

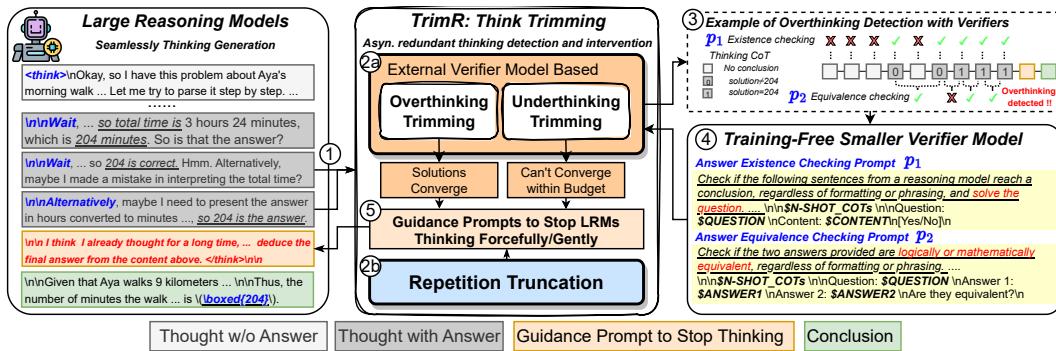
000 TRIMR: VERIFIER-BASED TRAINING-FREE THINKING 001 TRIMMING FOR EFFICIENT TEST-TIME SCALING

002 **Anonymous authors**

003 Paper under double-blind review

004 ABSTRACT

005 Large Reasoning Models (LRMs) demonstrate exceptional capability in tackling
006 complex mathematical and logical tasks by leveraging extended Chain-of-
007 Thought (CoT) reasoning. Test-time scaling methods—such as prolonging CoT
008 with explicit token-level exploration—can push LRM’s accuracy boundaries, but
009 they incur significant decoding overhead. A key inefficiency source is LRM’s often
010 generate redundant thinking CoTs, which demonstrate clear structured over-
011 thinking and underthinking patterns. Inspired by human cognitive reasoning pro-
012 cesses and numerical optimization theories, we propose **TrimR**, a **Verifier-based**,
013 **training-free**, **efficient framework to trim reasoning and enhance test-time**
014 **scaling**, explicitly tailored for production-level deployment. Our method employs
015 a lightweight, pretrained, instruction-tuned verifier to detect and truncate redun-
016 dant intermediate thoughts of LRM without any LRM or verifier fine-tuning.
017 We present both the core algorithm and asynchronous online system engineered
018 for high-throughput industrial applications. Empirical evaluations on **Ascend**
019 **NPUs and vLLM** show that our framework delivers substantial gains in infer-
020 ence efficiency under large-batch workloads. In particular, on the four MATH500,
021 AIME24/25, and GPQA benchmarks, **the reasoning runtime of QwQ-32B, DeepSeek-R1-Distill-Qwen-32B, and Pangu-R-38B is improved by up to 70% with negligible impact on accuracy.**



022 **Figure 1:** The TrimR framework. (1) CoTs are divided into sub-thoughts with reflection tokens in Sec. 3.1; (2a)
023 Over/underthinking detection through answer convergence analysis in Sec. 3.2, 3.3 (2b) repetition truncation in
024 Sec. 3.4; (3) By simplifying redundant detection as the binary answer existence and equivalence classification
025 with prompts p_1 and p_2 in Appendix F, 7B instruction models replace PRMs/ORMs, avoiding instability from
026 full-sequence scoring; (4) The verifier applies p_1 to assess conclusion completeness in individual thoughts and
027 p_2 to identify overthinking when consecutive thoughts yield identical answers. Early termination is triggered
028 with Algorithm 1, 2; (5) Thinking termination prompts for LRM are generated based on verifier decisions to
029 halt redundant reasoning gently or forcefully in Sec. 3.5.

030 1 INTRODUCTION

031 Large Reasoning Models (LRMs) such as OpenAI o1 (OpenAI, 2024), DeepSeek R1 (Guo et al.,
032 2025), and Qwen QwQ (Team, 2025) achieve expert-level performance on *mathematical and sci-
033 entific reasoning tasks* by decomposing problems into manageable subproblems, conducting step-
034 by-step analysis, detecting and correcting errors, and exploring alternative solutions. However, this
035 extended “thinking” incurs substantial decoding overhead and inference cost, hindering deployment
036 in resource-constrained settings (Guo et al., 2025).

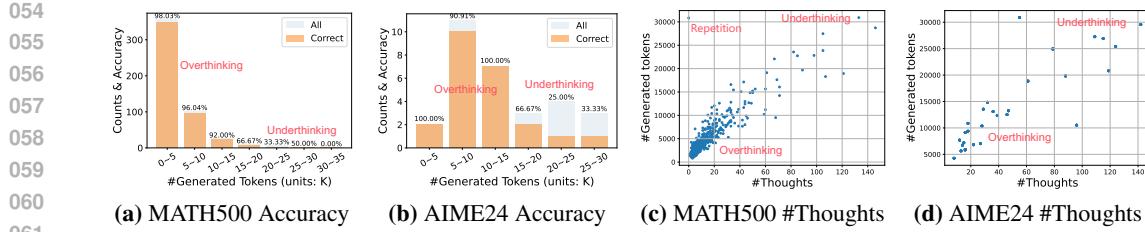


Figure 2: Correlation between occurrence, accuracy, and the number of generated thoughts/tokens of QwQ-32B on MATH500 (Hendrycks et al., 2021) and AIME24 (AIME2024). For hard questions, QwQ can spin out up to 140 lengthy yet incorrect thoughts—indicating a need for underthinking trimming. For easier questions, QwQ delivers brief, highly accurate answers—yet there’s still room to make them even more concise by overthinking trimming.

Benchmarking on AIME24 (AIME2024) and MATH500 (Hendrycks et al., 2021) reveals two key inefficiencies: **overthinking** and **underthinking** (Marjanović et al., 2025). **Overthinking** manifests as redundant verification of already-correct steps—often signaled by reflection tokens such as “Wait” or “Alternatively”—which increases output length without accuracy gains (as in Fig. 1, 2). **Underthinking** occurs on harder problems when the model oscillates among incomplete reasoning chains and fails to converge, producing lengthy yet inaccurate responses (Fig. 2). Representative examples are provided in Appendix G.

The extended Chain-of-Thought (CoT) reasoning in LMRs poses deployment challenges: decoding lengths vary widely and runtime scales superlinearly with sequence length. Training-based methods such as (Yan et al., 2025; Chen et al., 2024; Munkhbat et al., 2025; Yu et al., 2024) reduce token generation in LMRs but impose heavy training and computational costs on large models and may degrade their general capabilities. In contrast, training-free techniques integrate seamlessly and preserve original behavior: TokenBudget (Han et al., 2024) dynamically adjusts token budgets via prompting, and Chain of Draft (Xu et al., 2025) uses concise instructions to shorten output, yet both require invasive inference-time modifications.

We thus propose **TrimR**, a **verifier-based, training-free approach for online thinking trimming** while maintaining reasoning performance. We detect overthinking and underthinking in **intermediate thought answers** with a **lightweight verifier and trigger prompt-based LRM thinking early termination upon redundancy detection**. By simply checking answer existence and equivalence in brief thought segments, our method leverages compact verifier models instead of complex Process Reward Models (PRMs) or Outcome Reward Models (ORMs) (Lightman et al., 2024). The non-invasive early termination preserves original LRM capabilities. Finally, we present an online asynchronous system collaborating LMRs and small verifiers to support industrial-scale workloads. **TrimR offers a significant advantage by maintaining the performance and knowledge of LMRs** through a targeted intervention strategy that is activated only when redundant reasoning is detected.

Notations. Given an input X , the LRM Π generates a response $Y = \Pi(X)$. We denote $\mathbf{y}_{<t} = [y_1, y_2, \dots, y_{t-1}]$ as the previously generated tokens. Each time $y_t = p_\Pi(X, \mathbf{y}_{<t})$. With a slight abuse of notation, let $\Pi(X, \mathbf{y}_{<t})$ denote full response including $\mathbf{y}_{<t}$.

Theoretical Foundation. Our method unifies human cognitive heuristics—overthinking and underthinking as confidence-threshold and diminishing-returns processes—with mathematical optimization. People typically stop thinking further after finding answers to simple questions and give up on complex tasks after too many unsuccessful attempts. We model reasoning process as an optimization problem in “language space”, where LMRs traverse token trajectories and converge to an optimal solution. Mirroring numerical optimizers’ early-stopping, we introduce a termination criterion that halts reasoning once reasoning converges or marginal gains fall below a preset threshold. Formally, given an input X with a partial response $\mathbf{y}_{<t}$, the reasoning performance of LRM Π is denoted as $Perf(X, \mathbf{y}_{<t} | \Pi)$. We derive a compression rule \mathbf{c} to determine the stopping time $t' \in \{0, \dots, t-1\}$. The goal is to minimize the inference cost of $\mathbf{y}_{<t'}$ referred to as $Infer_Cost(\mathbf{y}_{<t'})$ without degrading reasoning performance.

$$\min_{\mathbf{c}(\cdot)} \quad Infer_Cost(\mathbf{y}_{<t'}) \quad \text{s.t.} \quad t' = \mathbf{c}(\mathbf{y}_{<t})$$

$$Perf(X, \mathbf{y}_{<t'} | \Pi) \geq Perf(X, \mathbf{y}_{<t} | \Pi)$$

108 **Contributions.**

109

- 110 We propose a lightweight, training-free method for dynamic thinking trimming in
111 LRM斯—mitigating both over- and underthinking—via a small verifier, enabling efficient
112 test-time scaling with negligible loss of reasoning accuracy.
- 113 We develop an asynchronous thinking trimming system for industrial-scale deployment,
114 seamlessly integrating with existing inference infrastructures.
- 115 Through extensive evaluations and ablations on diverse models (QwQ-32B, Deepseek-R1-
116 Distill-Qwen-32B, and Pangu-R-38B), we demonstrate consistent reductions in reasoning
117 cost and token usage across standard reasoning benchmarks in production settings.

118 **2 RELATED WORKS**

119 This section reviews recent advancements in efficient reasoning techniques for LRM斯. For a more
120 detailed discussion, see comprehensive surveys (Sui et al., 2025; Qu et al., 2025; Su et al., 2025).
121 The slight difference of the fast and slow thinking models motivates efficient long-to-short reasoning
122 with model merging, which directly fuse weights of models with different thinking patterns using
123 carefully designed algorithms (Team et al., 2025; Wu et al., 2025a). Apart from it, there are three
124 types of commonly used efficient reasoning methods.

125 **Training-based Methods.** Approaches in this category modify or fine-tune LRM斯 to generate con-
126 cise reasoning. Reinforcement learning with length-based rewards prunes verbose chains (Hou et al.,
127 2025; Xia et al., 2025; Zhang et al., 2025a; Team et al., 2025; Chen et al., 2024; Arora & Zanette,
128 2025; Chen et al., 2025; Fang et al., 2025), while supervised fine-tuning teaches models to compress
129 explanations (Munkhbat et al., 2025; Yu et al., 2024). Latent-space techniques further minimize
130 token usage by operating in a compact semantic embedding space (Hao et al., 2024). Although ef-
131 fective, these methods demand substantial compute, risk task-specific overfitting, and may degrade
132 general-purpose capabilities.

133 **Self-Evaluation Methods.** These techniques prompt LRM斯 to assess their own confidence and
134 decide when to stop reasoning. Adaptive schemes ask the model to predict the benefit of restart-
135 ing (Manvi et al., 2024) or to estimate certainty at key junctures (Yang et al., 2025). SelfThink (Zeyu
136 et al., 2025) is proposed to facilitate LRM斯’ intrinsic task complexity classification capabilities to
137 dynamically switch between fast and slow thinking. ST-BoN leverages embedding-distance metrics
138 for early truncation (Wang et al., 2025b). While they avoid external models, the added inference
139 steps may introduce latency. Some contemporaneous works intervene in inference to reduce token
140 usage in reasoning: SpeedAdapt (Lin et al., 2025) adjusts hidden states dynamically to control the
141 reasoning speed; NoThink (Ma et al., 2025) suppresses reasoning entirely; AlphaOne (Zhang et al.,
142 2025b) modulates slow and fast thinking at test time based on average thinking length.

143 **Model Collaboration.** Hybrid frameworks use auxiliary evaluation models, reward models, or
144 thought proposers to guide decoding. Dynasor monitors semantic entropy and reward-model outputs
145 for early stopping (Fu et al., 2024; Kuhn et al., 2023). Speculative Rejection uses partial-output
146 scores from a reward model to terminate best-of- N search (Sun et al., 2024). However, above works
147 heavily rely on the performance of the reward model. Another work (Xi et al., 2024) trains a critique
148 model which provides step-level feedback to guide the reasoning model on the fly. There are also
149 concurrent works on improve LRM efficiency by generating thoughts with smaller reasoning models
150 for speculative reasoning (Pan et al., 2025; Wang et al., 2025a). CoThink (Fan et al., 2025) leverages
151 an instruction model to guide reasoning and reduce reasoning steps.

152 Unlike prior work, our TrimR framework is training-free and non-invasive: a lightweight verifier
153 dynamically detects and helps truncate redundant reasoning. No extra self-evaluation steps are in-
154 troduced, so our method can be easily integrated into existing inference frameworks.

155 **3 METHOD**

156 We propose an **efficient, verifier-based, training-free thinking trimming** algorithm that dynam-
157 ically prunes redundant CoT generation during online inference. Our algorithm replicates human
158 cognitive mechanisms which utilize internal verifiers to check and stop thinking. We introduce
159 smaller verifiers to detect redundant thinking without fine-tuning verifiers or LRM斯. Designed for
160 industrial-scale batch processing, our framework (Fig. 1) comprises three modules: **overthinking**

162 **trimming** (Sec. 3.2), **underthinking trimming** (Sec. 3.3), and **repetition truncation** (Sec. 3.4).
 163 Guidance prompts for halting reasoning are detailed in Sec. 3.5. The online system is presented in
 164 Sec. 3.6.
 165

166 3.1 THOUGHT DETECTION AND ANSWER EXTRACTION

168 The reasoning thoughts have clear structured patterns and are usually separated with **reflection to-**
 169 **kens**, such as “\n\nWait”, “\n\nBut”, and “\n\nAlternatively”. In addition, LRM normally gen-
 170 erate answers at the end of thoughts and then verify them as in Fig. 1.

171 During thinking, we periodically segment the thinking process into sub-thoughts $[r_1, r_2, \dots, r_k]$:

$$173 \text{Think_Seg}(\mathbf{y}_{<t}) = [r_1, r_2, \dots, r_k].$$

174 Here, we separate it by reflection tokens (full list in Appendix E). When new reflection tokens
 175 are detected asynchronously, we split the whole thought between two consecutive reflection tokens
 176 into sentences and concatenate the last several sentences of the current thought as the extracted
 177 **intermediate thought answers**. Formally, The last N_{sent} sentences of each sub-thought r_l are
 178 denoted as s_l , i.e. $s_l = \text{Last_sentences}(r_l, N_{sent})$. The last several sentences are shorter but more
 179 informative than the whole sub-thought and normally contain answers as demonstrated in Fig. 1.
 180 We do not further process the answer but only skip the extremely short thoughts, which normally do
 181 not contain useful thoughts.

182 We detect predefined reflection tokens asynchronously so that LRM continue generating tokens
 183 without throughput loss. Since reflection tokens account for only 0.5% of all outputs and thoughts
 184 are significantly less than tokens as in Fig. 2, the verifier is invoked infrequently and can serve
 185 multiple LRMs concurrently as analyzed in Appendix M. It makes the overhead of redundancy
 186 checks negligible compared to the gains from thinking compression.

187 3.2 OVERTHINKING TRIMMING

189 In overthinking scenarios, LRMs typically ar-
 190 rive at correct solutions using only 30 ~ 50%
 191 of the total generated tokens, yet continue pro-
 192 ducing redundant reasoning paths or alterna-
 193 tive justifications before finalizing an answer (Chen
 194 et al., 2024). While this may aid complex or
 195 uncertain tasks, it burdens simpler ones with
 196 uninformative content and higher latency and
 197 inference costs without accuracy gains. We
 198 propose an overthinking detection and com-
 199 pression algorithm that uses lightweight veri-
 200 fier models to prune redundant reasoning while
 201 preserving LRM accuracy. It emulates veri-
 202 fier confidence based human thinking termina-
 203 tion and convergence-based early termination
 204 mechanisms of numerical optimizers.

205 **Simple Tasks to Utilize Smaller Models.** We
 206 simplify overthinking detection as answer ex-
 207 isting and equivalence checking, which are sim-
 208 pler binary classification problems than scor-
 209 ing the whole sequences. By reducing detec-
 210 tion to checking answer existence and com-
 211 paring final outputs of consecutive interme-
 212 diate thought answers, we can deploy compact
 213 (7B) verifiers with satisfactory language com-
 214 prehension and instruction following capabili-
 215 ties without fine-tuning, greatly lowering in-
 216 ference overhead compared to full-sequence re-
 217 ward models (Lightman et al., 2024; Liu et al., 2025). Moreover, training PRMs/ORMs for accurate

Algorithm 1: Overthinking Compression

```

Input: Input X, repeat-threshold M
Output: Generated output Y
stopped ← False; yt ← pΠ(X);
prev.concluded.thought ← None; count ← 0;
while yt ≠ eosi do // LRM iteratively generates
  yt
    // Periodically segment and check for a
    // new concluded thought
    // Fv(p1(sl)) defined in Eqn. 1, r(si, sj)
    // defined in Eqn. 2.
    if yt = </think> then
      | stopped = True, break;
    end
    if a new segment sl is found and Fv(p1(sl)) = 1 and
      stopped = False then
      if prev.concluded.thought ≠ None then
        | if r(prev.concluded.thought, sl) = 1
        | | then
        | | | count ← count + 1;
        | | | else
        | | | | // reset on any mismatch
        | | | | count ← 0;
        | | end
      end
      if count ≥ M then
        // early-stop after M repeats
        yt ← pΠ((X, y<t, stop_tokens));
        stopped ← True, break;
      end
      prev.concluded.thought ← sl;
    else
      | yt ← pΠ((X, y<t, yt));
    end
  end
return (y<t, yt)

```

216 scoring is complex and often yields unstable rewards on identical sequences, undermining reliable
 217 overthinking detection (Sun et al., 2024).
 218

219 **Two-Stage Verification.** To minimize intervention from answerless intermediate checks, we in-
 220 troduce two-stage verification. First, we confirm the candidate solution is present in the current
 221 reasoning thought. We then verify that intermediate thought answers are semantically or logically
 222 equivalent, regardless of correctness. Both tasks can be handled by the lightweight verifier $\mathcal{F}_v: x$

223 (1) *Answer existence checking:* The set of thoughts with solutions S^* is defined as:

$$224 \quad S^* = \{s_i \mid \mathcal{F}_v(p_1(s_i))\}; \mathcal{F}_v(p_1(s_i)) = \mathbb{I}\left[p_{\mathcal{F}_v}(y = \text{"Yes"} \mid p_1(s_i)) > p_{\mathcal{F}_v}(y = \text{"No"} \mid p_1(s_i))\right], \quad (1)$$

225 where $p_1(s_i) = p_1(X, s_i)$ (X omitted for simplicity) is the verifier prompt for answer existence
 226 taking the problem X and thought answer s_i as parameters.
 227

228 (2) *Equivalence checking:* The equivalence r between two consecutive thoughts in S^* is computed
 229 as:

$$231 \quad r(s_i^*, s_{i+1}^*) = \mathcal{F}_v(p_2(s_i^*, s_{i+1}^*)) = \mathbb{I}\left[p_{\mathcal{F}_v}(y = \text{"Yes"} \mid p_2(s_i^*, s_{i+1}^*)) > p_{\mathcal{F}_v}(y = \text{"No"} \mid p_2(s_i^*, s_{i+1}^*))\right], \quad (2)$$

232 where $p_2(s_i, s_j) = p_2(X, s_i, s_j)$ (X omitted for simplicity) is the verifier prompt for answer equiva-
 233 lence, taking the problem X and two consecutive thought answers in S^* that both achieve solutions.
 234

235 **Verifier Prompts.** Verifiers take the answer existence prompt p_1 and answer equivalence prompt
 236 p_2 and directly return the binary classification results with the probabilities of "Yes" and "No"
 237 tokens. Therefore, only prefilling of verifiers is required, which reduces the computational cost.
 238 The two prompts for verifiers (p_1 and p_2) in Fig. 1 with placeholders are available here. Details of
 239 placeholders are in Appendix F. The answer existence checking prompt p_1 consists of system prompt
 240 for ignoring unimportant formats and phrases, n-shot CoTs with positive and negative examples, the
 241 current question, and the intermediate answer. Similarly, the verifier only checks if two consecutive
 242 intermediate thought answers are semantically or logically equivalent with the prompt p_2 .
 243

244 **Verifier Prompt p_1 for Answer Existence:**

245 Check if the following sentences from a rea-
 246 soning model reach an answer, regardless of
 247 formatting or phrasing, and solve the question.
 248 Return 'Yes' if the content finds a solution, oth-
 249 erwise 'No'. Return only 'Yes' or 'No' with no
 250 explanation.
 $\$N\text{-SHOT_COTs}$ (details in Appendix)

251 Question: $\$QUESTION$

Content: $\$CONTENT \backslash n$ [Yes/No]:

244 **Verifier Prompt p_2 for Answer Equivalence**

245 Check if the two answers provided are logically or mathematically
 246 equivalent, regardless of formatting or phrasing. Return 'Yes' if they
 247 are equal in meaning/value and valid solutions to the question, other-
 248 wise 'No'. Return only 'Yes' or 'No' with no explanation.

249 $\$N\text{-SHOT_COTs}$ (details in Appendix)

250 Question: $\$QUESTION$

251 Answer 1: $\$ANSWER1$

252 Answer 2: $\$ANSWER2$

253 Are they equivalent? [Yes/No]:

254 **Early Termination.** We implement early termination through the prompt mechanism introduced in
 255 Sec. 3.5, specifically when the model consecutively agrees with previous reasoning steps M times.
 256 The process is described in Algorithm 1. We only consider thoughts in S^* to bypass intermediate
 257 thoughts lacking definitive solutions. Such a protocol analogously replicates human cognitive pat-
 258 terns where reasoning halts upon achieving $M + 1$ consecutive consistent solutions, paralleling the
 259 convergence termination criterion in optimization algorithms.

260 **Resource Saving.** By limiting inputs to 200–400 tokens instead of processing full reasoner outputs
 261 (8K–128K tokens) as did in PRMs/ORMs (Sun et al., 2024), we drastically cut memory and com-
 262 pute overhead, boosting batch throughput and reducing verification latency with **shorter verifier
 263 prompts**. Also, we always reuse the KV cache for the system prompt and question with **pre-
 264 fix caching**, then batch answers in triplets—remapping the second answer to a fixed placeholder
 265 ($\$ANSWER1$ in p_2). This lets us retain its cache across examples, so only the cache for the other
 266 placeholder ($\$ANSWER2$, used by the first and third answers) needs updating.

267

3.3 UNDERTHINKING TRIMMING

268 In particularly difficult tasks, LRM_s repeatedly verify intermediate steps—an indicator of uncer-
 269 tainty that leads to divergent, redundant reasoning. We find that when the model oscillates between
 270 different thought paths, it seldom converges: proposed solutions often fail verification, triggering

further exploration that inflates latency (Fig. 6) without boosting accuracy, and excessive diversity can even hinder convergence (Chen et al., 2024).

To mitigate this, we repurpose the verifier’s answer-existence/consistency feedback to detect hesitant overthinking. Concretely, if after $R_{thres}\%$ of the token budget and N_{thres} reasoning rounds, the model has not produced at least three consecutive answer-bearing chains that agree on the same solution, we flag the run as underthinking (Appendix B Algorithm 2). At that point, a guidance prompt instructs the model to halt further reasoning and summarize its existing insights. It mimics human tendency to abandon difficult tasks that exceed its capabilities after multiple failed attempts.

This mechanism relies on two principles: (1) repeated convergence on the same solution indicates that the reasoning process is coherent and sufficiently thorough; and (2) the thresholds R_{thres} and N_{thres} must be calibrated to afford the model adequate opportunity to explore alternative solution paths before termination. Ablation studies on these thresholds are presented in Appendix J.

3.4 REPETITION TRUNCATION

Recurrent token loops in LRM often persist despite probabilistic sampling tweaks (e.g., high temperature), especially when sequence length increases. Robust repetition detection and truncation is therefore essential to prevent wasted computation and improve user experience. Our solution uses a rolling-hash-based detector to identify repeating token-ID subsequences in real time. By updating hashes incrementally for each new token, we avoid recomputing whole-sequence hashes, enabling efficient, on-the-fly repetition checks. This dynamic algorithm is enabled by default unless otherwise specified. The effectiveness of this module is provided in Appendix I.

3.5 GUIDANCE PROMPTS FOR LRMS THINKING COMPRESSION

We devised two **gentle and forceful guidance prompts** (presented in Appendix F) to guide terminating the reasoner’s thought process. The gentle prompt curbs overthinking by steering the model toward concluding its reasoning with `**Final Answer**\n`, whereas the forceful prompt both prevents underthinking and breaks any repetitive loops that occur before the designated “think end” token (e.g. `</think>`) is emitted. In our experiments, the forceful prompt consistently halted endless generation during the reasoning stage.

3.6 ONLINE SYSTEM DESIGN

Our inference system is built on vLLM (Kwon et al., 2023) (v0.6.0) running on Ascend-910B-64GB NPUs, though it can be integrated equally well with other inference frameworks (e.g. SGLang (Zheng et al., 2024)) or hardware platforms (e.g. GPUs). As shown in Fig. 3, it comprises two tightly coupled elements: the inference engine itself, which orchestrates decoding via vLLM, and the Test-Time Thinking Trimming System (T4S), which performs real-time reasoning compression in parallel. T4S is structured around a **Reasoning Verifier**—responsible for detok-

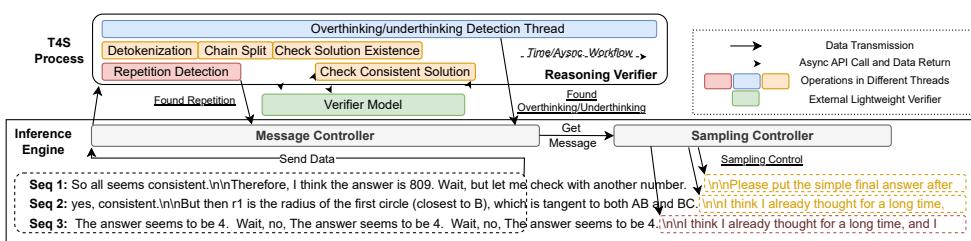


Figure 3: System design for the Test-Time Thinking Trimming System (T4S). The figure shows three sequences that are flagged as (1) overthinking; (2) underthinking; and (3) repetitive generation. The inference engine streams updates through the Message Controller into the external T4S process, which issues asynchronous API calls to a lightweight verifier.

enizing output into discrete reasoning chains, detecting overthinking or underthinking, and using an external small verifier model to confirm solution validity—a **Message Controller** that exchanges operational data (prompt and generated token IDs) with the verifier at configurable intervals, and a **Sampling Controller** that adaptively modifies output logits to enforce specific token generations.

324

4 EXPERIMENTS

325

4.1 EXPERIMENT SETUP

328 **Benchmarks.** This work focuses on mathematical and scientific problems: *Mathematical benchmarks*: **MATH500** (Hendrycks et al., 2021), **AIME24** (AIME2024), **AIME25** (AIME2025) and
 329 *Scientific benchmarks*: **GPQA Diamond** (Rein et al., 2024). Details are available in Appendix H.
 330 We additionally include results on LiveCodeBench (Jain et al., 2024) for dense and MoE models.
 331 As the primary emphasis of this work is on mathematical and scientific reasoning, code tasks are
 332 included for supplementary reference in Appendix O.
 333

334 **Metrics.** Apart from accuracy (**Acc.**), we mainly care about the efficiency metrics. **Runtime** denotes
 335 the total wall-clock time to process all requests in each dataset. **TPR** is the average Time Per
 336 Request, while **TPR-T90** is the TPR of the fastest/top 90% requests. **#Tokens(M)** is the number
 337 of generated tokens in millions. Runtime represents the total waiting time of requests, while lower
 338 TPR and TPR-T90 indicate better single user experience and higher Queries-per-Second (QPS).
 339

340 **Configurations.** All the experiments are conducted on servers with 8 Ascend 910B-64GB NPUs
 341 and 192-cores Kunpeng-920 CPUs with 1.5TB memory. All dataset requests (e.g., the 500 ques-
 342 tions in MATH500)¹ are submitted to vLLM concurrently, and we record key metrics such as total
 343 wall-clock time and per-request latency. We benchmark two open-source models (QwQ-32B and
 344 DeepSeek-R1-Distill-Qwen-32B(R1Q-32B)) alongside the closed-source Pangu-R-38B, using fixed
 345 input/output lengths of 2K/30K tokens. *For readers’ interest, the results on Pangu Pro MoE (Tang*
 346 *et al., 2025), publicly released after the date of this study, are presented in Appendix N.* Although
 347 extending outputs to \sim 128K tokens yields marginal gains, such settings are impractical for produc-
 348 tion, so we cap the output at 30K. $M=2$, $N_{send}=50$. The verifier is Pangu-7B (open-sourced) (Pangu
 349 Team, 2025) (by default) and Qwen2.5-7B-Instruct (Qwen Team, 2024) (Table 4).
 350

351

4.2 MAIN RESULTS

352 Table 1 shows that introducing TrimR delivers consistent and substantial efficiency gains across
 353 all three models and four benchmarks (up to 70% runtime reduction), with accuracy largely unaf-
 354 fected (less than 1.7% drop). Specifically, **runtime** is reduced by 16–39% for QwQ-32B (e.g., from
 355 4,413s to 3,118s on MATH500, –29.3%), 19–39% for Pangu-R-38B (–33.8% on MATH500), and
 356 an impressive 53–70% for R1Q-32B (–67.0% on MATH500; –70.0% on GPQA Diamond). Similar
 357 reductions are seen in **TPR** (e.g., R1Q-32B’s TPR on AIME24 drops from 3,717.6s to 1,433.9s,
 358 –61.4%). **Token usage** also drops by 8–46% overall, with R1Q-32B showing the largest reduction
 359 (from 2.447M to 1.320M tokens, –46.1% on GPQA Diamond). Despite these gains, **accuracy** is
 360 preserved or even improved. QwQ-32B gains on MATH500 (+1.2%) and AIME25 (+0.8%), while
 361 R1Q-32B improves 2.0–13.2% on three benchmarks. *For readers’ interest, we discussed the reasons*
 362 *behind such improvements in Appendix K.* Minor regressions are all under 2%, a reasonable tradeoff
 363 for significant runtime reductions.

364 In Table 2, we include concurrent baselines (Certain-
 365 dex (Fu et al., 2024), CoThink (Fan et al., 2025),
 366 SpeedAdapt (Lin et al., 2025); preprints at the time of
 367 submission, with some released after our initial draft).
 368 we find that TrimR consistently delivers substan-
 369 tial token savings while maintaining accuracy across
 370 datasets. For example, on R1Q-32B with AIME24,
 371 TrimR reduces token usage by 35.6% with stable
 372 accuracy, whereas CoThink achieves only a 12.5%
 373 reduction but suffers a sharp 13.3% accuracy drop.
 374 SpeedAdapt, which adjusts hidden states to control
 375 reasoning speed, maintains accuracy reasonably well
 376 but yields only 4.0–9.3% token savings on QwQ-32B.
 377 Certaindex introduces extra inference cost yet still underperforms TrimR in token reduction. Over-
 378 all, TrimR strikes a more favorable balance between efficiency and reliability relative to the baseline
 379 methods.

380 **Table 2:** Comparing TrimR with baseline meth-
 381 ods. Acc.: accuracy change relative to respective
 382 baseline; #Tok.: token usage reduction.

Model	MATH500		AIME24	
	Acc.	#Tok.	Acc.	#Tok.
QwQ-32B				
CoThink	-3.0%	-19.1%	3.3%	-16.2%
SpeedAdapt	0.0%	-4.0%	1.1%	-9.3%
TrimR (ours)	1.2%	-14.3%	0.0%	-23.3%
DeepSeek-R1-Distill-Qwen-32B				
Certaindex	-4.0%	-19.0%	-4.0%	-15.0%
CoThink	-2.0%	-36.6%	-13.3%	-12.5%
SpeedAdapt	0.7%	-7.3%	1.5%	-12.7%
TrimR (ours)	2.4%	-40.1%	3.3%	-35.6%

383 ¹Since AIME has only 30 questions, we replicate it eightfold to ensure the engine receives enough requests.

378
 379 **Table 1:** Performance comparison of QwQ-32B, Pangu-R-38B, and DeepSeek-R1-Distill-Qwen-32B on the
 380 MATH500, AIME24, AIME25, and GPQA Diamond benchmarks. Relative **improvements** are highlighted in
 381 green, and **regressions** in red.

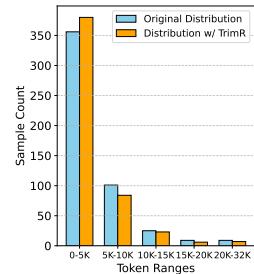
Model	Runtime(s)	TPR(s)	TPR-T90(s)	Acc.	#Tokens(M)
MATH500					
QwQ 32B	4413	593.1	439.7	95.6%	2.278
w/ TrimR	3118 <small>-29.3%</small>	499.7 <small>-15.7%</small>	377.4 <small>-14.2%</small>	96.8% <small>1.2%</small>	1.953 <small>-14.3%</small>
DeepSeek-R1-Distill-Qwen-32B	7602	733.1	278.8	92.4%	2.219
w/ TrimR	2511 <small>-67.0%</small>	315.8 <small>-56.9%</small>	218.0 <small>-21.8%</small>	94.4% <small>2.0%</small>	1.330 <small>-40.1%</small>
Pangu-R-38B	3665	447.4	300.8	95.6%	1.912
w/ TrimR	2426 <small>-33.8%</small>	367.6 <small>-17.8%</small>	264.6 <small>-12.0%</small>	94.4% <small>-1.2%</small>	1.551 <small>-18.9%</small>
AIME24					
QwQ 32B	6992	2437.6	2138.6	76.6%	3.189
w/ TrimR	4255 <small>-39.1%</small>	1572.6 <small>-35.5%</small>	1431.6 <small>-33.1%</small>	76.6% <small>-0%</small>	2.444 <small>-23.3%</small>
DeepSeek-R1-Distill-Qwen-32B	10299	3717.6	3156.0	66.6%	3.252
w/ TrimR	4799 <small>-53.4%</small>	1433.9 <small>-61.4%</small>	1228.7 <small>-61.1%</small>	70.0% <small>3.3%</small>	2.096 <small>-35.6%</small>
Pangu-R-38B	6164	1912.3	1639.4	78.3%	2.466
w/ TrimR	3848 <small>-37.6%</small>	1299.4 <small>-32.0%</small>	1154.8 <small>-29.6%</small>	76.6% <small>-1.7%</small>	2.006 <small>-18.6%</small>
AIME25					
QwQ 32B	7436	2771.5	2513.8	60.0%	3.426
w/ TrimR	6215 <small>-16.4%</small>	2302.5 <small>-16.9%</small>	2032.4 <small>-19.2%</small>	60.8% <small>0.8%</small>	3.070 <small>-10.4%</small>
DeepSeek-R1-Distill-Qwen-32B	13055	5474.3	4861.7	47.9%	3.932
w/ TrimR	6169 <small>-52.7%</small>	1897.0 <small>-65.3%</small>	1549.6 <small>-68.1%</small>	56.3% <small>8.4%</small>	2.434 <small>-38.1%</small>
Pangu-R-38B	9216	3053.4	2723.6	57.5%	3.117
w/ TrimR	5591 <small>-39.3%</small>	1958.6 <small>-35.9%</small>	1731.6 <small>-36.4%</small>	57.5% <small>0.0%</small>	2.470 <small>-20.8%</small>
GPQA Diamond					
QwQ 32B	4406	1302.6	1115.0	66.0%	1.572
w/ TrimR	3198 <small>-27.4%</small>	1170.7 <small>-10.1%</small>	1025.5 <small>-8.0%</small>	65.2% <small>-0.8%</small>	1.438 <small>-8.5%</small>
DeepSeek-R1-Distill-Qwen-32B	11366	3568.0	2786.2	45.4%	2.447
w/ TrimR	3411 <small>-70.0%</small>	902.0 <small>-74.7%</small>	720.4 <small>-74.1%</small>	58.6% <small>13.2%</small>	1.320 <small>-46.1%</small>
Pangu-R-38B	3120	994.7	866.8	59.1%	1.378
w/ TrimR	2516 <small>-19.4%</small>	901.2 <small>-9.4%</small>	788.9 <small>-9.0%</small>	60.1% <small>1.0%</small>	1.273 <small>-7.6%</small>

4.3 ANALYSIS

408 **Effects of Trimming Methods.** Ablation results in Table 3 demonstrate that combining overthinking
 409 and underthinking trimming achieves the greatest efficiency gains with minimal accuracy trade-
 410 offs. For QwQ-32B on MATH500, overthinking trimming alone reduces TPR by 12.0% and tokens
 411 by 10.6% while improving accuracy by 1.2%, whereas underthinking trimming yields smaller gains
 412 (TPR: -4.8%, tokens: -3.3%) with a minor 0.6% accuracy drop. Their combination maintains the
 413 96.8% accuracy while achieving TPR and token reductions of 15.8% and 14.3%, respectively.

414 For Pangu-R-38B, the combined approach reduces TPR by 17.8% and tokens by 18.9% on
 415 MATH500 with negligible 1.2% accuracy loss. R1Q-32B shows even stronger gains: combined
 416 trim slashes TPR by 56.9% and tokens by 40.1% while boosting accuracy from 90.4% to 92.4%.
 417 Model-specific patterns emerge: QwQ-32B exhibits lower redundancy (overthinking TPR reduc-
 418 tion: 12.0% vs. 43.5% for R1Q-32B). Conversely, R1Q-32B’s high token usage (2.447 vs. 1.572 for
 419 QwQ-32B on GPQA) reflects frequent self-verification, which dynamic trimming mitigates. Vari-
 420 ations in trimming efficacy across models and benchmarks (e.g., QwQ-32B: MATH500 TPR -12.0%
 421 vs. AIME24 -13.1%) underscore the need to apply both strategies to optimize efficiency across
 422 tasks.

423 **Effects on Distribution.** Fig. 4 depicts the empirical token-count distributions for our reasoning tasks before and after applying TrimR to QwQ-
 424 32B. In the original (untrimmed) setting, approximately 64% of problem
 425 instances fell within the lowest bin (0–5K tokens), with the remainder
 426 spread across higher token ranges (5–32K tokens). After trimming, this
 427 proportion rises to nearly 70%, and the frequency of “long-context” in-
 428 stances (≥ 10 K tokens) drops by over 25%. In particular, the heaviest tail
 429 (20–32K tokens) is reduced by more than two-thirds, from roughly 6%
 430 of cases down to under 2%. This pronounced leftward shift in the
 431 distribution demonstrates that TrimR effectively prunes superfluous context,
 432 lowering the average token footprint per query.



433 **Figure 4:** Token distributions with/without TrimR.

432 **Table 3:** Ablation analysis of overthinking and underthinking trimming, showing that both methods markedly
 433 reduce TPR and token usage without compromising reasoning accuracy.

	MATH500			AIME24		
	TPR	#Tokens	Accuracy	TPR	#Tokens	Accuracy
Pangu-R-38B	-	-	95.6%	-	-	78.3%
w/ overthinking trimming	-16.6%	-16.1%	95.4%	-30.2%	-15.7%	76.6%
w/ underthinking trimming	-10.5%	-8.3%	95.8%	-26.0%	-13.7%	75.4%
w/ both	-17.8%	-18.9%	94.4%	-32.1%	-18.6%	76.6%
QwQ 32B	-	-	95.6%	-	-	76.6%
w/ overthinking trimming	-12.0%	-10.6%	96.8%	-13.1%	-12.4%	76.6%
w/ underthinking trimming	-4.8%	-3.3%	95.0%	-27.0%	-16.1%	76.3%
w/ both	-15.8%	-14.3%	96.8%	-35.5%	-23.3%	76.6%
DeepSeek-R1-Distill-Qwen-32B	-	-	90.4%	-	-	60.0%
w/ overthinking trimming	-43.5%	-28.1%	91.6%	-49.0%	-22.3%	63.3%
w/ underthinking trimming	-41.4%	-25.0%	92.8%	-57.1%	-29.8%	63.8%
w/ both	-56.9%	-40.1%	92.4%	-61.4%	-35.6%	63.3%

447 **Verifier Accuracy.** To assess chain-level consistency, we used Pangu-R-38B to generate full reasoning
 448 traces for all MATH500 questions, split them into T4S-defined chains, and manually annotated
 449 answer consistency for 684 randomly sampled adjacent chain pairs.

450 **Table 4:** Verification accuracy, and downstream performance of Pangu-7B and Qwen2.5-7B-Instruct verifiers
 451 on the MATH500 dataset. *Verifier Acc.(%)* denotes the fraction of correctly judged chain pairs; The rest are
 452 downstream performance on MATH500: *MATH500 Acc.(%)* is downstream task accuracy.

Verifier	Verifier Acc.(%)	Runtime	TPR	MATH500 Acc.(%)	#Tokens(M)
Pangu-7B	87.87	3,665	447.4	95.6	1.912
w/o in context examples	85.67	3,894	455.1	95.6	2.032
Qwen2.5-7B-Instruct	86.70	3,722	459.2	95.0	1.982
w/o in context examples	83.48	3,938	474.2	94.8	2.103

453 As shown in Table 4, Pangu-7B outperforms Qwen2.5-7B-Instruct in annotation accuracy (87.87%
 454 vs. 86.70%), speeds up thinking trimming (3,665s vs. 3,722s), and reduces total tokens (1.912M vs.
 455 1.982M). Downstream accuracy on MATH500 is essentially unchanged (95.6% vs. 95.0%), showing
 456 that occasional consistency errors have negligible effect. Omitting in-context demonstrations in the
 457 verifier’s prompt (in-context examples shown in Appendix F) slightly increases runtime and token
 use of the LRM when the verifiers are applied to the downstream task (Pangu-7B: +229s, +0.120M;
 458 Qwen: +216s, +0.121M). Limited by space, the full results using Qwen as verifier are presented in
 459 Appendix P. Overall, the choice of verifier model has negligible effects on the overall performance.

460 **TrimR in Test-time Scaling with BoN.** Beyond sequential token extension, additional test-time
 461 scaling approaches to improve LRM accuracy include BoN sampling (Lightman et al., 2024),
 462 Monte Carlo Tree Search (MCTS) (Wu et al., 2025b), and beam search (Lightman et al., 2024).
 463 Integrating TrimR with BoN (N=8), as evidenced in Table 9, Appendix Q, demonstrates significant
 464 reductions in token consumption (-13.8-16.2%), and runtime duration (up to 23.3%) while main-
 465 taining performance parity (-3.3% on AIME24). These results highlight TrimR’s broad applicability
 466 across diverse test-time scaling frameworks.

473 5 CONCLUSION

474 This work introduces TrimR, a training-free, verifier-based framework that dynamically trims rea-
 475 soning in Large Reasoning Models (LRMs) to eliminate redundant thinking. By leveraging a
 476 lightweight pre-trained verifier to truncate unnecessary intermediate steps, TrimR significantly
 477 improves inference efficiency without compromising accuracy. Empirical results on MATH500,
 478 AIME24/25, and GPQA benchmarks demonstrate up to a 70% reduction in runtime across models,
 479 particularly in large-batch industrial settings. We also present T4S, TrimR’s online deployment sys-
 480 tem integrated with Ascend NPUs/vLLM, highlighting TrimR’s scalability for high-throughput de-
 481 ployments. By balancing computational efficiency and reasoning rigor, TrimR offers a cost-effective
 482 solution for real-world LRM applications, advancing the viability of production-level AI reasoning
 483 systems.

486 6 REPRODUCIBILITY STATEMENT
487488 The following materials may help reproduce this work.
489490

- 491 • Sec. 3 details the proposed method;
- 492 • Sec. 3.6 discusses the system design of the proposed method;
- 493 • Algorithms 1 and 2 describe the underthinking/overthinking trimming procedures;
- 494 • Appendix F offers the full prompts used in verifiers;
- 495 • Appendix H describes the experimental setup in detail;
- 496 • This study mainly evaluates on open-sourced LRM (QwQ-32B, DeepSeek-R1-Distill-
497 Qwen-32B, and Pangu-Pro-MoE (Appendix N)) and open-sourced verifier models
498 (Qwen2.5-7B-Instruct and Pangu-7B);
- 499 • We will make the code available upon acceptance.

500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539

540 REFERENCES
541

542 AIME2024. American invitational mathematics examination (aime), 2024.

543 AIME2025. American invitational mathematics examination (aime), 2025.

544 Daman Arora and Andrea Zanette. Training language models to reason efficiently, 2025. *URL*
545 <https://arxiv.org/abs/2502.04463>, 2025.

546 Qiguang Chen, Dengyun Peng, Jinhao Liu, HuiKang Su, Jiannan Guan, Libo Qin, and Wanxiang
547 Che. Aware first, think less: Dynamic boundary self-awareness drives extreme reasoning effi-
548 ciency in large language models. *arXiv preprint arXiv:2508.11582*, 2025.

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

552 Siqi Fan, Peng Han, Shuo Shang, Yequan Wang, and Aixin Sun. Cothink: Token-efficient reasoning
553 via instruct models guiding reasoning models. *arXiv preprint arXiv:2505.22017*, 2025.

554 Gongfan Fang, Xinyin Ma, and Xinchao Wang. Thinkless: Llm learns when to think. *arXiv preprint*
555 *arXiv:2505.13379*, 2025.

556 Yichao Fu, Junda Chen, Siqi Zhu, Zheyu Fu, Zhongdongming Dai, Aurick Qiao, and Hao Zhang.
557 Efficiently serving llm reasoning programs with certainindex. *arXiv preprint arXiv:2412.20993*,
558 2024.

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

562 Tingxu Han, Zhenting Wang, Chunrong Fang, Shiyu Zhao, Shiqing Ma, and Zhenyu Chen. Token-
563 budget-aware llm reasoning. *arXiv preprint arXiv:2412.18547*, 2024.

564 Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong
565 Tian. Training large language models to reason in a continuous latent space. *arXiv preprint*
566 *arXiv:2412.06769*, 2024.

567 Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song,
568 and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv*
569 *preprint arXiv:2103.03874*, 2021.

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

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

576 Lorenz Kuhn, Yarin Gal, and Sebastian Farquhar. Semantic uncertainty: Linguistic invariances for
577 uncertainty estimation in natural language generation. *arXiv preprint arXiv:2302.09664*, 2023.

578 Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E.
579 Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model
580 serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating*
581 *Systems Principles*, 2023.

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

585 Zhengkai Lin, Zhihang Fu, Ze Chen, Chao Chen, Liang Xie, Wenxiao Wang, Deng Cai, Zheng
586 Wang, and Jieping Ye. Controlling thinking speed in reasoning models. *arXiv preprint*
587 *arXiv:2507.03704*, 2025.

594 Zijun Liu, Peiyi Wang, Runxin Xu, Shirong Ma, Chong Ruan, Peng Li, Yang Liu, and Yu Wu.
 595 Inference-time scaling for generalist reward modeling. *arXiv preprint arXiv:2504.02495*, 2025.
 596

597 Wenjie Ma, Jingxuan He, Charlie Snell, Tyler Griggs, Sewon Min, and Matei Zaharia. Reasoning
 598 models can be effective without thinking. *arXiv preprint arXiv:2504.09858*, 2025.

599 Rohin Manvi, Anikait Singh, and Stefano Ermon. Adaptive inference-time compute: Llms can
 600 predict if they can do better, even mid-generation. *arXiv preprint arXiv:2410.02725*, 2024.
 601

602 Sara Vera Marjanović, Arkil Patel, Vaibhav Adlakha, Milad Aghajohari, Parishad BehnamGhader,
 603 Mehar Bhatia, Aditi Khandelwal, Austin Kraft, Benno Krojer, Xing Han Lù, et al. Deepseek-r1
 604 thoughtology: Let's; think; about llm reasoning. *arXiv preprint arXiv:2504.07128*, 2025.

605 Tergel Munkhbat, Namgyu Ho, Seo Hyun Kim, Yongjin Yang, Yujin Kim, and Se-Young Yun. Self-
 606 training elicits concise reasoning in large language models. *arXiv preprint arXiv:2502.20122*,
 607 2025.

608 OpenAI. Openai o1 system card, December 2024. URL <https://openai.com/index/openai-o1-system-card/>.

609 Rui Pan, Yinwei Dai, Zhihao Zhang, Gabriele Oliaro, Zhihao Jia, and Ravi Netravali. Specreation-
 610 son: Fast and accurate inference-time compute via speculative reasoning. *arXiv preprint arXiv:2504.07891*, 2025.

611 Pang Team. openpangu-embedded-7b-model, 2025. URL <https://ai.gitcode.com/ascend-tribe/openpangu-embedded-7b-model>. Ascend-native open-source reposi-
 612 tory for the openPang Embedded 7B model.

613 Xiaoye Qu, Yafu Li, Zhaochen Su, Weigao Sun, Jianhao Yan, Dongrui Liu, Ganqu Cui, Daizong
 614 Liu, Shuxian Liang, Junxian He, et al. A survey of efficient reasoning for large reasoning models:
 615 Language, multimodality, and beyond. *arXiv preprint arXiv:2503.21614*, 2025.

616 Qwen Team. Qwen2.5: A party of foundation models, September 2024. URL <https://qwenlm.github.io/blog/qwen2.5/>.

617 David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Di-
 618 rani, Julian Michael, and Samuel R Bowman. Gpqa: A graduate-level google-proof q&a bench-
 619 mark. In *First Conference on Language Modeling*, 2024.

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

622 Yang Sui, Yu-Neng Chuang, Guanchu Wang, Jiamu Zhang, Tianyi Zhang, Jiayi Yuan, Hongyi Liu,
 623 Andrew Wen, Hanjie Chen, Xia Hu, et al. Stop overthinking: A survey on efficient reasoning for
 624 large language models. *arXiv preprint arXiv:2503.16419*, 2025.

625 Hanshi Sun, Momin Haider, Ruiqi Zhang, Huitao Yang, Jiahao Qiu, Ming Yin, Mengdi Wang, Peter
 626 Bartlett, and Andrea Zanette. Fast best-of-n decoding via speculative rejection. *arXiv preprint arXiv:2410.20290*, 2024.

627 Yehui Tang, Xiaosong Li, Fangcheng Liu, Wei Guo, Hang Zhou, Yaoyuan Wang, Kai Han, Xianzhi
 628 Yu, Jinpeng Li, Hui Zang, Fei Mi, Xiaojun Meng, Zhicheng Liu, Hanting Chen, Binfan Zheng,
 629 Can Chen, Youliang Yan, Ruiming Tang, Peifeng Qin, Xinghao Chen, Dacheng Tao, and Yunhe
 630 Wang. Pangu pro moe: Mixture of grouped experts for efficient sparsity, 2025. URL <https://arxiv.org/abs/2505.21411>.

631 Kimi Team, A Du, B Gao, B Xing, C Jiang, C Chen, C Li, C Xiao, C Du, C Liao, et al. Kimi k1. 5:
 632 Scaling reinforcement learning with llms, 2025. URL <https://arxiv.org/abs/2501.12599>, 2025.

633 Qwen Team. Qwq-32b: Embracing the power of reinforcement learning, March 2025. URL
 634 <https://qwenlm.github.io/blog/qwq-32b/>.

648 Jikai Wang, Juntao Li, Lijun Wu, and Min Zhang. Efficient reasoning for llms through speculative
 649 chain-of-thought. *arXiv preprint arXiv:2504.19095*, 2025a.
 650

651 Yiming Wang, Pei Zhang, Siyuan Huang, Baosong Yang, Zhuosheng Zhang, Fei Huang, and Rui
 652 Wang. Sampling-efficient test-time scaling: Self-estimating the best-of-n sampling in early de-
 653 coding. *arXiv preprint arXiv:2503.01422*, 2025b.
 654

655 Han Wu, Yuxuan Yao, Shuqi Liu, Zehua Liu, Xiaojin Fu, Xiongwei Han, Xing Li, Hui-Ling Zhen,
 656 Tao Zhong, and Mingxuan Yuan. Unlocking efficient long-to-short llm reasoning with model
 657 merging. *arXiv preprint arXiv:2503.20641*, 2025a.
 658

659 Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. Inference scaling laws:
 660 An empirical analysis of compute-optimal inference for llm problem-solving. In *The Thirteenth
 661 International Conference on Learning Representations*, 2025b.
 662

663 Zhiheng Xi, Dingwen Yang, Jixuan Huang, Jiafu Tang, Guanyu Li, Yiwen Ding, Wei He, Boyang
 664 Hong, Shihan Do, Wenyu Zhan, et al. Enhancing llm reasoning via critique models with test-time
 665 and training-time supervision. *arXiv preprint arXiv:2411.16579*, 2024.
 666

667 Heming Xia, Yongqi Li, Chak Tou Leong, Wenjie Wang, and Wenjie Li. Tokenskip: Controllable
 668 chain-of-thought compression in llms. *arXiv preprint arXiv:2502.12067*, 2025.
 669

670 Silei Xu, Wenhao Xie, Lingxiao Zhao, and Pengcheng He. Chain of draft: Thinking faster by writing
 671 less. *arXiv preprint arXiv:2502.18600*, 2025.
 672

673 Yuchen Yan, Yongliang Shen, Yang Liu, Jin Jiang, Mengdi Zhang, Jian Shao, and Yueting Zhuang.
 674 Inftythink: Breaking the length limits of long-context reasoning in large language models. *arXiv
 675 preprint arXiv:2503.06692*, 2025.
 676

677 An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li,
 678 Chengyuan Li, Dayiheng Liu, Fei Huang, et al. Qwen2 technical report. *arXiv preprint
 679 arXiv:2407.10671*, 2024.
 680

681 Chenxu Yang, Qingyi Si, Yongjie Duan, Zheliang Zhu, Chenyu Zhu, Zheng Lin, Li Cao, and Weiping
 682 Wang. Dynamic early exit in reasoning models. *arXiv preprint arXiv:2504.15895*, 2025.
 683

684 Ping Yu, Jing Xu, Jason Weston, and Ilia Kulikov. Distilling system 2 into system 1. *arXiv preprint
 685 arXiv:2407.06023*, 2024.
 686

687 XING Zeyu, Xing Li, Huiling Zhen, Xianzhi Yu, Mingxuan Yuan, and Sinno Jialin Pan. Large
 688 reasoning models know how to think efficiently. In *ES-FoMo III: 3rd Workshop on Efficient
 689 Systems for Foundation Models*, 2025.
 690

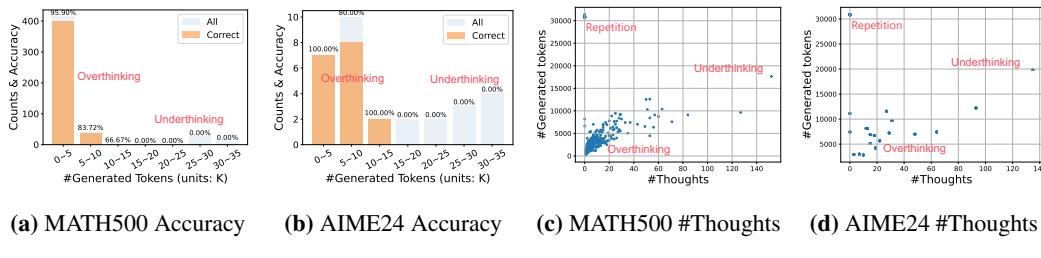
691 Jintian Zhang, Yuqi Zhu, Mengshu Sun, Yujie Luo, Shuofei Qiao, Lun Du, Da Zheng, Huajun
 692 Chen, and Ningyu Zhang. Lighththinker: Thinking step-by-step compression. *arXiv preprint
 693 arXiv:2502.15589*, 2025a.
 694

695 Junyu Zhang, Runpei Dong, Han Wang, Xuying Ning, Haoran Geng, Peihao Li, Xialin He, Yutong
 696 Bai, Jitendra Malik, Saurabh Gupta, et al. Alphaone: Reasoning models thinking slow and fast at
 697 test time. *arXiv preprint arXiv:2505.24863*, 2025b.
 698

699 Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Livia Sun, Jeff Huang, Cody Hao Yu, Shiyi
 700 Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. Sqlang: Efficient execution of
 701 structured language model programs. *Advances in Neural Information Processing Systems*, 37:
 62557–62583, 2024.

702 **A USE OF LARGE LANGUAGE MODELS**
703704 We employed LLMs for proofreading the manuscript, while ensuring that all substantive points and
705 contributions remain entirely original.
706707 **B ALGORITHM OF UNDERTHINKING TRIM**
708709 We provide the formulation of underthinking trim in Algorithm 2. The underthinking detection
710 depends on the result of overthinking detection. If a sequence can not converge to a solution within
711 the given budget R_{thres} and N_{thres} , we use *stop_tokens* (Sec. 3.5) to stop further thinking.
712713 **Algorithm 2:** Underthinking Trimming

714 **Input:** Input X , underthinking threshold R_{thres} , N_{thres}
 715 **Output:** Generated output Y
 716 $stopped \leftarrow False$;
 717 $y_t \leftarrow p_{\Pi}(X)$;
 718 **while** $y_t \neq eos$ **do** // LRM iterately generates y_t in while loop
 719 **// Check if this sequence is flagged as overthinking**
 720 $is_overthinking \leftarrow$ check Algorithm 1 current state;
 721 $num_thoughts \leftarrow \text{len}(\text{Think_Seg}(y_{\leq t}))$;
 722 **if** $y_t = /think$ **then**
 723 $stopped = True$, **break**
 724 **end**
 725 **if** $t > R_{thres} \% \cdot M$ and $num_thoughts > N_{thres}$ and $stopped = False$ **then**
 726 $y_t \leftarrow p_{\Pi}((X, y_{<t}, stop_tokens))$, $stopped = True$, **break**;
 727 **else**
 728 $y_t \leftarrow p_{\Pi}((X, y_{<t}, y_t))$
 729 **end**
 730 **end**
 731 **return** $(y_{<t}, y_t)$;

728 **C MORE ANALYSIS OF THINKING LENGTH AND ACCURACY**
729730 The distribution of decoding length occurrence and corresponding accuracy of Deepseek-R1-Distill-
731 Qwen-32B (R1Q-32B) on the MATH500 and AIME24 datasets are available in Fig. 5a, 5b. The
732 correlation of the number of thoughts and generated tokens are available in Fig. 5c, 5d.
733734 The Deepseek-R1-Distill-Qwen-32B model is weaker than QwQ-32B and tends to generate long but
735 wrong responses, which is underthinking. The accuracy of R1Q-32B quickly decreases from above
736 80%~90% to 0% as visualized in Fig. 5a, 5b. In contrast, QwQ-32B still is able to solve some
737 complex questions as in Fig. 2a, 2b. In addition, the repetition occurs more frequently, resulting in
738 non-stopping 32K tokens. The number of thoughts of R1Q-32B linearly correlates with the number
739 of tokens, similar to QwQ-32B in Fig. 2.
740748 **Figure 5:** Histogram of occurrence and accuracy of decoding length and scatter plot of the number of thoughts
749 and generated tokens of of Deepseek-R1-Distill-Qwen-32B (R1Q-32B) on MATH500 and AIME24 datasets
750751 **D DECODING THROUHPUT/LATENCY OVER DECODING LENGTH**
752753 Deploying LRM in large-scale production environments presents substantial challenges for im-
754 proving reasoning efficiency. First, effective methods to mitigate redundant reasoning in LRM are
755 critical, as such inefficiencies significantly hinder the performance of inference systems in produc-
756 tion. Since the generation latency of LLMs typically increases linearly with decoding length (as

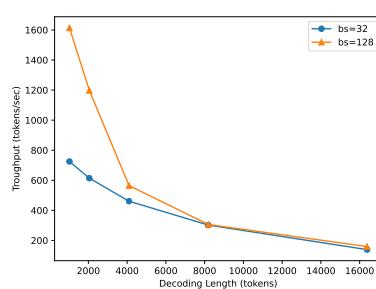


Figure 6: Throughput reduction due to increasing length of QwQ-32B. The one-step attention complexity is linear to the sequence length or KV cache length, so inference with long decoding length or long input length has the same issue. For example, for the 32K decoding step, the latency of attention is same as the first decoding step with 32K input length, which is significantly higher than the attention latency with shorter length.

shown in Figure 6), reducing unnecessary token generation can yield super-linear gains in runtime reduction relative to the proportion of tokens saved. This, in turn, enhances the efficiency and scalability of test-time compute. Second, proposed solutions must be compatible with state-of-the-art inference infrastructures designed for large-scale deployment, such as vLLM and SGLang.

E FULL LIST OF REFLECTION TOKENS

We utilize the following markers as reflection tokens to partition model reasoning into sub-thoughts: “\n\nBut”, “\n\nWait”, “\n\nHowever”, “\n\nHmm”, “\n\nLet me verify this”, and “\n\nAlternatively”. We do not utilize those without “\n\n” such as “but” and “But” as reflection tokens to reduce the number of answer existence checking with verifiers, because many are in the internal step checking before approaching answers. “\n\n” is a strong structural separator for different thoughts in Deepseek R1 and Qwen QwQ-32B.

These reflection tokens were chosen by prompting the LRM with a small set of manually-crafted questions and then capturing the outstanding reflection phrases. Some of the questions are presented as follows:

- Emma walks into her kitchen in the morning and finds the floor wet and several small puddles near the sink. She notices the faucet handle is turned on, but no water is flowing. What most likely happened, and what should Emma do next?
- An A/B test shows a +2.1% uplift with $p=0.049$ on day 7, but on day 14 the effect shrinks to +0.7% with $p=0.18$. Provide at least two explanations and the next diagnostic checks.
- A classifier has 98% accuracy on a dataset where 97% of samples are negative. Why might this be misleading? Propose better metrics and a quick verification.

(Continue on the next page)

810 F DETAILED PROMPTS
811
812
813814 Default Verifier Prompt for Answer Existence p_1
815

816 Check if the following sentences from a reasoning model reach a conclusion, regardless of formatting or phrasing, and solve the
817 question. Return 'Yes' if the content finds a solution, otherwise 'No'. Return only 'Yes' or 'No' with no explanation.

818 **Example 1:**

819 **Question:** $2 + 3 = ?$

820 **Content:** The answer is 5.

821 You should return Yes.

822 **Example 2:**

823 **Question:** $2 + 3 = ?$

824 **Content:** I think it should be 5, but I am not sure.

825 You should return Yes.

826 **Example 3:**

827 **Question:** $2 + 3 = ?$

828 **Content:** Wait, I think I made a mistake.

829 You should return Yes.

830 **Example 4:**

831 **Question:** If $f(x) = \frac{3x-2}{x-2}$, what is the value of $f(-2) + f(-1) + f(0)$? Express your answer as a common fraction.

832 **Content:** $6 + 5$ is 11, and $11 + 3$ is 14. Yes, so $\frac{14}{3}$. So, $f(-2) + f(-1) + f(0) = \frac{14}{3}$.

833 You should return Yes.

834 **Example 5:**

835 **Question:** If $f(x) = \frac{3x-2}{x-2}$, what is the value of $f(-2) + f(-1) + f(0)$? Express your answer as a common fraction.

836 **Content:** Since all denominators are 3, we can add the numerators: $6 + 5 + 3 = 14$. Therefore, the sum is $\frac{14}{3}$.

837 You should return Yes.

838 **Example 6:**

839 **Question:** If $f(x) = \frac{3x-2}{x-2}$, what is the value of $f(-2) + f(-1) + f(0)$? Express your answer as a common fraction.

840 **Content:** Wait, another thought: When adding the fractions, is $\frac{14}{3}$ the correct sum? Let's compute it in decimal to cross-verify. $\frac{14}{3}$ divided is approximately 4.666...

841 You should return No.

842 **Question:** \$QUESTION

843 **Content:** \$CONTENT

844 [Yes/No]:

845 Verifier Prompt for Answer Consistency p_2

846 Check if the two answers provided are logically or mathematically equivalent, regardless of formatting or phrasing. Return 'Yes' if
847 they are equal in meaning/value and a valid solution to the question, otherwise 'No'. Return only 'Yes' or 'No' with no explanation.

848 **Example 1:**

849 **Question:** $2 + 3 = ?$

850 **Answer1:** the answer is 5.

851 **Answer2:** the answer seems to be five.

852 Are they equivalent? [Yes/No]: Yes

853 **Example 2:**

854 **Question:** Define

$$855 \quad p = \sum_{k=1}^{\infty} \frac{1}{k^2} \quad \text{and} \quad q = \sum_{k=1}^{\infty} \frac{1}{k^3}.$$

856 Find a way to write

$$857 \quad \sum_{j=1}^{\infty} \sum_{k=1}^{\infty} \frac{1}{(j+k)^3}$$

858 in terms of p and q .

859 **Answer 1:**

$$860 \quad (p - 1) - (q - 1) = p - 1 - q + 1 = p - q$$

861 Therefore, the original double sum is equal to $p - q$.

862 **Answer 2:** Given that the terms are decreasing and positive, the convergence seems plausible. However, since $p - q \approx 0.4429$ and
863 our partial sum is about 0.3513, it's possible that the analytical result is correct. Therefore, unless there's a mistake in the analytical
864 steps, the answer should be $p - q$.

865 Are they equivalent? [Yes/No]: Yes

866 **Question:** \$QUESTION

867 **Answer 1:** \$ANSWER1

868 **Answer 2:** \$ANSWER2

869 Are they equivalent? [Yes/No]:

864

Guidance Prompt to Stop Thinking

865

866

Gentle Prompt (Overthinking Trim):

867

\n\n I think I already thought for a long time, and I should be able to put the simple
 final answer after **Final Answer**\n with \boxed{} and end thinking now! ****Final**
Answer\n**

868

869

Forceful Prompt (Underthinking & Repetition Truncation):

870

\n\n I think I already thought for a long time, and I should be able to deduce the
 final answer from the content above. </think>\n\n

871

872

873

874

875

876

877

878

G OVERTHINKING/UNDERTHINKING EXAMPLES

879

880

881

882

The following is an example of overthinking in QwQ-32B. The question comes from AIME24. During the thinking process, reflection tokens like “\n\nWait”, “\n\nBut” and “\n\nAlternatively” appear 14 times. After the reflection tokens, the content verifies the answer “204” which first appears in the first third of the entire reasoning process.

883

884

885

886

887

888

889

890

An example of overthinking phenomenon (QwQ-32B)

891

892

Question: Every morning Aya goes for a 9-kilometer-long walk and stops at a coffee shop afterwards. When she walks at a constant speed of s kilometers per hour, the walk takes her 4 hours, including t minutes spent in the coffee shop. When she walks $s + 2$ kilometers per hour, the walk takes her 2 hours and 24 minutes, including t minutes spent in the coffee shop. Suppose Aya walks at $s + \frac{1}{2}$ kilometers per hour. Find the number of minutes the walk takes her, including the t minutes spent in the coffee shop.

893

894

895

896

897

898

899

900

901

902

903

904

905

906

907

908

909

910

911

912

913

914

915

The example of underthinking in QwQ-32B is present below. The question taken from AIME24 is difficult and the LRM doesn't reach the correct answer. The reasoning contents switch between different approaches frequently and can't provide an answer confidently. In fact, the figure 37/128 is made up without reasoning and the answer “475” has been checked to be wrong in previous thinking.

918
919

An example of underthinking phenomenon (QwQ-32B)

920
921
922
923

Question: Each vertex of a regular octagon is independently colored either red or blue with equal probability. The probability that the octagon can then be rotated so that all of the blue vertices end up at positions where there were originally red vertices is $\frac{m}{n}$, where m and n are relatively prime positive integers. What is $m + n$?

924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940

Answer: Okay, so I need to find the probability that **Alternatively**, using Burnside's lemma **Wait**, Burnside's lemma counts the number of orbits, but I'm not sure. **Alternatively**, let me think of it as follows **Therefore**, this is the same as before. Perhaps the expected value approach **But** expectation of the union is tricky, but maybe we can approximate it **Alternatively**, since B is randomly chosen, the events for different elements are not independent, but perhaps we can compute it **Alternatively**, since the problem is from an Olympiad, the answer might be **37/128** or something **Alternatively**, maybe the correct answer is **37/128**? **Wait**, let me think differently **Alternatively**, perhaps the number is equal to the sum over all non-zero rotations of **But** without knowing the overlaps, it's hard This is getting too involved. Given that this is an Olympiad problem, perhaps there's a smarter way **But** without the second term, we can't compute it. **Alternatively**, perhaps assume that the events are independent **Wait**, this is confusing. **Alternatively**, if the number of subsets suggests that this count is too low. **Alternatively**, perhaps the correct answer is **37/128**? **Wait** **But** the problem says "can be rotated so that all blue vertices end up at originally red vertices" **Therefore**, perhaps the count is indeed **219**, giving **475**. Given that this is a competition problem, and **475** is a plausible answer (as $219+256=475$), I think this is the intended answer. **Therefore**, I will go with **475**. **Final Answer** **475** </think>

H DETAILED EXPERIMENT SETUP

Benchmark description: **MATH500** (Hendrycks et al., 2021) comprises 500 problems spanning arithmetic, algebra, geometry, and calculus with varying difficulty levels. It evaluates models' ability in complex mathematical formalism, equation solving, and reasoning. **AIME24** (AIME2024) consists of Olympiad-style problems assessing logical deduction and advanced problem-solving skills. **AIME25** (AIME2025) features updated problems from the same competition as AIME24. **GPQA Diamond** (Rein et al., 2024) is a challenging dataset containing 198 multiple-choice questions, written by domain experts in biology, physics, and chemistry.

All the experiments are conducted on servers with 8 Ascend 910B-64GB NPUs and 192-cores Kunpeng-920 CPUs with 1.5TB memory. During decoding, TorchAir (Torch Ascend Intermediate Representation) captures the computation graph to accelerate kernel dispatch and alleviate host-bound bottlenecks. The maximum number of concurrent decoding batches is set to 128. We configure vLLM to pre-allocate eight scheduler steps, thereby reducing scheduling overhead. All dataset requests (e.g., the 500 questions in MATH500) are submitted to vLLM concurrently, and we record key metrics such as total wall-clock time and per-request latency. We employ the Qwen2.5 math evaluation tool to score the solutions (Yang et al., 2024) and apply postprocessing to ensure that formatting quirks (e.g., spacing, notation style) don't penalize valid solutions.

In the BoN experiments, we use Pangu-ORM (close-source) as the Outcome Reward Model to select the best solution from the generated N solutions. The ratio of LRM and ORM is 1:1 in our experiments, although in production this ratio can be much higher.

I EFFECTIVENESS OF REPETITION TRUNCATION

Table 5: Effects of Repetition Truncation over five GPQA Diamond runs.

Model	Runtime (H:M:S)	TPR	Accuracy	#Tokens	Detected Repetitions
R1Q-32B <i>with repetition truncation</i>	4:02:51 3:09:26	5355.66 3568.02	0.444 0.454	3.09M 2.45M	— 29 out of 198

As shown in Table 5, applying repetition truncation to DeepSeek-R1-Distill-Qwen-32B yields substantial efficiency gains without sacrificing—and even slightly improving—accuracy: Enabling truncation reduces total runtime by 22.0%, cuts TPR by 50.10%, and decreases token consumption by

20.7%, while delivering a 1% accuracy gain. Further analysis of truncated outputs confirms that most early termination occur during later stages of reasoning, where the model becomes stuck in particular attention patterns and fails to generate diverse contents. We also found that the guidance prompt effectively mitigates infinite repetitive outputs by steering the language model toward contextually relevant generation grounded in prior analysis. This module thus serves as an effective, low-overhead component of the T4S, streamlining inference and enhancing answer clarity.

J UNDERTHINKING THRESHOLDS

We chose our initial configurations for R_{thres} and N_{thres} by prompting the model with a small suite of several challenging calculus problems that are not within the test sets of the benchmark datasets. We inspected model traces: on these problems, Pangu-R-38B typically exhibited underthinking after $\sim 15K$ tokens, whereas successful solutions consistently converged within 10K tokens and 20 rounds. Guided by these observations, we fixed a uniform $R_{thres} = 50\%$ and $N_{thres} = 20$ for all benchmarks and models, under which performance remained strong.

One of the questions that can easily trigger underthinking is:

$$S_N = \sum_{k=1}^N \ln\left(1 + \frac{k}{N}\right) - N \ln 2 + \frac{1}{2} \ln N.$$

Tasks:

1. Prove that S_N admits a full asymptotic expansion as $N \rightarrow \infty$ via the Euler-Maclaurin formula; compute explicitly the first four terms (including the constant term).
2. Identify the Bernoulli-number contributions and state a rigorous remainder bound in terms of $\|f^{(m)}\|_\infty$ on $[0, 1]$.
3. Discuss numerical stability: which truncation order minimizes the *actual* error for moderately large N (e.g., $N \approx 10^3$ – 10^5)?

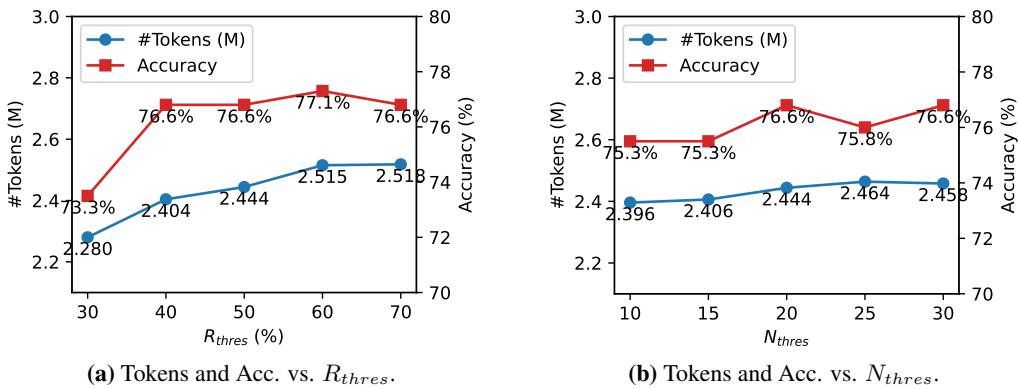


Figure 7: Effect of token- and round-budget thresholds on token count and accuracy (QwQ-32B on AIME24). Although performance is largely insensitive to R_{thres} and N_{thres} , accuracy degrades when $R_{thres} < 40\%$.

After setting the hyperparameters used for deriving the main results, we change these values and observe the performance as an ablation study. Figure 7 plots accuracy and token usage against the token threshold R_{thres} and round threshold N_{thres} . Raising R_{thres} from 30% to 40% increases accuracy from 73.3% to 76.6% while tokens rise modestly (2.28 M to 2.40 M). Beyond 50%, accuracy plateaus but token count continues to grow. Likewise, increasing N_{thres} from 10 to 20 rounds boosts accuracy to 76.6% (2.44 M tokens) with no clear gains thereafter. Though lowering the threshold to $R_{thres} = 40\%$ yields greater token savings without sacrificing accuracy, our default operating point set by inspection, though not optimal, already offers a reasonably good trade-off between token efficiency and accuracy.

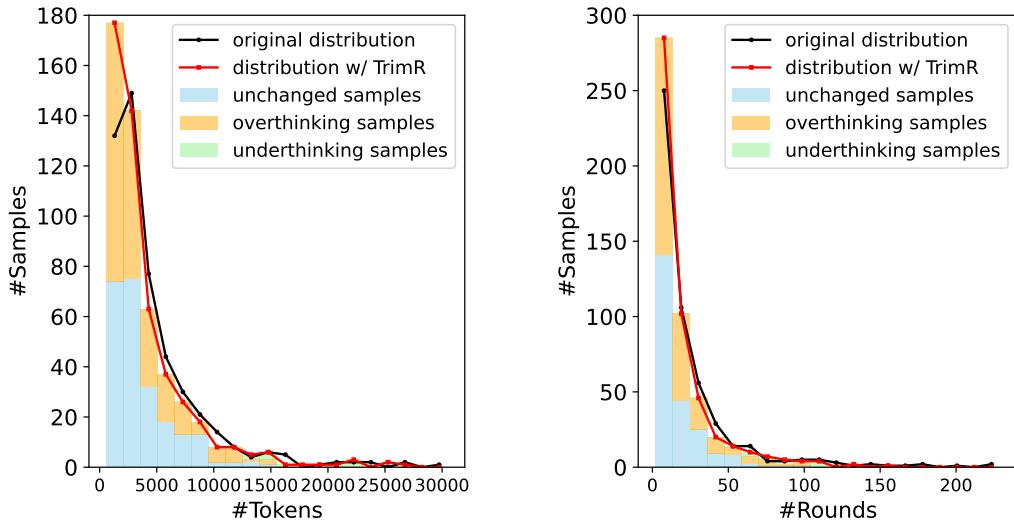
1026 K DISCUSSION ON PERFORMANCE CHANGES

1028 We performed a case-by-case review of all instances in which correctness shifted—either positively
 1029 or negatively—after applying TrimR.

1031 In those cases where accuracy improved, two phenomena were at work. First, on some occasions
 1032 (approximately 30%), the model would descend into repetitive or lengthy generation and exhaust
 1033 its token budget without ever producing a final answer; TrimR’s early-exit mechanism enables the
 1034 model to stop and emit its best partial solution instead, turning non-answers into correct outputs.
 1035 Second, we observed that very long chains of thought can degrade model performance, as global
 1036 attention becomes less effective over extended sequences. In such samples, the model arrived at a
 1037 correct solution midway through its reasoning but then later contradicted itself; by detecting conver-
 1038 gence and triggering a timely exit, TrimR preserves that correct intermediate answer.

1039 On the other hand, a small fraction of samples saw a slight drop in accuracy. In these cases, the
 1040 verifier identified a premature convergence on an incorrect hypothesis—often when the model re-
 1041 peatedly reaffirmed the same wrong answer—thereby cutting off subsequent reflections that might
 1042 have corrected the error. While such instances are uncommon, they highlight a trade-off inher-
 1043 ent in any early-exit strategy between minimizing wasted computation and allowing extra time for
 1044 late-stage corrections. Overall, however, the net effect of TrimR remains strongly positive, yielding
 1045 significant token savings with only minimal impact on accuracy.

1046 L EXTENDED ANALYSIS ON DISTRIBUTION



1066 (a) The distribution of number of tokens with and
 1067 without TrimR

1066 (b) The distribution of number of rounds with and
 1067 without TrimR

1069 **Figure 8:** The distribution of tokens and reasoning rounds with and without TrimR (QwQ-32B on MATH500).
 1070 The original distribution is indicated by the black curve, while the unchanged, overthinking, and underthinking
 1071 samples are shown as stacked bars. After applying dynamic think trim, both the total tokens and number of
 1072 reasoning rounds are substantially reduced compared to the original distribution.

1073 As shown in Figure 8a, dynamic thinking trim produces a marked leftward shift in the token–usage
 1074 distribution. In the uncompressed model, the “knee” lies around 3,500 tokens, with a substantial
 1075 tail beyond 10,000 tokens. After truncation, over 80% of samples require fewer than 6,000 tokens,
 1076 a roughly 30% increase. Also, the heavy tails ($>15,000$ tokens) are nearly eliminated. Decompos-
 1077 ing the stacked bars reveals that unchanged samples remain tightly clustered in the low-token bins
 1078 ($\leq 4,000$ tokens), overthinking samples are effectively truncated into the lower-range bins (0–8,000
 1079 tokens), and underthinking samples ($< 5\%$ of cases) occupy lengths that are modest relative to the
 original tail.

1080 A parallel effect appears in the reasoning-round distribution (Figure 8b). Prior to trimming, a non-
 1081 trivial fraction extends beyond 50 rounds (with outliers over 200), whereas after trimming over 85%
 1082 of samples complete within 30 rounds—an increase of nearly 20 percentage points in the ≤ 20 -round
 1083 regime. Overthinking cases shift from the heavy tail into the 0–30-round interval, while underthink-
 1084 ing cases, though rare, have been truncated to mid-range (50–120 rounds).

1085 Taken together, these results show that TrimR preserves valid reasoning, curtails redundant over-
 1086 thinking, and minimally affects cases needing additional confirmation—thus markedly improving
 1087 inference efficiency in both token and round dimensions.

1089 M ONE VERIFIER CAN SERVE MULTIPLE LRMs

1090 A single verifier model can simultaneously support multiple LRM instances. Runtime performance
 1091 data indicates that when serving a single LRM instance, the verifier’s computational workload re-
 1092 mains within manageable thresholds. Each LRM instance generates an average request rate of 9
 1093 requests per second, while the verifier demonstrates an average processing capacity of approxi-
 1094 mately 128 requests per second - establishing an LRM-to-verifier request ratio of 14:1 under ideal
 1095 conditions. However, production systems adopt a conservative 8:1 deployment ratio to maintain
 1096 operational safety margins.

1097 When accounting for verifier infrastructure costs, TPR (Time Per Request) improvements must be
 1098 adjusted by a cost-efficiency factor: $\frac{14 \times 8}{14 \times 8 + 1} = 99.1\%$ (8 Ascend 910-64GB per LRM instance, 1
 1099 Ascend 910-64GB per verifier instance). This calculation demonstrates that the performance gains
 1100 per computational instance remain effectively preserved (99.1%) despite the additional verification
 1101 overhead. It is noteworthy that enhancements in token utilization and user-perceived TPR cannot
 1102 be further optimized by scaling the number of computing instances within the cluster. In contrast,
 1103 TrimR demonstrates significant performance gains by reducing user waiting times, achieving mea-
 1104 surable improvements in latency reduction.

1107 N PERFORMANCE ON PANGU PRO MoE

1108 We present the results on an open-sourced Pangu model, Pangu Pro MoE, released after the date of
 1109 this study, in Table 6. The results show that TrimR can readily reduce the token usage with very
 1110 minor or no performance regression.

1111 **Table 6:** Performance comparison of Pangu-Pro-MoE on the MATH500, AIME24, AIME25, and GPQA
 1112 Diamond benchmarks. Relative **improvements** are highlighted in green, and **regressions** in red.

1115 Model	1116 Runtime(s)	1117 TPR(s)	1118 TPR-T90(s)	1119 Acc.	1120 #Tokens(M)					
MATH500										
Pangu Pro MoE	6520	805.4	569.2	95.2%	1.618					
w/ TrimR	4900	-24.8%	662.2	-17.8%	488.3	-14.2%	95.6%	+0.4%	1.383	-14.5%
AIME24										
Pangu Pro MoE	3047	1263.9	1096.3	80.0%	0.269					
w/ TrimR	2231	-26.8%	1111.0	-12.1%	989.1	-9.8%	76.7%	-3.3%	0.231	-14.1%
AIME25										
Pangu Pro MoE	3273	1361.7	1194.4	70.0%	0.290					
w/ TrimR	2256	-31.1%	1225.8	-10.0%	1112.3	6.9%	66.7%	-3.3%	0.256	-11.7%
GPQA Diamond										
Pangu Pro MoE	7729	1068.0	927.3	76.6%	1.347					
w/ TrimR	6415	-17.0%	924.3	-13.5%	878.8	-15.3%	76.0%	-0.6%	1.146	-14.9%

1121
 1122
 1123
 1124
 1125
 1126
 1127
 1128
 1129
 1130
 1131
 1132
 1133

1134 O RESULTS ON LIVECODEBENCH

1136 Table 7 shows that, similar to the results on
 1137 mathematical and scientific reasoning, TrimR
 1138 can readily reduce token usage by around 10%
 1139 while preserving the performance of models.

1140 Beyond mathematical and scientific reasoning,
 1141 the results on LiveCodeBench (Table 7) further
 1142 demonstrate the robustness of TrimR. TrimR
 1143 achieves nearly 10% token reduction with accu-
 1144 racy drops consistently below 1%. This trade-off
 1145 indicates that the method effectively removes redundant exploration while retaining the essential
 1146 steps needed for correctness. Moreover, its effectiveness across both dense (QwQ-32B) and MoE
 1147 (Pangu-Pro-MoE) architectures suggests that TrimR is largely architecture-agnostic.

1149 P PERFORMANCE OF QWEN 7B AS VERIFIER

1151 The performance of QwQ-32B (LRM) and Qwen2.5-7B-Instruct (verifier) with and without TrimR
 1152 across benchmarks is presented in Table 8. The results show that the choice of verifier model has
 1153 negligible effects on TrimR’s effectiveness.

1154 **Table 8:** Performance of QwQ-32B (LRM) and Qwen2.5-7B-Instruct (verifier) with and without TrimR across
 1155 benchmarks. Relative changes are shown in parentheses.

1157 Dataset	1158 Model	1159 Runtime (s)	1160 TPR (s)	1161 TPR-T90 (s)	1162 Acc.
MATH500	QwQ 32B	4413	593.1	439.7	95.6%
MATH500	QwQ 32B w/ TrimR	3169 (-28.2%)	507.1 (-14.5%)	380.3 (-13.5%)	96.6% (+1.0%)
AIME25	QwQ 32B	7436	2771.5	2513.8	60.0%
AIME25	QwQ 32B w/ TrimR	6321 (-15.0%)	2342.2 (-15.5%)	2061.3 (-18.0%)	60.8% (+0.8%)
GPQA Diamond	QwQ 32B	4406	1302.6	1115.0	66.0%
GPQA Diamond	QwQ 32B w/ TrimR	3305 (-25.0%)	1185.4 (-9.0%)	1037.0 (-7.0%)	66.0% (0%)

1164 Q TEST-TIME SCALING WITH BoN

1166 Beyond sequential token extension, additional test-time scaling approaches to improve LRM’s accu-
 1167 racy include BoN sampling (Lightman et al., 2024), Monte Carlo Tree Search (MCTS) (Wu et al.,
 1168 2025b), and beam search (Lightman et al., 2024). Integrating TrimR with BoN (N=8), as evidenced
 1169 in Table 9, demonstrates significant reductions in token consumption (-13.8-16.2%), and runtime
 1170 duration (up to 23.3%) while maintaining performance parity (-3.3% on AIME24).
 1171

1172 **Table 9:** Integrating TrimR with Best-of-N (BoN) (Pangu-R-38B) yields comparable efficiency improvements
 1173 and token reduction while preserving the accuracy-performance trade-offs.

	1174 MATH500				1175 AIME24					
	1176 Runtime	1177 TPS	1177 Acc. (%)	1177 #Tokens(M)	1176 Runtime	1177 TPS	1177 Acc. (%)	1177 #Tokens(M)		
BoN	19,423	656	97.4	13.055	35,084	1,628	86.6	18,882		
w/ TrimR	16,171	-16.7%	562 -14.3%	97.2 -0.2	11,251	-13.8%	26,911 -23.3%	1,281 -21.3%	83.3 -3.3%	15,822 -16.2%

1179 R LIMITATIONS

1181 In order to improve the overthinking and underthinking detection accuracy, we introduce an addi-
 1182 tional small 7B verifier model. It may require additional hardware for verifiers during deployment,
 1183 which may increase the complexity of the inference system. However, our algorithm and online
 1184 system design can dramatically reduce the LRM inference cost. In addition, one verifier can serve
 1185 multiple LRMs as discussed in Sec. M, so the additional deployment complexity for small veri-
 1186 fiers is acceptable compared with the runtime reduction of LRMs. In addition, verifiers can also
 1187 be deployed in the same hardware with LRMs and they can be collaborated in the way like serving
 1188 smaller speculative models and larger generation models in speculative decoding.

1188 **S BOARDER IMPACTS**
11891190 Large Reasoning Models are widely used to provide more guidance with internal thinking process
1191 and improve user experience. Our work improves the efficiency of LRM, which is helpful to reduce
1192 the cost and carbon footprint of LRM. This approach can enhance the accessibility of LRM for a
1193 broader population while minimizing environmental impact.1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241