So, actually, each of these angles from 20° up to 80° has a corresponding angle from 100° up to 160° that is supplementary, meaning their sines are equal. So, the product is actually $[\sin 20^\circ \cdot \sin 40^\circ \cdot \sin 60^\circ \cdot \sin 80^\circ]^2$ because each term from 20° to 80° is squared when considering their supplementary counterparts. So, the original product simplifies to $[\sin 20^\circ \cdot \sin 40^\circ \cdot \sin 60^\circ \cdot \sin 80^\circ]^2$. Now, I need to compute this square. Maybe I can compute the product inside the square first and then square the result. I remember that $\sin 60^\circ$ is $\frac{\sqrt{3}}{2}$, which is approximately 0.866, **but** I'm not sure if that helps directly. Let me think about other identities. There's a product formula for multiple sine terms, **but** I can't recall it exactly. **Alternatively**, I might use complex numbers or roots of unity, **but** that might be too advanced for now. **Wait**, I think there's a formula for the product of sines of equally spaced angles. Let me try to recall. For example, the product $\sin \theta \cdot \sin(\theta + \phi) \cdots$ where ϕ is a common difference. **But** I'm not sure if that applies here. **Alternatively**, I can use the identity that relates the product of sines to a sum, **but** I'm not sure how to apply that here. Maybe another approach: using the identity $\sin A \sin B = \frac{\cos(A-B) - \cos(A+B)}{2}$. **But** with four terms, that might get complicated. **Wait**, perhaps I can pair the terms in a way that makes it easier. Let me pair $\sin 20^\circ$ with $\sin 80^\circ$, and $\sin 40^\circ$ with $\sin 60^\circ$, leaving $\sin 100^\circ$ as is. So, $[\sin 20^\circ \sin 80^\circ] \cdot [\sin 40^\circ \sin 60^\circ] \cdot \sin 100^\circ$. Hmm, **but** I'm not sure if that helps. **Wait**, I think $\sin A \sin(60^\circ - A) \sin(60^\circ + A) = \frac{\sin 3A}{4}$. Is that correct? Let me test with $A = 20^\circ$. Then, $\sin 20^\circ \sin 40^\circ \sin 80^\circ = \sin 20^\circ \sin(60^\circ - 20^\circ) \sin(60^\circ + 20^\circ) = \frac{\sin 60^\circ}{4}$, according to that identity. Since $\sin 60^\circ = \frac{\sqrt{3}}{2}$, then $\sin 20^\circ \sin 40^\circ \sin 80^\circ = \frac{\sqrt{3}}{8}$. **Wait**, **but** I'm trying to compute $\sin 20^\circ \sin 40^\circ \sin 60^\circ \sin 80^\circ$, which includes $\sin 60^\circ$ as well. So, if $\sin 20^\circ \sin 40^\circ \sin 80^\circ = \frac{\sqrt{3}}{8}$, then multiplying by $\sin 60^\circ = \frac{\sqrt{3}}{2}$ gives $\left(\frac{\sqrt{3}}{8}\right)\left(\frac{\sqrt{3}}{2}\right) = \frac{3}{16}$. So, the product inside the square is $\frac{3}{16}$, and then squaring it gives $(\frac{3}{16})^2 = \frac{9}{256}$. **Wait**, **but** let me double-check that. If $\sin 20^\circ \sin 40^\circ \sin 80^\circ = \frac{\sqrt{3}}{8}$, then multiplying by $\sin 60^\circ = \frac{\sqrt{3}}{2}$ indeed gives $(\frac{\sqrt{3}}{8})(\frac{\sqrt{3}}{2}) = \frac{3}{16}$. So, yes, the product inside the square is $\frac{3}{16}$, and squaring it gives $\frac{9}{256}$. So, the final answer should be $\frac{9}{256}$. **Wait**, **but** I'm a bit confused because I thought the product of $\sin 20^\circ \sin 40^\circ \sin 60^\circ \sin 80^\circ$ would involve more steps. Maybe I should verify this identity again. Let me consider the identity: $\sin A \sin(60^\circ - A) \sin(60^\circ + A) = \frac{\sin 3A}{4}$. So, if I set $A = 20^\circ$, then $\sin 20^\circ \sin 40^\circ \sin 80^\circ = \frac{\sin 60^\circ}{4} = \frac{\frac{\sqrt{3}}{2}}{4} = \frac{\sqrt{3}}{8}$. Then, including $\sin 60^\circ = \frac{\sqrt{3}}{2}$, the total product becomes $\left(\frac{\sqrt{3}}{8}\right)\left(\frac{\sqrt{3}}{2}\right) = \frac{3}{16}$. So, that seems correct. **Alternatively**, maybe I can use another approach to confirm. Let's compute the product step by step. Starting with $\sin 20^\circ \approx 0.3420$, $\sin 40^\circ \approx 0.6428$, $\sin 60^\circ \approx 0.8660$, $\sin 80^\circ \approx 0.9848$, $\sin 100^\circ = \sin 80^\circ \approx 0.9848$, $\sin 120^\circ = \sin 60^\circ \approx 0.8660$, $\sin 140^\circ = \sin 40^\circ \approx 0.6428$, $\sin 160^\circ = \sin 20^\circ \approx 0.3420$. Multiplying all these together yields approximately 0.03515625, which matches $\frac{9}{256}$. Therefore, I'm confident that the product is $\frac{9}{256}$. </think> The product simplifies using trigonometric identities and properties of supplementary angles. By recognizing pairs of sines and applying the identity for the product of sines of angles in arithmetic progression, the result is found to be 9/256.

1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133

Answer: $\frac{9}{256}$

1134 **D LLM USAGE**
11351136 GPT-5 was employed to provide writing assistance, particularly in polishing and refining the language,
1137 during the preparation of this manuscript.
11381139 **E LIMITATIONS**
11401141 While CyclicReflex effectively mitigates under-reflection and over-reflection by cyclically modulating
1142 reflection token logits, its theoretical foundations remain underexplored. The proposed analogy
1143 between reflection token allocation in reasoning and learning rate scheduling in optimization offers a
1144 promising direction for future theoretical investigation. A key open question is why LRM^s exhibit
1145 under- or over-reflection during generation. Understanding the underlying causes of these behaviors is
1146 essential for developing a principled understanding of CyclicReflex. As such, future work should aim
1147 to formalize the generative dynamics of reflection in LRM^s and establish a deeper theoretical basis
1148 for the design and improvement of reasoning resource allocation strategies, including CyclicReflex.
11491150 **F BROADER IMPACTS**
11511152 On the positive side, our work demonstrates that scheduling reflection token appearance can effectively
1153 address under-reflection and over-reflection, leading to more accurate and effective responses from
1154 LRM^s. This contributes to enhanced reasoning capabilities and improved performance on complex
1155 problem-solving tasks. Moreover, by drawing a connection between learning rate schedules in
1156 optimization and reflection token dynamics in reasoning, our work opens new research directions and
1157 may inspire more interpretable and controllable LRM designs.
11581159 On the negative side, CyclicReflex could potentially be misused to manipulate reasoning traces. For
1160 example, an adversary could deliberately modulate reflection token usage to craft outputs that embed
1161 sensitive or hallucinated content in a more convincing manner, potentially evading safety filters. To
1162 mitigate such risks, it is crucial that advanced decoding strategies, such as CyclicReflex, are deployed
1163 within robust ethical and safety frameworks, especially in the context of unlearning and high-stakes
1164 applications. We hope this research contributes to the development of LRM^s that are not only efficient
1165 and capable but also safe, trustworthy, and aligned with human values.
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187