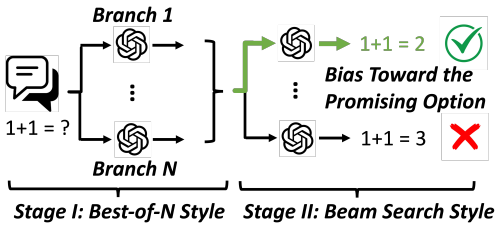
# Think Hard Only When Needed: A Hybrid Best-of-N and Beam Search for Efficient Test-Time Compute

**Hyewon Suh[1], Chaojian Li[1], Cheng-Jhih Shih[1], Zheng Wang[1], Kejing Xia[1], Yonggan Fu[1], Yingyan (Celine) Lin[1],**

[1]Georgia Institute of Technology, USA,
{hsuh45, celine.lin}@gatech.edu

## Abstract

Test-time compute has emerged as a promising paradigm that enables small language models (SLMs) to achieve large language model (LLM)-level capabilities by allocating additional compute for explicit reasoning during inference. Two common approaches are beam search and Best-of-N sampling. Beam search improves reasoning quality by scoring and optimizing token sequences using Process Reward Models (PRMs), but can incur non-trivial computational overhead and latency. In contrast, Best-of-N executes all reasoning trajectories without PRM guidance, often wasting compute on low-quality trajectories that may have gone astray early in the generation process. To address both inefficiencies, we propose THROW (THink haRd Only When needed)—a hybrid inference pipeline that combines the diversity of Best-of-N with the reasoning trajectory optimization of beam search. THROW introduces a selective branch truncation and expansion mechanism: it generates shorter initial trajectories than Best-of-N and evaluates them using PRMs to classify each query as "easy" or "hard". Based on this classification, THROW applies branch truncation for easy queries, mimicking Best-of-N, and PRM-guided branch expansion for hard ones, similar to beam search. Evaluations on MATH500, AMC23, and AIME24 demonstrate that THROW achieves 1.54× and 14.38× latency speedups and 35.7% and 80.4% token reductions on average while preserving high reasoning accuracy compared to Best-of-N and Beam Search, respectively.




(a)

| Type | Efficiency | Branch Diversity | Accuracy |
|---|---|---|---|
| Best-of-N | High | High | Low |
| Beam Search | Low | Low | High |
| Ours | High | High | High |

Goal: Unifying the Strengths of Both Approaches

(b)

Figure 1: (a) Our proposed THROW is a hybrid inference pipeline that combines the diversity of Best-of-N with the targeted trajectory refinement of beam search. (b) As a result, our THROW achieves high efficiency, high branch diversity, and high accuracy.

## 1 Introduction

Large language models (LLMs) exhibit remarkable reasoning and planning capabilities (Wei et al., 2022), yet their substantial inference-time cost significantly impedes deployment in resource-constrained applications such as mobile assistants, embedded robotics, and real-time augmented and virtual reality (AR/VR) systems (Chen et al., 2024). An emerging solution to this challenge is the *test-time compute* (TTC) paradigm—also known as inference-time scaling—in which a smaller language model (SLM) is allocated additional computation during inference to iteratively refine its outputs, rather than relying solely on pre-trained knowledge (Wu et al., 2024; Liu et al., 2025). Analogous to human cognition, where investing more time and effort improves performance on difficult tasks, SLMs granted additional computational iterations can substantially enhance reasoning accuracy on complex queries (Damani et al., 2025). Recent work has shown that strategically allocating TTC enables small models to approach or even surpass the performance of significantly larger models on challenging reasoning tasks (Wu et al., 2024; Liu et al., 2025). This form of TTC inference is particularly critical in resource-constrained settings where

storing or running full-scale LLMs is impractical, yet high-quality language reasoning remains essential (Chen et al., 2024).

Two dominant strategies have emerged in the TTC paradigm: Best-of-N sampling and beam search guided by Process Reward Models (PRMs) (Snell et al., 2025; Wu et al., 2024). Best-of-N sampling generates $N$ independent completions and selects the best output via external evaluation or majority voting (Wu et al., 2024; Manvi et al., 2024). Conversely, PRM-guided beam search incrementally expands only the most promising partial sequences, pruning less likely candidates, and yields notable accuracy improvements (Snell et al., 2025). However, beam search incurs significant computational overhead due to iterative decoding and sequential scoring, and struggles to exploit batch parallelism, resulting in higher latency compared to sampling-based methods (Snell et al., 2025). Practitioners thus face a fundamental trade-off: Best-of-N sampling is parallel but compute-inefficient; beam search improves accuracy at the expense of latency and throughput. Neither paradigm alone fully exploits variability in query complexity, highlighting the need for hybrid, adaptive TTC methods.

A promising yet underexplored approach to addressing this trade-off is the dynamic integration of Best-of-N sampling and PRM-guided beam search within a unified, hybrid inference pipeline. To achieve the aforementioned goal, we propose THROW (THink haRd Only When needed). As shown in Fig. 1, THROW is a new TTC inference framework explicitly designed for streaming, batch-1 scenarios. THROW combines the diversity of Best-of-N sampling with the targeted refinement of PRM-guided beam search. For each query, THROW first generates shorter reasoning trajectories and evaluates intermediate outputs using a PRM to classify the query as “easy” or “hard.” It then truncates low-utility branches early for easy queries to save computation and expands promising ones for hard queries to improve reasoning quality. This adaptive compute allocation not only minimizes unnecessary computation but also improves throughput and parallelism, enabling earlier processing of subsequent streaming inputs and reducing average latency.

Our contributions are as follows:

- **A Hybrid TTC Pipeline Unifying Best-of-N and Beam Search**: We propose THROW, a hybrid inference framework uniquely integrating Best-of-N sampling and PRM-guided beam search to leverage their complementary strengths, diversity and targeted refinement, thereby bridging previously disjoint TTC paradigms.
- **Adaptive Branch Truncation and Expansion**: THROW integrates an effective indicator by first generating shorter initial reasoning trajectories and using PRM-guided scoring to classify queries as “easy" or “hard". It then adaptively truncates or expands branches based on this classification to allocate the computation more effectively.
- **Experiments and ablation studies consistently validate the effectiveness of THROW**: For example, on MATH500, AMC23, and AIME24, THROW achieves 1.54× and 14.38× average wall-clock latency speedups and reduces the number of generated tokens by 35.7% and 80.4% compared to Best-of-N and beam search, respectively. These gains come with only 0.9% accuracy reduction relative to beam search for the Qwen2.5-1.5b-Instruct model, establishing a new state-of-the-art (SOTA) efficiency–quality trade-off for streaming, batch-1 TTC.

## 2 Prior Work

**Process Reward Models in Test Time Compute.** Process Reward Models (PRMs) enhance SLM reasoning by evaluating intermediate reasoning steps rather than only final outputs. Early work by (Lightman et al., 2024) showed that step-level verification substantially improves reasoning accuracy. Subsequent research refined data-construction methods to better capture meaningful intermediate signals (Wang et al., 2024) and improves training reliability using human-preference feedback (Dong et al., 2024). Data quality was shown to be critical for PRM effectiveness (He et al., 2024). Recent advances integrated Monte Carlo estimation with LLM-as-a-Judge evaluations, reducing optimization bias and achieving SOTA results on challenging reasoning benchmarks (Zhang et al., 2025).

**Search Strategies in Test Time Compute.** Effective search is key to generating reasoning trajectories for PRM evaluation. Best-of-N sampling is widely used for its simplicity, parallelization, and strong performance when combined with PRM verification (Snell et al., 2025; Zhang et al.,

2025). Beam search offers a more targeted approach, expanding promising partial solutions iteratively (Snell et al., 2025). Extensions like Diverse Verifier Tree Search (DVTS) improve diversity and accuracy by maintaining multiple exploration paths without major computational cost (Beeching et al.). However, no single search method consistently dominates, motivating adaptive approaches that dynamically allocate compute based on query complexity (Liu et al., 2025).

**Efficient and Adaptive Test Time Compute.** While TTC improves reasoning, it also increases latency and compute cost. Adaptive methods aim to optimize resource use via techniques such as early stopping based on confidence signals (Yang et al., 2025; Huang et al., 2025), confidence-based truncation (Fu et al., 2025), and self-truncating sampling (Wang et al., 2025). Other strategies boost compute for harder queries based on inferred difficulty (Damani et al., 2025; Manvi et al., 2024). Yet, most methods adjust compute in a single direction, limiting flexibility. Addressing this, our THROW framework introduces a hybrid, bidirectional approach that reduces compute for simple queries while expanding it for complex ones, enabling more fine-grained and efficient inference.

## 3 The Proposed THROW Framework

### 3.1 THROW: Overview

Motivated by the complementary strengths and limitations of existing TTC inference pipelines—Best-of-N sampling and PRM-guided beam search—our proposed THROW framework explicitly combines their respective advantages into an unified approach. As shown in Fig. 2, traditional Best-of-N sampling generates multiple reasoning trajectories independently without PRM-based guidance. While highly parallelizable and capable of producing diverse trajectories, this approach often leads to large computational waste due to the exploration of low-quality reasoning paths. Conversely, PRM-guided beam search incrementally evaluates and expands only the most promising reasoning branches, achieving higher reasoning accuracy. However, the iterative decoding and frequent PRM evaluations required by beam search introduce substantial computational overhead and latency.

Our proposed THROW framework resolves these limitations through a two-stage adaptive TTC inference pipeline. Initially, THROW rapidly generates short and diverse reasoning trajectories, akin to Best-of-N sampling. Subsequently, THROW leverages PRM evaluations on intermediate results to explicitly classify queries as either "easy" or "hard". For easy queries, THROW adaptively truncates less-promising branches early, conserving computational resources. For hard queries, THROW selectively expands promising reasoning trajectories through beam search refinement. This adaptive, hybrid pipeline thus enables precise, bidirectional computational budget allocation, improving the achievable trade-off between accuracy and inference efficiency (e.g., latency and throughput).

By coherently integrating these mainstream TTC inference paradigms, THROW improves inference efficiency, latency, and overall hardware utilization, making it particularly suited for real-time, resource-constrained applications where high reasoning accuracy and responsiveness are critical.

### 3.2 THROW: Motivation for Combining Best-of-N Sampling and Beam Search

Before diving into the details of our hybrid pipeline design in the THROW framework, we first explain why combining the strengths of Best-of-N sampling and beam search leads to a more efficient reasoning pipeline. Specifically, Best-of-N sampling defers PRM scoring until the final selection, proceeding through the entire reasoning process without intermediate guidance from PRM. This represents an extreme design that favors high efficiency and diversity but often suffers from lower accuracy due to a lack of targeted refinement. In contrast, beam search frequently consults PRM feedback during generation, guiding the reasoning process toward more promising directions. While this approach achieves higher accuracy, it comes at the cost of reduced efficiency and diversity due to repeated PRM evaluations and narrower exploration. These two extremes, each excelling in different aspects, present an opportunity to strike a balance.

By building a hybrid pipeline that leverages the efficient and diverse trajectory generation of Best-of-N in the first stage and incorporates PRM-guided pruning and refinement in the second stage based on query difficulty, THROW effectively marries the best of both approaches.

### 3.3 THROW: The Proposed Hybrid TTC Pipeline

**Stage I: Difficulty Categorization.** As visualized in Fig. 2(c), when new queries are streamed into

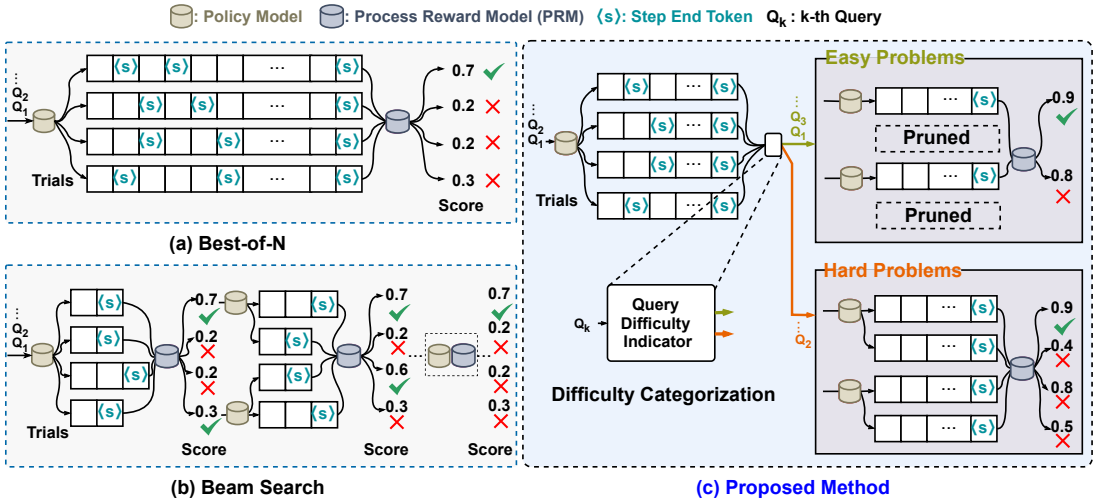



Figure 2: Comparing the reasoning processes of (a) Best-of-N sampling, which completes all reasoning trajectories without PRM guidance; (b) beam search, which selects the most promising trajectories based on PRM scores at each step; and (c) our proposed method, THROW, which begins by mimicking Best-of-N sampling with shorter reasoning trajectories to leverage its diversity, and then classifies queries into "easy" and "hard" categories based on an indicator guided by PRM scores. The "easy" and "hard" queries undergo branch truncation and expansion, respectively, benefiting from the targeted trajectory refinement of beam search.

our proposed THROW framework, Stage I generates initial incomplete trajectories that are shorter than those in standard Best-of-N sampling. These intermediate trajectories serve as partial reasoning results and are used to assess the difficulty of the input queries. Specifically, we employ a PRM to evaluate each trajectory and obtain a PRM score $s_i$ for the $i$-th branch of a given input query. The difficulty of the input query is then quantified as:

$$I = \frac{N - \sum_{i=1}^{N} \mathbf{1}_{\{s_i > v_{th}\}}}{N} \tag{1}$$

where $v_{th}$ is a pre-defined threshold and $\mathbf{1}_{\{s_i > v_{th}\}}$ is an indicator function that returns 1 if the condition $s_i > v_{th}$ is true, and 0 otherwise. A higher value of $I$ suggests that fewer branches are above the threshold, implying a more difficult input query. If $I > t_{th}$, where $t_{th}$ is another threshold, the query is classified as "hard"; otherwise, it is classified as "easy".

**Stage II: Branch Truncation and Expansion.** Thanks to the hybrid design that combines both Best-of-N sampling and beam search, we are able to dynamically adjust the reasoning budget for queries classified as "easy" or "hard" in Stage I. Specifically, for "easy" queries, we reduce the reasoning budget by performing *branch truncation*, where only the top-$K_{\text{easy}}$ branches with the highest PRM scores are retained, and the less promising branches are pruned. In contrast, for "hard" queries, we increase the reasoning budget by performing *branch expansion*, where additional branches are expanded from the top-$K_{\text{hard}}$ most promising trajectories identified in Stage I.

### 3.4 THROW: Theoretical Savings in TTC Inference

To understand under what conditions our proposed THROW is more computationally efficient than baseline methods, we analytically formulate its adaptive theoretical computation cost required to solve a single query as follows:

$$\begin{aligned} C_{\text{THROW}} = C_{\text{policy}} \frac{L_{\text{Stage\_I}}}{L} N + C_{\text{PRM}} N \\ + C_{\text{policy}} \frac{L_{\text{Stage\_II}}}{L} \left(K_{\text{easy}} + b K_{\text{hard}}\right) \end{aligned} \tag{2}$$

where $C_{\text{policy}}$ is the computation cost of executing the policy model for one branch with total trajectory length $L$. $L_{\text{Stage\_I}}$ and $L_{\text{Stage\_II}}$ are the length of reasoning steps in Stage I and Stage II of THROW, respectively, satisfying $L_{\text{Stage\_I}} + L_{\text{Stage\_II}} = L$. $C_{\text{PRM}}$ is the computation cost of running the PRM model once $K_{\text{easy}}$ and $K_{\text{hard}}$ are the numbers of branches continued in Stage II for "easy" and "hard" queries, respectively. Finally, $b$ is the ratio of branch expansion in Stage II for "hard" queries. Unlike the baselines, THROW adapts its

compute based on query difficulty. Specifically, the parameters in Eq. 2 take the following values:

$$(K_{\text{easy}}, b, K_{\text{hard}}) = \begin{cases} (K_{\text{easy}}, 0, 0) & \text{if easy} \\ (0, b, K_{\text{hard}}) & \text{if hard} \end{cases} \quad (3)$$

In contrast, the theoretical computation cost for the baseline Best-of-N (BoN) and beam search (BS) in TTC is:

$$C_{\text{BoN}} = C_{\text{policy}} \cdot N + C_{\text{PRM}} \cdot N \quad (4)$$

$$C_{\text{BS}} = C_{\text{policy}} \cdot N + S \cdot C_{\text{PRM}} \cdot N \quad (5)$$

where $S$ is the number of times the PRM model is invoked during the beam search reasoning process. The resulting computation saved by our proposed THROW is:

$$C_{\text{BS}} - C_{\text{THROW}} = (S-1)C_{\text{PRM}}N + C_{\text{policy}}\frac{L_{\text{Stage\_II}}}{L}\Big(N - (K_{\text{easy}} + bK_{\text{hard}})\Big) \quad (6)$$

$$C_{\text{BoN}} - C_{\text{THROW}} = C_{\text{policy}}\frac{L_{\text{Stage\_II}}}{L}\Big(N - (K_{\text{easy}} + bK_{\text{hard}})\Big) \quad (7)$$

To ensure a fair comparison in terms of total reasoning steps, if we assume $L_{\text{Stage\_I}} = L_{\text{Stage\_II}} = \frac{L}{2}$, under this setting, as long as $(K_{\text{easy}} + b \cdot K_{\text{hard}}) < N$, our method requires fewer policy model inferences than both baselines. In the case of beam search, this advantage is further amplified by the additional savings associated with reduced PRM invocations, captured by the term $(S-1) \cdot C_{\text{PRM}} \cdot N$.

Furthermore, in our experimental configuration we adopt the constraints $K_{\text{easy}} \leq \frac{N}{2}$ and $b \cdot K_{\text{hard}} \leq \frac{N}{2}$. These conditions imply $(K_{\text{easy}} + b \cdot K_{\text{hard}}) \leq \frac{N}{2}$, which allows us to derive lower bounds on the expected computational savings achieved by THROW:

$$C_{\text{BS}} - C_{\text{THROW}} \geq C_{\text{policy}} \cdot \frac{L_{\text{Stage\_II}}}{L} \cdot \frac{N}{2} + (S-1) \cdot C_{\text{PRM}} \cdot N \quad (8)$$

$$C_{\text{BoN}} - C_{\text{THROW}} \geq C_{\text{policy}} \cdot \frac{L_{\text{Stage\_II}}}{L} \cdot \frac{N}{2} \quad (9)$$

These results provide a formal theoretical guarantee for the computational efficiency of THROW.

# 4 Experiments

## 4.1 Experiment Setup

**Dataset and Models** We utilize a selection of representative mathematical datasets, including MATH500 (Lightman et al., 2024), AMC23 (math-ai, 2025) and AIME24 (AI-MO, 2024), to comprehensively evaluate our approach. For the PRM, we employ the Qwen2.5-Math-PRM-7B, a SOTA open-source PRM, alongside instruction-tuned variants Qwen2.5-1.5B-Instruct and Qwen2.5-3B-Instruct (Qwen et al., 2025) as policy models.

**Baselines and Implementation** We conduct all main experiments on a single NVIDIA A100-SXM4-40GB GPU (hereafter referred to as the A100 GPU) to enable an apples-to-apples comparison of THROW, PRM-guided beam search, and Best-of-N sampling under identical conditions. To further evaluate our approach in a more resource-constrained environment, we additionally report results on an NVIDIA RTX A5000 GPU (hereafter referred to as the A5000 GPU) in Appendix C.

We tune the hyperparameters of each policy model via grid search over $t_{\text{th}}$, $K_{\text{easy}}$, $K_{\text{hard}}$, and $b$. The first-stage generation length $L_{\text{Stage\_I}}$ is fixed at 350 tokens across all datasets and policy models. Because dataset difficulty varies substantially, we use different hyperparameter configurations for the general-purpose MATH500 dataset and for the more challenging AMC23 and AIME24 benchmarks. All generations use a maximum output length of 2048 tokens and a temperature of 0.7. Full hyperparameter settings and additional implementation details are provided in Appendix A.

## 4.2 Benchmark with SOTA pipelines

We benchmark THROW against SOTA pipelines on Qwen2.5-1.5B-Instruct and Qwen2.5-3B-Instruct policy models shown in Table 1 and Table 2, respectively. We present the main results with $N = 16$ in the main text and include the full results of $N = 8, 16, 32$ values in Appendix B. We observe that THROW achieves 1.54× and 14.38× average latency speedups, along with 35.7% and 80.4% average token reductions compared to Best-of-N and beam search, respectively, across multiple trajectory counts ($N = 8, 16, 32$) for the Qwen2.5-1.5B-Instruct model. These efficiency gains are attained while delivering a 1.1% accuracy improvement over Best-of-N and incurring only a 0.9% accuracy reduction relative to the compute-intensive beam search baseline. Similarly, for the Qwen2.5-3B-Instruct model, THROW achieves 1.51× and 11.86× latency speedups and reduces token generation by 37.1% and 79.3% compared to Best-of-N and beam search, respectively. Moreover, THROW improves accuracy by an average 2.4% over Best-of-N while remaining within 2.2% of beam search, demonstrating a consistently favorable efficiency-quality trade-off across model scales. By truncating

Table 1: Comparison of THROW with beam search and Best-of-N baselines on the Qwen2.5-1.5B-Instruct model with $N = 16$. The final row shows THROW's relative improvements over each baseline, with higher values indicating greater token savings and faster inference speedups, reported as Beam Search / Best-of-N.

| Method | MATH500 | | | AMC23 | | | AIME24 | | |
|---|---|---|---|---|---|---|---|---|---|
| | Lat. | Tok. | Acc. | Lat. | Tok. | Acc. | Lat. | Tok. | Acc. |
| Beam Search | 52.02 | 25469 | 0.73 | 84.55 | 44228 | 0.52 | 112.17 | 57973 | 0.06 |
| Best-of-N | 7.08 | 9574 | 0.73 | 9.30 | 12419 | 0.46 | 10.78 | 14916 | 0.09 |
| **THROW** | **5.33** | **7405** | 0.73 | **5.94** | **7577** | 0.50 | **7.31** | **8881** | 0.13 |
| | (×9.8 / ×1.33) | (70.9% / 22.7%) | - | (×14.2 / ×1.56) | (82.9% / 38.9%) | - | (×15.3 / ×1.47) | (84.7% / 40.5%) | - |

Table 2: Comparison of THROW with beam search and Best-of-N baselines on the Qwen2.5-3B-Instruct model with $N = 16$. The final row shows THROW's relative improvements over each baseline, with higher values indicating greater token savings and faster inference speedups, reported as Beam Search / Best-of-N.

| Method | MATH500 | | | AMC23 | | | AIME24 | | |
|---|---|---|---|---|---|---|---|---|---|
| | Lat. | Tok. | Acc. | Lat. | Tok. | Acc. | Lat. | Tok. | Acc. |
| Beam Search | 54.58 | 29849 | 0.79 | 97.75 | 41275 | 0.65 | 136.71 | 56290 | 0.20 |
| Best-of-N | 8.86 | 9557 | 0.79 | 12.67 | 13170 | 0.49 | 14.99 | 15323 | 0.16 |
| **THROW** | **7.08** | **7606** | 0.79 | **7.67** | **7467** | 0.53 | **8.48** | **8014** | 0.17 |
| | (×7.7 / ×1.25) | (74.5% / 20.4%) | - | (×12.7 / ×1.65) | (81.9% / 43.3%) | - | (×16.1 / ×1.77) | (85.8% / 47.7%) | - |

Table 3: Token usage and latency breakdown on the MATH500 dataset using the Qwen2.5-1.5B-Instruct model. Results compare the THROW framework against the Best-of-N baseline, illustrating differences in computational cost and inference time.

| **Method** | **THROW** | **Best-of-N** |
|---|---|---|
| **Avg. tokens** | **6728** | **9541** |
| **Avg. latency (s)** | **8.33 (100.0%)** | **10.77 (100.0%)** |
| Policy Model | 6.19 (74.3%) | 8.31 (77.1%) |
| PRM | 2.14 (25.7%) | 2.46 (22.9%) |

the second-stage generation to $N/2$ for both easy and hard queries, THROW reuses idle GPU memory to start processing subsequent queries earlier, thereby improving resource utilization and overall latency. Meanwhile, selective branch expansion on hard queries preserves accuracy by focusing compute on the most promising reasoning paths. These gains are especially pronounced on challenging AMC23 and AIME24 benchmarks, where policy models otherwise struggle to reason effectively.

These results highlight THROW's strong generalization across both Qwen2.5-1.5B-Instruct and Qwen2.5-3B-Instruct models, consistently achieving comparable or superior accuracy to baselines with significantly lower token usage and latency. Notably, these efficiency gains align with our theoretical predictions in Sec. 3.4. By reallocating computation adaptively, THROW remains robust even on challenging benchmarks like AMC23 and AIME24, underscoring its effectiveness across diverse task difficulties and model capacities.

### 4.3 Analysis

**Decomposing Latency and Token Savings.** To further understand where these improvements come from, Table 3 break down the sources of efficiency gains. Token savings primarily arise from trial pruning based on the query difficulty indicator, which filters out unpromising trajectories early. On the latency side, Best-of-N is dominated by policy generation time, whereas THROW reduces this cost by pruning low-value trajectories and streaming inputs to better reuse model states. The indicator-based pruning accounts for the majority of token savings across all settings, particularly on harder queries.

**Token Inefficiency in Baseline Methods.** An additional source of THROW's efficiency gains lies in the large amount of redundant token generation in existing decoding pipelines. As shown in Table 1 and Table 2, PRM-guided beam search generates many unnecessary tokens at each decoding step, especially on more difficult tasks such as AMC23 and AIME24, where longer reasoning chains exacerbate this inefficiency. Although Best-of-N sampling avoids generating wasteful tokens like THROW, its trajectory quality is limited due to the lack of PRM evaluation. As a result, it achieves 1.1% and 2.4% lower average accuracy compared to THROW for the Qwen2.5-1.5B-Instruct and Qwen2.5-3B-Instruct models, respectively.

Table 4: Results on math benchmarks using the Llama3.1-8B-PRM-Deepseek-Data model ($N = 16$). Token reduction, latency speedup, and accuracy difference are reported for THROW relative to Best-of-N and Beam Search.

| Benchmark | THROW vs. Best-of-N | | | THROW vs. Beam Search | | |
|---|---|---|---|---|---|---|
| | Tok. Red. ↑ (%) | Lat. Speedup ↑ (×) | Acc. Diff ↑ | Tok. Red. ↑ (%) | Lat. Speedup ↑ (×) | Acc. Diff ↑ |
| MATH | 39.3 | 1.53 | +0.010 | 79.5 | 14.10 | +0.018 |
| AMC23 | 51.0 | 1.88 | +0.125 | 84.5 | 18.75 | +0.125 |
| AIME24 | 54.5 | 1.93 | +0.067 | 86.7 | 24.21 | +0.000 |
| **Avg** | **48.3** | **1.78** | **+0.067** | **83.6** | **19.02** | **+0.048** |

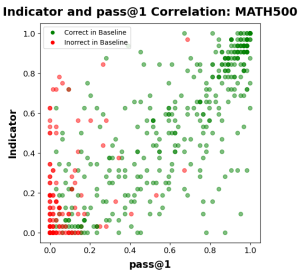
Table 5: Results on math benchmarks using the LLaMA3.2-3B-Instruct model ($N = 16$). Token reduction, latency speedup, and accuracy difference are reported for THROW relative to Best-of-N and Beam Search.

| Benchmark | THROW vs. Best-of-N | | | THROW vs. Beam Search | | |
|---|---|---|---|---|---|---|
| | Tok. Red. ↑ (%) | Lat. Speedup ↑ (×) | Acc. Diff ↑ | Tok. Red. ↑ (%) | Lat. Speedup ↑ (×) | Acc. Diff ↑ |
| MATH | 33.7 | 1.48 | −0.006 | 83.1 | 11.82 | −0.024 |
| AMC23 | 35.4 | 1.37 | +0.000 | 84.1 | 11.84 | −0.075 |
| AIME24 | 35.5 | 1.37 | +0.000 | 86.7 | 15.11 | +0.067 |
| **Avg** | **34.9** | **1.41** | **−0.002** | **84.6** | **12.92** | **−0.011** |

## 4.4 Ablation Studies

### 4.4.1 Generalization Across PRMs and Policy Models

To examine the robustness and versatility of THROW, we evaluate its performance using alternative PRMs and policy models across $N = 8, 16, 32$, with the complete results provided in Appendix D. First, we substitute the PRM with LLaMA3.1-8B-PRM-DeepSeek-Data (Xiong et al., 2024) with the Qwen2.5-1.5B-Instruct as the policy model to further test THROW's robustness to reward model choice. Despite this PRM being weaker than the Qwen-based one, THROW still achieves 1.78× and 19.02× speedups, along with 6.7% and 4.8% accuracy improvements over Best-of-N and beam search, respectively, as shown in Table 4. Across all $N$, THROW maintains an average of 1.77× and 19.62× speedups with 4.3% and 2.2% average accuracy improvements. These results highlights that THROW remains effective even when the trajectory evaluation signal is noisier, and can even enhance accuracy, indicating that its improvements stem from the underlying algorithmic design rather than dependence on a specific model family. Next, we replace the original Qwen policy with LLaMA3.2-3B-Instruct (Grattafiori et al., 2024) and observe that THROW continues to deliver substantial efficiency gains, achieving 1.39× and 12.67× latency speedups over Best-of-N and beam search, respectively, while maintaining comparable accuracy, as shown in Table 5.




| Benchmark | Pearson $\lvert r \rvert$ |
|---|---|
| MATH500 | 0.801 |
| AMC23 | 0.804 |
| AIME24 | 0.679 |

Figure 3: Correlation between the proposed indicator $I$ and *pass@1* accuracy on the MATH500 dataset using Qwen2.5-1.5B ($N = 32$) (left), along with Pearson correlation coefficients across benchmarks (right), indicating strong predictive power of the indicator.

### 4.4.2 Ablation Studies of the Proposed Hybrid TTC Pipeline

We conduct a series of ablation experiments to better understand the design choices and key components of THROW. All experiments are performed using the Qwen2.5-1.5B-Instruct model on the MATH500 benchmark, which serves as a representative setting for the overall trends observed across models and tasks. This choice enables us to isolate the contribution of each component while keeping the analysis clear and computationally efficient.

**Stage I: Evaluating the Query Difficulty Indicator.** We first assess the effectiveness of the proposed query difficulty indicator $I$ (Eq. 1) in capturing true problem difficulty. As shown in Fig. 3, the indicator exhibits a strong linear correlation with the oracle difficulty signal, measured by the $pass@1$ accuracy. Interestingly, problems associ-

Table 6: Accuracy comparison on the MATH500 dataset for random selection versus THROW using the Qwen2.5-1.5B-Instruct model. Numbers in parentheses indicate absolute accuracy improvements over random selection.

| Method | N=8 | N=16 | N=32 |
|---|---|---|---|
| Random | 0.604 | 0.672 | 0.706 |
| **THROW** | **0.674**$_{(+0.07)}$ | **0.722**$_{(+0.05)}$ | **0.742**$_{(+0.04)}$ |

ated with false-positive or false-negative indicator values tend to be misclassified by the baseline Best-of-N approach as well. This observation suggests that deviations from the linear trend do not significantly impact the overall performance of our pipeline, underscoring its robustness even under indicator noise. Also, Fig. 3 reports the Pearson correlation between $I$ and the oracle $pass@1$ accuracy across benchmarks. We observe strong absolute correlations ranging from 0.679 to 0.801, demonstrating that $I$ reliably estimates query difficulty without requiring full trajectory generation. This enables the system to adaptively allocate computation based on problem complexity. The slightly lower correlation on AIME24 is primarily due to its challenging nature, as many queries are unsolvable ($pass@1 = 0$) but still receive non-zero $I$ scores. Regardless, $I$ remains a useful signal for guiding downstream decisions.

**Effect of Stage I Reasoning Length.** We next investigate the effect of reasoning length $L_{\text{Stage\_I}}$, a key hyperparameter that governs the trade-off between accuracy and computational cost. Fig. 4 shows results for five settings ($L_{\text{Stage\_I}} \in \{150, 250, 350, 450, 550\}$, labeled [1–5]), where shorter reasoning windows reduce token usage but may slightly degrade accuracy. Conversely, longer reasoning windows improve or stabilize accuracy at the cost of higher token consumption. This trade-off is more pronounced when partial trajectories are limited, as difficulty estimates become less reliable. Overall, a moderate reasoning length offers the best balance between efficiency and performance, guiding our default choice in all main experiments.

**Stage II: Effectiveness of Adaptive Branch Selection.** Finally, we investigate the impact of our proposed trajectory selection mechanism in Stage II (Sec. 3.3) by comparing THROW against a random selection baseline. As shown in Table 6, THROW achieves substantially higher accuracy, outperforming random selection by 5.2% on Qwen2.5-1.5B-Instruct for the MATH500 benchmark. This performance gap highlights the effectiveness of our selective refinement strategy, which focuses computation on the most promising reasoning paths and plays a critical role in maintaining accuracy while reducing overall computational cost.

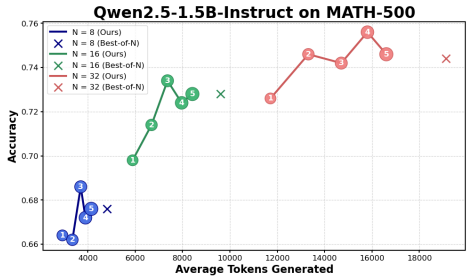



Figure 4: Accuracy versus average token usage per query on the MATH500 dataset under different Stage I trajectory lengths ($L_{\text{Stage\_I}}$).

## 5 Conclusion

In this work, we present THROW, a novel hybrid inference framework explicitly designed for efficient and adaptive TTC in streaming, batch-1 scenarios. THROW uniquely integrates the complementary strengths of Best-of-N sampling and PRM-guided beam search, enabling nuanced bidirectional adaptivity by dynamically reducing computation for simpler queries and selectively increasing it for more challenging ones. Through a novel query-classification mechanism leveraging intermediate PRM evaluations, THROW effectively allocates computational resources tailored explicitly to query complexity. Extensive evaluations and ablation studies on representative reasoning benchmarks, including MATH500, AMC23, and AIME24, demonstrate that THROW achieves 1.54× and 14.38× latency speedups compared to Best-of-N and beam search baselines respectively, all while maintaining comparable reasoning accuracy. By bridging two previously disjoint TTC paradigms, THROW sets a new SOTA in balancing efficiency and reasoning quality, underscoring the substantial promise of hybrid adaptive approaches for real-time, resource-constrained TTC inference tasks.

## 6 Limitations

While the proposed THROW inference pipeline enables adaptive reasoning based on input query difficulty and achieves a superior accuracy–efficiency trade-off compared to Best-of-N and beam search (as shown in Sec. 4.2), it has several limitations, as outlined below:

- **Verifier-Specific Design:** THROW is specifically designed for the verifier-based TTC framework (i.e., those relying on PRMs). Its effectiveness in non-verifier settings remains an open question and a potential direction for future work.

- **Dependence on PRM Signals:** Although THROW demonstrates strong robustness even under weaker reward models (Appendix D), its decision-making pipeline still relies on step-wise PRM scores to estimate query difficulty and guide branch truncation or expansion. Consequently, systematic PRM biases or severely miscalibrated scores may affect the quality of difficulty estimation, particularly in domains where reliable process supervision is unavailable.

Despite these limitations, THROW demonstrates strong practical benefits in verifier-based TTC settings, particularly in low-memory environments. Future work may explore extensions to non-verifier inference paradigms and improved methods for uncertainty-aware or calibration-robust difficulty estimation.

## Acknowledgments

This article is based upon work supported by the National Science Foundation (NSF) (Award IDs: 2312758, 2434166, 2048183, 2016727, 2400511, 2403297), the Department of Health and Human Services Advanced Research Projects Agency for Health (ARPA-H) under Award Number AY1AX 000003 and Agreement Number 140D042490003, and CoCoSys, one of the seven centers in JUMP 2.0, a Semiconductor Research Corporation (SRC) program sponsored by DARPA. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied of the Advanced Research Projects Agency Health or the U.S. Government.

## References

AI-MO. 2024. AIME 2024 dataset. https://huggingface.co/datasets/AI-MO/aimo-validation-aime.

Edward Beeching, Lewis Tunstall, and Sasha Rush. Scaling test-time compute with open models.

Wei Chen, Zhiyuan Li, Zhen Guo, and Yikang Shen. 2024. Octo-planner: On-device language model for planner-action agents. *arXiv preprint arXiv:2406.18082*.

Mehul Damani, Idan Shenfeld, Andi Peng, Andreea Bobu, and Jacob Andreas. 2025. Learning how hard to think: Input-adaptive allocation of LM computation. In *The Thirteenth International Conference on Learning Representations*.

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

Yichao Fu, Junda Chen, Yonghao Zhuang, Zheyu Fu, Ion Stoica, and Hao Zhang. 2025. Reasoning without self-doubt: More efficient chain-of-thought through certainty probing. In *ICLR 2025 Workshop on Foundation Models in the Wild*.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, and 542 others. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Jujie He, Tianwen Wei, Rui Yan, Jiacai Liu, Chaojie Wang, Yimeng Gan, Shiwen Tu, Chris Yuhao Liu, Liang Zeng, Xiaokun Wang, Boyang Wang, Yongcong Li, Fuxiang Zhang, Jiacheng Xu, Bo An, Yang Liu, and Yahui Zhou. 2024. Skywork-o1 open series.

Chengsong Huang, Langlin Huang, Jixuan Leng, Jiacheng Liu, and Jiaxin Huang. 2025. Efficient test-time scaling via self-calibration. *Preprint*, arXiv:2503.00031.

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

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

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

math-ai. 2025. AMC 2023 dataset. https://huggingface.co/datasets/math-ai/amc23.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan

Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, and 25 others. 2025. Qwen2.5 technical report. *Preprint*, arXiv:2412.15115.

Charlie Victor Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2025. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *The Thirteenth International Conference on Learning Representations*.

Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. 2024. Math-shepherd: Verify and reinforce LLMs step-by-step without human annotations. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9426–9439, Bangkok, Thailand. Association for Computational Linguistics.

Yiming Wang, Pei Zhang, Siyuan Huang, Baosong Yang, Zhuosheng Zhang, Fei Huang, and Rui Wang. 2025. Sampling-efficient test-time scaling: Self-estimating the best-of-n sampling in early decoding. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. 2022. Chain of thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*.

Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. 2024. Scaling inference computation: Compute-optimal inference for problem-solving with language models. In *The 4th Workshop on Mathematical Reasoning and AI at NeurIPS'24*.

Wei Xiong, Hanning Zhang, Nan Jiang, and Tong Zhang. 2024. An implementation of generative prm. `https://huggingface.co/RLHFlow/Llama3.1-8B-PRM-Deepseek-Data`.

Chenxu Yang, Qingyi Si, Yongjie Duan, Zheliang Zhu, Chenyu Zhu, Qiaowei Li, Minghui Chen, Zheng Lin, and Weiping Wang. 2025. Dynamic early exit in reasoning models. *Preprint*, arXiv:2504.15895.

Zhenru Zhang, Chujie Zheng, Yangzhen Wu, Beichen Zhang, Runji Lin, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. 2025. The lessons of developing process reward models in mathematical reasoning. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 10495–10516, Vienna, Austria. Association for Computational Linguistics.

## A Implementation Details and Hyperparameter Settings of THROW

Following the state-of-the-art implementation (Liu et al., 2025), we employ the `vLLM` library for policy model inference (decoding) and the `transformers` library for PRM inference (prefilling), all on a single GPU (A100 or A5000). To avoid redundant PRM scoring of intermediate results in THROW, we reuse the PRM KV cache for previously computed results. For all methods, step-wise PRM scores are aggregated by taking their product, and the trajectory with the highest aggregate score is selected as the final prediction. The threshold hyperparameter $v_{th} = 0.9$ is fixed based on the score distribution of the Qwen2.5-Math-PRM-7B model. We report the average over 3 runs.

For hyperparameter tuning, we perform a grid search and select the configuration that minimizes computation while maintaining accuracy comparable to or better than the baselines. Specifically, we search over the following candidate values: $t_{th} \in \{0.5,\ 0.75,\ 0.875,\ 1.0\}$, $K_{\text{easy}} \in \left\{\frac{N}{2},\ \frac{N}{4},\ \frac{N}{8},\ 1\right\}$, $K_{\text{hard}} \in \left\{\frac{N}{2},\ \frac{N}{4},\ \frac{N}{8},\ 1\right\}$, and $b \in \{1,\ 2,\ 4\}$. Table A-1 lists the searched optimal hyperparameter values for each benchmark setting. Here, $t_{th} = 1$ implies that a query is categorized as "hard" if all trajectories receive low PRM scores ($I \geq t_{th}$). Specifically, the grid search indicates that the selected $K_{\text{hard}}$ values for AMC23 and AIME24 are generally smaller than those chosen for MATH500 (e.g., $\frac{N}{8} \sim \frac{N}{4}$ vs. $\frac{N}{4} \sim \frac{N}{2}$). We conjecture that this is because most queries in AMC-2023 and AIME-2024 are more challenging, often yielding low-scoring partial trajectories. As a result, THROW allocates computation to only a few promising candidates, rather than spreading resources across many unpromising ones.

## B Comprehensive Results of Comparison on A100

We report the results of THROW compared to Best-of-N and beam search baseline on A100 GPU across all trajectory counts ($N = 8, 16, 32$).

**Experiments with Qwen2.5-1.5B-Instruct as the Policy Model.** As shown in Table B-2, THROW achieves average latency speedups of 1.54× and 14.38× compared to Best-of-N and PRM-guided beam search baselines. These improvements are attributed to the corresponding average token reductions of 35.7% and 80.4%, respectively, for processing each query.

**Experiments with Qwen2.5-3B-Instruct as the Policy Model.** As shown in Table B-3, THROW achieves average latency speedups of 1.51× and 11.86× compared to Best-of-N and PRM-guided beam search baselines. These improvements are attributed to the corresponding average token reductions of 37.1% and 79.3%, respectively, for processing each query.

## C Comprehensive Results of Comparison on A5000

In addition to evaluating THROW, PRM-guided beam search, and Best-of-N sampling on the A100 GPU, we further benchmark these methods on an A5000 GPU to emulate a more resource-constrained deployment setting. Because beam search exceeds the memory capacity of the A5000, we report results only for Best-of-N and THROW under this configuration. Across all benchmarks, THROW consistently delivers significant latency speedups while maintaining accuracy comparable to the Best-of-N baseline, even under limited GPU memory conditions.

**Experiments with Qwen2.5-1.5B-Instruct as the Policy Model.** As shown in Table C-4, THROW achieves an average latency speedup of 1.56× with 42.2% token reduction compared to Best-of-N, while maintaining comparable accuracy.

**Experiments with Qwen2.5-3B-Instruct as the Policy Model.** As shown in Table C-4, THROW achieves an average latency speedup of 1.28× compared to Best-of-N with comparable accuracy. These improvements are attributed to the corresponding average token reduction of 37.9% for processing each query.

## D Comprehensive Results of LLaMA PRM and Policy Model

We report the results of THROW compared to Best-of-N and beam search baseline on A100 GPU across all trajectory counts ($N = 8, 16, 32$) for generalization across LLaMA PRM and policy model.

**Experiments with LLaMA3.1-8B-PRM-DeepSeek-Data as the PRM.** As shown in Table D-5, THROW achieves average latency speedups of 1.77× and 19.62× compared to Best-of-N and PRM-guided beam search baselines. These improvements are attributed to the corresponding average token reductions of 47.2% and 82.4%, respectively, for processing each query. Notably, the use of a weaker PRM even results

Table A-1: Hyperparameter sets $(t_{th}, K_{\text{easy}}, K_{\text{hard}}, b)$ used for different policy models and benchmark difficulties

| Benchmark | Qwen2.5-1.5B-Instruct | | | Qwen2.5-3B-Instruct | | |
|---|---|---|---|---|---|---|
| | N=8 | N=16 | N=32 | N=8 | N=16 | N=32 |
| MATH-500 | (0.875,4,2,2) | (0.875,8,4,2) | (0.875,16,8,2) | (1.0,4,4,1) | (1.0,8,8,1) | (0.75,16,4,4) |
| AMC-2023, AIME-2024 | (0.875,4,1,2) | (0.75,4,8,1) | (1.0,4,4,2) | (0.5,4,1,2) | (1.0,4,1,2) | (1.0,8,8,2) |

in accuracy improvements over both baselines, highlighting THROW's ability to effectively balance PRM guidance and maintain performance even when reward signals are noisy.

**Experiments with LLaMA3.2-3B-Instruct as the Policy Model.** As shown in Table D-6, THROW achieves average latency speedups of 1.39× and 12.67× compared to Best-of-N and PRM-guided beam search baselines. These improvements are attributed to the corresponding average token reductions of 34.8% and 84.0%, respectively, for processing each query.

Table B-2: Results on math benchmarks using the Qwen2.5-1.5B-Instruct model on an A100 GPU. Token reduction, latency speedup, and accuracy difference are reported for THROW relative to Best-of-N (BoN) and Beam Search (BS) across different numbers of trajectories ($N = 8, 16, 32$).

| **Benchmark** | **N** | **THROW vs. BoN** | | | **THROW vs. BS** | | |
|---|---|---|---|---|---|---|---|
| | | **Tok. Red. ↑ (%)** | **Lat. Speedup ↑ (×)** | **Acc. Diff ↑** | **Tok. Red. ↑ (%)** | **Lat. Speedup ↑ (×)** | **Acc. Diff ↑** |
| MATH | 8 | 21.9 | 1.32 | -0.010 | 71.4 | 6.86 | -0.030 |
| | 16 | 22.7 | 1.33 | +0.000 | 70.9 | 9.76 | +0.000 |
| | 32 | 22.1 | 1.23 | +0.000 | 78.2 | 14.37 | +0.010 |
| AMC23 | 8 | 33.5 | 1.68 | +0.050 | 79.2 | 9.93 | +0.010 |
| | 16 | 39.0 | 1.57 | +0.040 | 82.9 | 14.23 | -0.020 |
| | 32 | 49.4 | 1.93 | +0.030 | 86.3 | 23.18 | -0.060 |
| AIME24 | 8 | 37.1 | 1.46 | -0.020 | 81.2 | 9.91 | -0.030 |
| | 16 | 40.5 | 1.47 | +0.040 | 84.7 | 15.35 | +0.070 |
| | 32 | 55.0 | 1.82 | -0.030 | 88.6 | 25.83 | -0.030 |
| **Avg** | — | **35.7** | **1.54** | **+0.011** | **80.4** | **14.38** | **-0.009** |

Table B-3: Results on math benchmarks using the Qwen2.5-3B-Instruct model on an A100 GPU. Token reduction, latency speedup, and accuracy difference are reported for THROW relative to Best-of-N (BoN) and Beam Search (BS) across different numbers of trajectories ($N = 8, 16, 32$).

| **Benchmark** | **N** | **THROW vs. BoN** | | | **THROW vs. BS** | | |
|---|---|---|---|---|---|---|---|
| | | **Tok. Red. ↑ (%)** | **Lat. Speedup ↑ (×)** | **Acc. Diff ↑** | **Tok. Red. ↑ (%)** | **Lat. Speedup ↑ (×)** | **Acc. Diff ↑** |
| MATH | 8 | 20.7 | 1.23 | -0.010 | 66.5 | 5.23 | -0.010 |
| | 16 | 20.4 | 1.25 | +0.000 | 74.5 | 7.71 | +0.000 |
| | 32 | 20.7 | 1.24 | +0.000 | 72.6 | 10.18 | +0.000 |
| AMC23 | 8 | 43.5 | 1.65 | +0.100 | 78.9 | 7.93 | -0.010 |
| | 16 | 43.3 | 1.65 | +0.040 | 81.9 | 12.74 | -0.120 |
| | 32 | 43.0 | 1.58 | +0.040 | 83.4 | 16.52 | -0.030 |
| AIME24 | 8 | 48.6 | 1.70 | +0.010 | 83.6 | 10.25 | +0.000 |
| | 16 | 47.7 | 1.77 | +0.010 | 85.8 | 16.12 | -0.030 |
| | 32 | 46.0 | 1.53 | +0.030 | 86.5 | 20.04 | +0.000 |
| **Avg** | — | **37.1** | **1.51** | **+0.024** | **79.3** | **11.86** | **-0.022** |

Table C-4: Comparison of THROW vs. Best-of-N on math benchmarks using Qwen2.5-1.5B and Qwen2.5-3B models on an A5000 GPU. Token reduction, latency speedup, and accuracy difference are reported relative to Best-of-N across different numbers of trajectories ($N = 8, 16, 32$).

**Qwen2.5-1.5B-Instruct**

| **Benchmark** | **N** | **THROW vs. BoN** | | |
|---|---|---|---|---|
| | | Tok. Red. ↑ | Spd. ↑ | Acc. Diff ↑ |
| MATH500 | 8 | 23.8 | 1.35 | -0.010 |
| | 16 | 24.0 | 1.29 | +0.000 |
| | 32 | 30.5 | 1.26 | +0.000 |
| AMC23 | 8 | 45.0 | 1.55 | +0.050 |
| | 16 | 48.6 | 1.57 | +0.040 |
| | 32 | 45.6 | 2.06 | +0.030 |
| AIME24 | 8 | 57.9 | 1.46 | -0.020 |
| | 16 | 55.4 | 1.59 | +0.040 |
| | 32 | 48.7 | 1.86 | -0.030 |
| **Avg** | — | **42.2** | **1.56** | **+0.011** |

**Qwen2.5-3B-Instruct**

| **Benchmark** | **N** | **THROW vs. BoN** | | |
|---|---|---|---|---|
| | | Tok. Red. ↑ | Spd. ↑ | Acc. Diff↑ |
| MATH500 | 8 | 23.4 | 1.14 | -0.010 |
| | 16 | 23.2 | 1.13 | +0.000 |
| | 32 | 23.2 | 1.10 | +0.000 |
| AMC23 | 8 | 40.0 | 1.21 | +0.100 |
| | 16 | 33.1 | 1.43 | +0.040 |
| | 32 | 51.4 | 1.52 | +0.040 |
| AIME24 | 8 | 52.0 | 1.34 | +0.010 |
| | 16 | 31.3 | 1.22 | +0.010 |
| | 32 | 58.2 | 1.40 | +0.030 |
| **Avg** | — | **37.9** | **1.28** | **+0.025** |

Table D-5: Results on math benchmarks using the Qwen2.5-1.5B-Instruct model with LLaMA3.1-8B-PRM-DeepSeek-Data. Token reduction, latency speedup, and accuracy difference are reported for THROW relative to Best-of-N (BoN) and Beam Search (BS) across different numbers of trajectories ($N = 8, 16, 32$).

| **Benchmark** | **N** | **THROW vs. BoN** | | | **THROW vs. BS** | | |
|---|---|---|---|---|---|---|---|
| | | **Tok. Red. ↑ (%)** | **Lat. Speedup ↑ (×)** | **Acc. Diff ↑** | **Tok. Red. ↑ (%)** | **Lat. Speedup ↑ (×)** | **Acc. Diff ↑** |
| MATH | 8 | 38.2 | 1.47 | -0.008 | 75.8 | 8.66 | -0.008 |
| | 16 | 39.3 | 1.53 | +0.010 | 79.5 | 14.10 | +0.018 |
| | 32 | 48.5 | 1.79 | -0.016 | 84.2 | 22.74 | +0.018 |
| AMC23 | 8 | 35.2 | 1.62 | +0.150 | 74.6 | 8.69 | +0.050 |
| | 16 | 51.0 | 1.88 | +0.125 | 84.5 | 18.75 | +0.125 |
| | 32 | 55.6 | 2.00 | +0.025 | 86.0 | 27.34 | +0.000 |
| AIME24 | 8 | 39.6 | 1.57 | +0.033 | 80.6 | 15.23 | -0.033 |
| | 16 | 54.5 | 1.93 | +0.067 | 86.7 | 24.21 | +0.000 |
| | 32 | 62.6 | 2.17 | +0.000 | 90.1 | 36.86 | +0.033 |
| **Avg** | — | **47.2** | **1.77** | **+0.043** | **82.4** | **19.62** | **+0.023** |

Table D-6: Results on math benchmarks using the LLaMA3.2-3B-Instruct model with Qwen2.5-Math-PRM-7B. Token reduction, latency speedup, and accuracy difference are reported for THROW relative to Best-of-N (BoN) and Beam Search (BS) across different numbers of trajectories ($N = 8, 16, 32$).

| **Benchmark** | **N** | **THROW vs. BoN** | | | **THROW vs. BS** | | |
|---|---|---|---|---|---|---|---|
| | | **Tok. Red. ↑ (%)** | **Lat. Speedup ↑ (×)** | **Acc. Diff ↑** | **Tok. Red. ↑ (%)** | **Lat. Speedup ↑ (×)** | **Acc. Diff ↑** |
| MATH | 8 | 22.0 | 1.28 | -0.034 | 76.8 | 7.13 | -0.048 |
| | 16 | 33.7 | 1.48 | -0.006 | 83.1 | 11.82 | -0.024 |
| | 32 | 44.6 | 1.48 | -0.018 | 87.1 | 16.40 | -0.038 |
| AMC23 | 8 | 40.4 | 1.60 | +0.050 | 83.1 | 9.14 | -0.100 |
| | 16 | 35.4 | 1.37 | +0.000 | 84.1 | 11.84 | -0.075 |
| | 32 | 28.9 | 1.27 | +0.050 | 86.0 | 17.31 | -0.075 |
| AIME24 | 8 | 44.9 | 1.43 | +0.033 | 84.2 | 8.60 | -0.033 |
| | 16 | 35.5 | 1.37 | +0.000 | 86.7 | 15.11 | +0.067 |
| | 32 | 28.1 | 1.23 | +0.033 | 85.3 | 16.67 | +0.067 |
| **Avg** | — | **34.8** | **1.39** | **+0.012** | **84.0** | **12.67** | **-0.029** |