TRIM: Hybrid Inference via Targeted Stepwise Routing in Multi-Step Reasoning Tasks

Anonymous authors
Paper under double-blind review

Abstract

Multi-step reasoning tasks like mathematical problem solving are vulnerable to cascading failures where a single incorrect step leads to complete solution breakdown. Current LLM routing methods assign entire queries to one model, treating all reasoning steps as equal. We propose TRIM (Targeted Routing in Multi-step reasoning tasks), which routes only critical steps to larger models while letting smaller models handle routine continuations. Our key insight is that targeted step-level interventions can fundamentally transform inference efficiency by confining expensive calls to precisely those steps where stronger models prevent cascading errors. TRIM operates at step-level granularity using process reward models to identify erroneous steps and makes routing decisions based on step-level uncertainty and budget constraints. We develop four routing strategies: a simple thresholding policy, two RL-trained policies (one using full sequential features, another using aggregated statistics), and a POMDP-based approach that handles uncertainty in step-level correctness estimates. On MATH-500, the thresholding policy already surpasses contemporary routing methods with 5x higher cost efficiency, while RL-trained and POMDP-based policies match the strong, expensive model's performance using 80% fewer expensive model tokens. All methods generalize effectively across mathematical reasoning datasets, demonstrating that step-level difficulty represents fundamental characteristics of multi-step reasoning.

1 Introduction

The rapid progress in large language models (LLMs) has led to an increasingly diverse ecosystem of models, spanning a wide spectrum of sizes, capabilities, and computational demands. Larger models typically achieve stronger performance but incur substantial serving costs, rendering them impractical for many routine applications. In contrast, smaller models are more affordable to deploy but often produce lower-quality responses. This trade-off poses a fundamental dilemma for practical deployment of LLMs: routing all queries to the largest available model ensures high-quality outputs but is prohibitively expensive, whereas relying solely on smaller models reduces serving costs at the expense of degraded response quality, especially on challenging queries.

Contemporary routing strategies attempt to mitigate this dilemma by assigning each query to a single model, which is then responsible for the *entire generation*. However, in LLM response generation, not all tokens are equally difficult: some tokens represent critical decision points that can dramatically alter the solution path (Bigelow et al., 2025; Setlur et al., 2024; Wang et al., 2025; Qu et al., 2025), while others are routine continuations that are easier to generate. When existing routing methods commit to a larger LLM over a smaller one for a given query, they implicitly assume that the intervention of the larger LLM is equally necessary at every token to produce a high-quality response. This inefficiency is particularly pronounced in multi-step reasoning tasks such as step-by-step reasoning or code generation, where mistakes early on can snowball into a complete failure (Zhang et al., 2024). It is precisely at these erroneous steps that the intervention of a stronger model is most valuable. Yet, contemporary routing methods often incur substantial inefficiency by defaulting to full generations from the larger model, even when targeted interventions at these erroneous steps would be sufficient.

To address this inefficiency, we introduce TRIM-**T**argeted Stepwise **R**outing for **I**nference in **M**ulti-step Reasoning Tasks — an approach that selectively routes only the most critical steps to larger LLMs. Unlike prompt-level routers, TRIM operates at the granularity of individual reasoning steps,

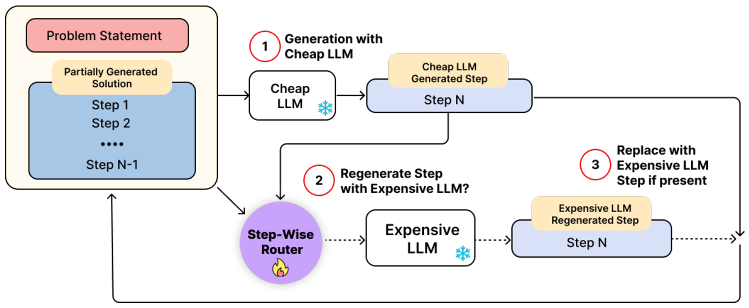
Problem Statement
Partially Generated Solution
Step 1
Step 2
••••
Step N-1
1 Generation with Cheap LLM
Cheap LLM
Cheap LLM Generated Step
Step N
2 Regenerate Step with Expensive LLM?
3 Replace with Expensive LLM Step if present
Step-Wise Router
Expensive LLM
Expensive LLM Regenerated Step
Step N

Figure 1: Schematic Overview of a Two-Model Setup for TRIM

generating solutions one step at a time. As illustrated in Figure 1, a stepwise router evaluates each intermediate step as it is generated and decides whether to accept the small model's output or to regenerate that specific step with a larger model. TRIM ensures that interventions occur only when necessary at individual steps, rather than handing over the entire remaining solution to the larger model. This step-by-step generation process with targeted intervention enables TRIM to achieve efficiency by confining expensive calls to precisely those steps where intervention prevents cascading errors while allowing the smaller model to handle routine continuations. This approach reduces the primary cost bottleneck in inference: the number of tokens generated by the expensive, larger LLM.

Building on this framework, we design multiple strategies for stepwise routing, each tailored to different computational and informational constraints: a simple thresholding policy that uses step-level scores to identify erroneous steps, two reinforcement learning-trained policies that reason about long-horizon trade-offs between accuracy and cost (one using full sequential features, another using aggregated statistics), and a Partially Observable Markov Decision Process (POMDP) based approach that accounts for the inherent uncertainty in step-level correctness estimates while enabling efficient policy recomputation across different cost budgets. Our cost metric focuses on the number of tokens generated by the expensive model, as prefill costs can be amortized through parallel decoding strategies (Leviathan et al., 2023; Cai et al., 2024) while generation tokens impose unavoidable sequential costs. We evaluate our approach on diverse mathematical reasoning benchmarks, including MATH (Hendrycks et al., 2021), Olympiad-Bench (He et al., 2024), and AIME (Di Zhang, 2025). Testing on MATH-500 shows that our basic thresholding policy is 5× more cost-effective than existing methods, while our trained RL and POMDP policies achieve the performance of the expensive LLM using only 20% of the expensive tokens.

Our main contributions are: **(1)** We establish the key insight that targeted step-level interventions can fundamentally transform the efficiency of multi-step reasoning. **(2)** We show that this insight translates into practical gains across different complexity levels—from simple thresholding policies that surpass existing query-level routing methods to advanced RL-trained and POMDP-based approaches that achieve competitive performance with oracle routers having perfect task knowledge, all while using significantly fewer expensive model tokens. **(3)** We establish that routing policies can be trained effectively with limited supervision data while achieving robust performance across diverse cost budgets, making the approach practical for real-world deployment scenarios. **(4)** Finally, we demonstrate robust generalization across datasets, with methods trained on AIME delivering strong efficiency gains on OlympiadBench and Minerva Math, suggesting that step-level difficulty patterns reflect universal characteristics of multi-step reasoning rather than dataset-specific features.

2 Related Work

The trade-off between model performance and computational cost has become increasingly important as large language models grow in size and capability. Our work on targeted stepwise routing builds upon several key areas of research: LLM routing strategies, process supervision and step-level verification, and multi-step reasoning optimizations.

LLM Routing and Model Selection. Traditional routing approaches operate at the query level, assigning entire queries to a single model based on estimated difficulty or uncertainty. Hybrid-LLM (Ding et al., 2024) frames this as a classification task using BERT-style encoders, while Zooter (Lu et al., 2023) employs reward-guided training for normalized reward prediction. RouteLLM (Ong et al.,

2024) learns routing directly from preference data, and AutoMix (Aggarwal et al., 2023) formulates query-level routing as a POMDP with self-verification for difficulty estimation. Recent advances have extended these approaches in complementary directions. BEST-Route (Ding et al., 2025) combines model selection with adaptive test-time compute allocation through best-of-n sampling, achieving up to 60% cost reduction. The unified approach of (Dekoninck et al., 2025) theoretically combines routing and cascading into "cascade routing," proving optimality conditions for model selection strategies. However, all these methods assume uniform difficulty across generation steps, leading to inefficient resource allocation in multi-step reasoning tasks where step difficulty varies significantly.

Process Supervision and Step-Level Verification. Process reward models (PRMs) have emerged as a powerful technique for evaluating intermediate reasoning steps. Human-in-the-loop step verfication method used by (Lightman et al., 2023) demonstrated that process supervision significantly outperforms outcome supervision on mathematical reasoning tasks. Since human annotation is not scalable, recent work has developed automated supervision methods: (Luo et al., 2024) proposed automated process supervision techniques, while (Wang et al., 2023) introduced framework to verify and reinforce LLMs step-by-step without human annotations. Recent advances have shown that strategic computation allocation (Hwang et al., 2024; Snell et al., 2024; Setlur et al., 2024) can significantly improve mathematical reasoning efficiency. While these works primarily use PRMs for candidate selection or exploration shaping, our approach leverages PRM scores to inform routing decisions during generation.

Multi-Step Reasoning and Test-Time Compute. Recent work has highlighted the importance of strategic computation allocation in multi-step reasoning. Research on "forking paths" (Bigelow et al., 2025) and (Wang et al., 2025) demonstrates that certain tokens represent critical decision points that dramatically alter solution trajectories. Similarly, work on reinforcement learning for mathematical reasoning (Setlur et al., 2024; DeepSeek-AI et al., 2025; Yang et al., 2025; Team et al., 2025) and optimizing test-time compute (Snell et al., 2024; Qu et al., 2025) shows that targeted interventions at crucial steps can be more effective than uniform computation increases. The observation that high-entropy minority tokens drive effective learning (Wang et al., 2025) further supports the notion that not all generation steps are equally important.

Speculative and Parallel Decoding. Our approach shares some similarities with speculative decoding (Leviathan et al., 2023) and parallel inference strategies (Cai et al., 2024), which also involve multiple models collaborating during generation. However, these methods focus on acceleration through draft-and-verify paradigms, while our work targets the quality-cost trade-off in multi-step reasoning through selective model escalation. Reward-Guided SD (Liao et al.) and SpecReason (Pan et al., 2025) are closer in spirit but differ fundamentally in objective. Both operate in a fixed, high-budget regime and aim to reduce latency and marginally boost accuracy of the strong model. In contrast, TRIM is designed to maximize accuracy under an explicit cost-performance tradeoff, where only a limited budget of strong-model tokens is allowed. Moreover, even when adapted to a routing context, they don't account for inaccuracies in correctness estimates (RSD relies on PRM, while SpecReason prompts the expensive model for scores) and remain inherently myopic, lacking the long-horizon planning needed for efficient cost allocation.

3 Preliminaries and Problem Statement

We define the problem of **stepwise routing** within a multi-step reasoning task as a sequential decision process. The objective is to derive a routing policy that, at each step of generation, decides whether to (i) accept the output of a cheap language model, or (ii) re-generate the step using a more capable but expensive LLM, thereby incurring an additional per-token cost. This policy must balance the trade-off between maximizing the task reward of the final solution and minimizing the serving cost, defined as the number of tokens generated from the expensive, stronger LLM. We formalize this problem below.

Problem Formulation. A multi-step reasoning task is specified by a query $q \in \mathcal{Q}$ and a sequence of reasoning steps

$$\mathbf{y}_{1:N} = (q, y_1, y_2, \ldots, y_N) \in \mathcal{Y}_N,$$

where y_i denotes the i-th reasoning step. For our purposes, we assume these steps are delimited by double newlines in the generated text. At time-step t, the current prefix is denoted $\mathbf{y}_{1:t} = (q, y_1, \ldots, y_t) \in \mathcal{Y}_t$, where $\mathcal{Y}_t$ denotes the set of all prefixes of length t. Within $\mathcal{Y}_t$, we distinguish two disjoint subsets: (1) $\mathcal{P}_t \subseteq \mathcal{Y}_t$, the set of *incomplete prefixes* (partial answers) that can be extended further; (2) $\mathcal{C}_t \subseteq \mathcal{Y}_t$, the set of *completed (terminated) answers* at step t.

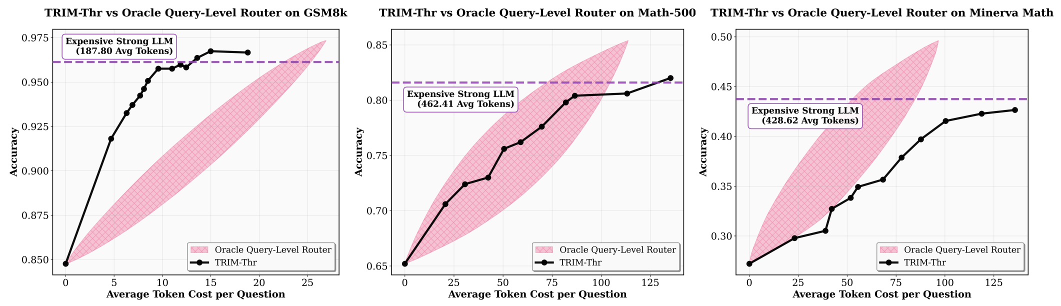

Figure 2: **Comparison of task performance–cost trade-offs** for Qwen2.5-3B-Instruct (M_w) and Claude 3.7 Sonnet (M_s), under the myopic thresholding policy (with Qwen2.5-Math-PRM-7B) versus the Idealized Oracle Query-level Router, across multiple math benchmarks. The Oracle Router is evaluated by incrementally varying the number of queries routed to M_s, selecting those solvable by M_s but not by M_w. Since multiple query subsets can achieve the same accuracy with different token costs, the oracle's performance–cost curve forms a shaded region rather than a single line, reflecting the full trade-off frontier.

Let $\mathcal{M}$ denote a set of language models, where each model $M \in \mathcal{M}$ can be viewed as a function that maps a partial reasoning trace $\mathbf{y}_{1:t} \in \mathcal{P}_t$ to the next step $\mathbf{y}_{1:t+1} \in \mathcal{Y}_{t+1}$, thereby extending the prefix or producing a completed solution. We consider a two-model setup consisting of two classes of models $\mathcal{M}$: (1) *strong expensive LLM* $\mathcal{M}_s$ that produce high-quality responses but incurs a per-token cost (2) *cheap LLM* $\mathcal{M}_w$ which offer relatively lower-quality responses at negligible cost. This is used to model the trade-off between quality and cost by transitioning from closed-source to open-source models.

Our goal is to learn a stepwise routing policy π that, at each reasoning step t, chooses an action $a_t \in \{\texttt{continue}, \texttt{regenerate}\}$, determining whether to accept the weak model's continuation or to replace it with the strong model's generation.

Formally, for $M_w \in \mathcal{M}_w$ and $M_s \in \mathcal{M}_s$

- If $a_t = \texttt{continue}$, the next prefix $\mathbf{y}_{1:t+1} = (\mathbf{y}_{1:t}, M_w(\mathbf{y}_{1:t}))$
- If $a_t = \texttt{regenerate}$, the policy instead sets $\mathbf{y}'_{1:t} = (\mathbf{y}_{1:t-1}, M_s(\mathbf{y}_{1:t-1}))$, and then continues with $\mathbf{y}_{1:t+1} = (\mathbf{y}'_{1:t}, M_w(\mathbf{y}'_{1:t}))$.

For $\mathbf{y}_{1:t} \in C_t$, choosing $a_t = \texttt{regenerate}$ yields $\mathbf{y}'_{1:t} = (\mathbf{y}_{1:t-1}, M_s(\mathbf{y}_{1:t-1}))$, while choosing $a_t = \texttt{continue}$ leaves the prefix unchanged; in either case, the reasoning process terminates.

The quality of intermediate steps in a reasoning trace can be estimated by assigning probabilistic scores. This can be achieved through methods such as self-verification, process reward models (PRMs), or other step-level evaluation techniques. For our experiments, we adopt a PRM that evaluates partial traces and produces a sequence of step-level scores. Formally, given a reasoning trace $\mathbf{y}_{1:t}$, the PRM assigns step-level rewards as $\mathbf{r}_{1:t} = \text{PRM}(\mathbf{y}_{1:t}) = (r_1, r_2, \ldots, r_t)$, where $r_i = (\mathbf{r}_{1:t})_i$ denotes the reward for step i.

We use these scores as proxies for step-level correctness, providing a signal for whether escalation to a larger model is likely to offer additional benefit given the current prefix. In practice, the PRM score for a solution can also be aggregated across steps using either the product of all step scores or the minimum score across steps (Wang et al., 2023; Lightman et al., 2023). These aggregated scores are widely used in practice for ranking and comparing multiple candidate solutions. Although PRMs have been used in prior work primarily to improve candidate selection in beam search or to shape exploration in reinforcement learning (Snell et al., 2024; Setlur et al., 2024), our use is distinct: we leverage PRM outputs to inform routing decisions during generation.

4 TRIM AND ROUTING STRATEGY DESIGNS

We develop a framework called TRIM to perform routing at the step-level for every query. Rather than routing an entire query to a strong model, TRIM operates directly at the level of routing steps in a response. We illustrate the design of TRIM in Figure 1. Concretely, the routing process of TRIM works as follows: at each step t of the reasoning process, the cheap model M_w proposes a

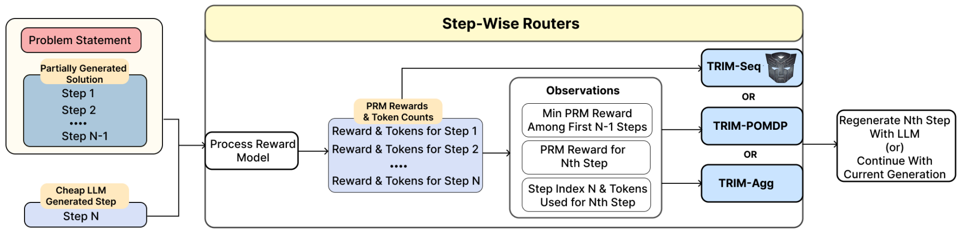
Problem Statement
Partially Generated Solution
Step 1
Step 2
Step N-1
Cheap LLM Generated Step
Step N
Step-Wise Routers
Process Reward Model
PRM Rewards & Token Counts
Reward & Tokens for Step 1
Reward & Tokens for Step 2
Reward & Tokens for Step N
Observations
Min PRM Reward Among First N-1 Steps
PRM Reward for Nth Step
Step Index N & Tokens Used for Nth Step
TRIM-Seq
OR
TRIM-POMDP
OR
TRIM-Agg
Regenerate Nth Step With LLM (or) Continue With Current Generation

Figure 3: **Step-Wise Router architecture for TRIM** using process rewards to evaluate partial solutions and uses RL-based policies or POMDP-based solvers for making routing decisions.

candidate continuation $y_t^w = M_w(\mathbf{y}_{1:t-1})$. The router policy then evaluates the partial reasoning trace together with y_t^w and decides whether to accept this step ($a_t = \texttt{continue}$) or to escalate to the strong model M_s ($a_t = \texttt{regenerate}$), which regenerates the step as $y_t^s = M_s(\mathbf{y}_{1:t-1})$. Based on this decision, the appended step is $y_t = y_t^w$ if $a_t = \texttt{continue}$ and $y_t = y_t^s$ otherwise. Thus, TRIM incrementally constructs the solution trace by appending at each position either the M_w-generated step or the M_s-regenerated step, depending on the router's action. This differs from query-level routing, which makes a single global decision to assign the entire query to either M_w or M_s.

We now describe different strategies for learning routing policies within TRIM. These strategies differ in how much information they incorporate about the reasoning trajectory and whether they make decisions in a myopic (step-local) or non-myopic (trajectory-aware) fashion. At one extreme, thresholding policies rely only on current step correctness, while RL and POMDP-based approaches account for long-horizon trade-offs between accuracy and cost.

4.1 TRIM-Thr: Myopic Thresholding Policy

We first introduce a simple yet effective routing policy that solely relies on the PRM score of the current generated step of the cheap model M_w to make routing decisions. If this probability falls below a predefined threshold k, the router regenerates the step with the strong model M_s; otherwise, it accepts the cheap model's output and continues generation. Adjusting the threshold parameter k provides a principled way to vary the task performance-cost trade-off. Formally, the policy is

$$\pi_{\text{thr},k}(\mathbf{y}_{1:t}) = \begin{cases} \texttt{regenerate}, & \text{if PRM}(\mathbf{y}_{1:t})_t < k, \\ \texttt{continue}, & \text{otherwise.} \end{cases} \tag{1}$$

4.2 RL-Trained Policies

While TRIM-Thr demonstrates strong performance (Figure 2), it is inherently *myopic* in the sense that it makes routing decisions solely based on the correctness estimate of the most recent step, without considering past context or future consequences. A richer set of signals can be exploited for more effective decision-making. For instance, even if the most recent step is predicted to be incorrect, regenerating it with a strong model may not be beneficial if the overall trajectory is already far from the correct solution, or if the additional cost of intervention outweighs the potential gain. Conversely, when prior steps are largely consistent and the trace remains plausibly aligned with a correct solution, targeted intervention can be highly impactful.

Prior work on stepwise verification (Lightman et al., 2023; Wang et al., 2023) and reranking with beam search highlights the importance of leveraging the sequence of correctness scores accumulated over the reasoning trace, rather than focusing exclusively on the most recent one. Additionally, token counts at each step provide information about the cost of regenerating with M_s. Incorporating these richer signals allows the router to reason jointly about (i) whether the trace remains plausibly on track toward a correct solution and (ii) whether the cost of intervention is justified, enabling more principled stepwise routing policies beyond TRIM-Thr.

TRIM-Seq: Learning to Route from Sequential Features. Formally, let $\mathbf{c}_{1:t} = (c_1, \ldots, c_t)$ denote the token counts associated with each step $y_1, \ldots, y_t$ in the reasoning trace $\mathbf{y}_{1:t}$ and $\mathbf{r}_{1:t} = (r_1, \ldots, r_t)$ be the stepwise correctness scores, then joint feature sequence is $\mathbf{f}_{1:t} = \big((r_1, c_1), (r_2, c_2), \ldots, (r_t, c_t)\big)$. These features provide complementary information: correctness estimates guide the policy toward semantic fidelity, while token counts enable cost-sensitive reasoning about the expense of regenerating the current step.

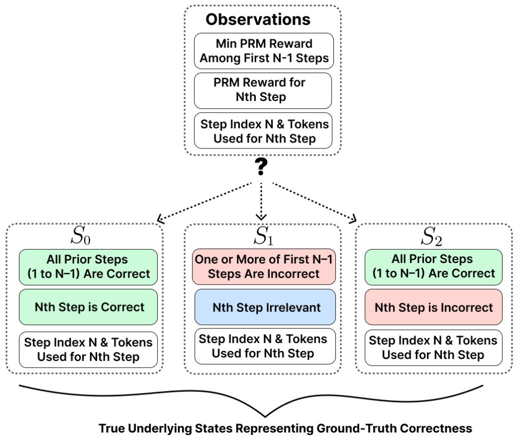
Observations
Min PRM Reward Among First N-1 Steps
PRM Reward for Nth Step
Step Index N & Tokens Used for Nth Step
?
S_0
All Prior Steps (1 to N–1) Are Correct
Nth Step is Correct
Step Index N & Tokens Used for Nth Step
S_1
One or More of First N–1 Steps Are Incorrect
Nth Step Irrelevant
Step Index N & Tokens Used for Nth Step
S_2
All Prior Steps (1 to N–1) Are Correct
Nth Step is Incorrect
Step Index N & Tokens Used for Nth Step
True Underlying States Representing Ground-Truth Correctness

Figure 4: **POMDP Components:** Observation space Ω and State Space S Classes

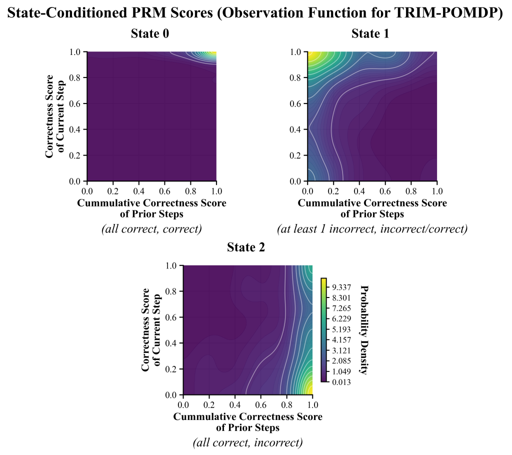

Figure 5: **Observation function** obtained from ProcessBench (Omni-MATH), shown as heatmaps of PDF of PRM-based observations conditioned on state class

We parameterize the routing policy with a transformer-based network that processes the feature sequence $\mathbf{f}_{1:t}$ and outputs a distribution over actions $a_t \in \{\texttt{continue}, \texttt{regenerate}\}$. The policy is optimized via RL: each regeneration action (`regenerate`) on a prefix $\mathbf{y}_{1:t}$ incurs a cost proportional to the number of tokens generated by the strong model M_s, $\lambda \cdot |M_s(\mathbf{y}_{1:t-1})|$, where $\lambda > 0$ is a cost-performance trade-off parameter, and $|M_s(\mathbf{y}_{1:t-1})|$ denotes the token length of the strong model's regenerated output. The episodic return further includes the (binary) terminal task reward R, which reflects the correctness of the final solution. The policy is thus optimized to maximize the expected return

$$J(\pi) = \mathbb{E}_\pi \left[R(\mathbf{y}_{1:T}) - \lambda \sum_{t=1}^{T} \mathbf{1}\{a_t = \texttt{regenerate}\} \cdot |M_s(\mathbf{y}_{1:t-1})| \right],$$

which balances solution correctness against the cumulative generation cost of invoking M_s.

TRIM-Agg: Learning to Route from Aggregated Features. TRIM-Seq, as described above, leverages the full sequence of stepwise correctness scores $(r_1, r_2, \ldots, r_{t-1})$ together with token lengths to model routing decisions. While this provides rich sequential context, it is often useful to consider simpler feature representations that capture the most salient statistics of the reasoning trace. Prior work (Lightman et al., 2023; Wang et al., 2023) suggests that solution-level correctness can be estimated from stepwise scores using reductions such as the minimum or product over steps. Motivated by this, we construct a reduced feature set $\tilde{\mathbf{f}}_{1:t} = \big(r_t,\ \min(\mathbf{r}_{1:t-1}),\ c_t,\ t\big)$, where $\min(\mathbf{r}_{1:t-1}) = \min(r_1, r_2, \ldots, r_{t-1})$, c_t denotes the token length of the current step, and t indexes the current position in the trace.

This reduced representation discards the full sequential history while retaining key aggregated indicators of correctness and cost. Using $\tilde{\mathbf{f}}_{1:t}$, we train a policy network under the same RL objective as in TRIM-Seq. Empirically, this design yields substantially faster training, and the performance gap relative to TRIM-Seq is negligible for any given tradeoff parameter λ.

4.3 TRIM-POMDP: POMDP-Based Router Policy

A central challenge in stepwise routing methods discussed above arises from the imperfect nature of process reward model (PRM) estimates. While PRMs provide informative signals about the correctness of intermediate steps, their predictions are often noisy and can misclassify correct steps as incorrect (or vice versa). When trained on large amounts of data, routing policies like TRIM-Agg can implicitly learn to discard errors in the PRM estimates. However, RL under long-horizon sparse rewards makes training such policies both sample-inefficient and expensive. This raises a key question: how can routing decisions be improved when only noisy correctness signals are available?

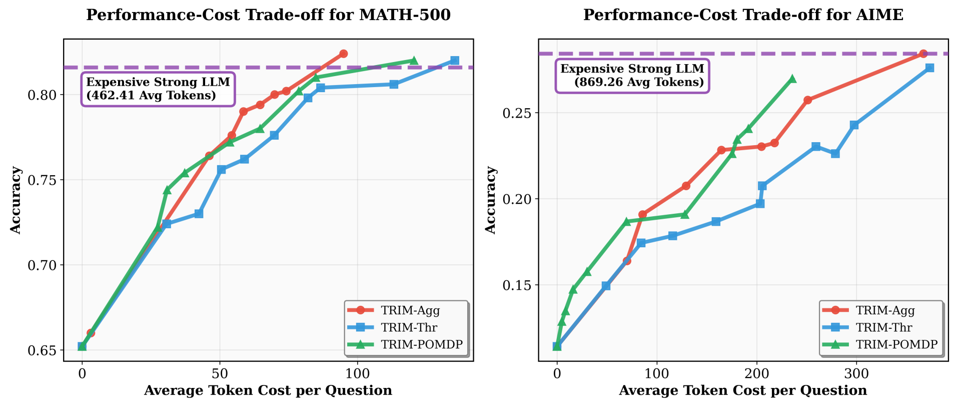

Figure 6: **Performance–cost trade-offs** of different TRIM routing approaches on MATH-500 and AIME.

Our solution is to explicitly treat PRM scores as imperfect observations of an unobserved latent state that reflects true correctness of the reasoning trajectory, and attempt to infer the true latent space first when learning a routing policy (akin to control in a partially-observed Markov decision process). As shown in Figure 4, in TRIM-POMDP, the latent state is defined in terms of three correctness classes (augmented with the current step index and token cost): (i) S_0, where the trajectory remains correct so far, (ii) S_1, where the trajectory has already diverged irrecoverably, and (iii) S_2, where the most recent step is incorrect but prior steps are correct, leaving the trajectory still potentially recoverable. If this latent state were perfectly observed, the routing problem would reduce to solving a fully observable MDP. In practice, however, the latent correctness state is hidden, and we only observe noisy proxies provided by the PRM. To bridge this gap, we learn an *observation function* that helps us map the entire history of observations ($\tilde{\mathbf{f}}_{1:t}$) to a probability distribution over the latent states. Concretely, this amounts to modeling the distribution of PRM outputs conditioned on state classes (see Figure 5), which can be fit offline using process supervision datasets with ground-truth step-level annotations (e.g., ProcessBench (Zheng et al., 2024)). Moreover, because the observation function only requires aligning PRM scores with annotated correctness labels, it can be trained once and reused across different performance–cost trade-off parameters λ. Once this mapping is learned, we can invoke a POMDP solver on-the-fly to compute routing policies that optimally balance accuracy and cost.

This compact POMDP formulation of the sequential routing problem enable efficient policy computation using standard POMDP solvers. Moreover, policy computation with modern POMDP solvers is both efficient and flexible, with offline solvers typically requiring less than a minute runtime. As a result, policies can be recomputed easily for different performance–cost trade-off parameters λ. A further advantage of TRIM-POMDP is that the resulting routing policy is largely agnostic to the specific choice of LLMs (M_s, M_w), depending only on their next-step accuracies provided as inputs to the transition function. See Appendix A.1 for further details and the complete POMDP formulation.

5 Experiments

We evaluate TRIM by benchmarking its stepwise routing strategies against established query-level routing approaches. Our primary comparison is with RouteLLM (Ong et al., 2024), a state-of-the-art approach for query-level routing, which uses preference data for making these decisions. We begin by introducing the evaluation metrics used throughout our analysis.

Metrics. We evaluate various router policies by quantifying the trade-off between task performance and the cost of invoking the strong model M_s. In our setting, the cost of a query is measured by the number of tokens generated by the expensive (strong) model M_s. We adapt evaluation metrics from prior work to the setting of stepwise routing in multi-step reasoning. For a query q in the set of queries $\mathcal{Q}$, let $C(q;\pi)$ denote the number of tokens generated by the strong model M_s under router policy π, and let $C_s(q)$ be the number of tokens generated when using M_s alone. We define $\bar{C}(\pi)$ as the average number of M_s tokens per query and $c(\pi)$ as the normalized fraction of tokens from M_s:

$$\bar{C}(\pi) = \frac{1}{|\mathcal{Q}|}\sum_{q\in\mathcal{Q}} C(q;\pi), \qquad c(\pi) = \frac{\sum_{q\in\mathcal{Q}} C(q;\pi)}{\sum_{q\in\mathcal{Q}} C_s(q)}. \tag{2}$$

	MATH-500				AIME			
Method	CPT(50%)	CPT(80%)	CPT(95%)	Δ_{IBC}	CPT(50%)	CPT(80%)	CPT(95%)	Δ_{IBC}
BERT	196.60 (42.52%)	331.45 (71.68%)	394.49 (85.31%)	0.08	331.85 (38.18%)	616.53 (70.93%)	701.58 (80.71%)	0.44
MF	160.83 (34.78%)	324.13 (70.10%)	432.60 (93.55%)	0.49	358.81 (41.28%)	602.62 (69.32%)	813.28 (93.56%)	0.65
SW Ranking	185.74 (40.17%)	279.47 (60.44%)	330.89 (71.56%)	0.37	297.08 (34.18%)	496.77 (57.15%)	715.72 (82.34%)	0.79
Smoothie	220.69 (47.73%)	345.19 (74.65%)	433.77 (93.81%)	0.30	396.79 (45.65%)	704.50 (81.05%)	822.09 (94.57%)	0.03
AutoMix-PRM	110.42 (23.88%)	198.73 (42.98%)	249.49 (53.96%)	0.95	380.48 (43.77%)	605.83 (69.7%)	703.77 (80.96%)	0.07
TRIM-Thr	43.68 (9.45%)	73.74 (15.95%)	115.99 (25.08%)	4.75	204.01 (23.47%)	314.7 (36.2%)	372.79 (42.89%)	1.81
TRIM-Agg	33.74 (7.3%)	**56.49 (12.22%)**	**79.58 (17.21%)**	5.67	**107.39 (12.35%)**	241.55 (27.79%)	330.42 (38.01%)	2.50
TRIM-POMDP	**29.27 (6.33%)**	66.63 (14.41%)	83.12 (17.98%)	**5.86**	139.21 (16.01%)	**206.06 (23.71%)**	**244.86 (28.17%)**	**5.00**

Table 1: Comparison of TRIM across AIME & MATH-500 benchmarks.

	OlympiadBench				Minerva Math			
Method	CPT(50%)	CPT(80%)	CPT(95%)	Δ_{IBC}	CPT(50%)	CPT(80%)	CPT(95%)	Δ_{IBC}
BERT	367.75 (55.03%)	584.14 (87.41%)	642.50 (96.14%)	-0.04	209.99 (48.99%)	378.24 (88.25%)	421.44 (98.33%)	-0.1
MF	369.15 (55.24%)	522.68 (78.21%)	601.77 (90.05%)	-0.07	166.99 (38.96%)	249.82 (58.28%)	326.06 (76.07%)	0.42
SW Ranking	351.34 (52.57%)	511.08 (76.48%)	635.05 (95.03%)	0.07	212.73 (49.63%)	342.71 (79.96%)	421.17 (98.26%)	0.04
Smoothie	348.59 (52.16%)	511.64 (76.56%)	615.49 (92.10%)	-0.08	234.66 (54.75%)	345.16 (80.53%)	402.19 (93.83%)	-0.09
AutoMix-PRM	265.95 (39.8%)	411.47 (61.57%)	481.49 (72.05%)	0.22	72.08 (16.82%)	140.24 (32.72%)	196.12 (45.76%)	1.35
TRIM-Thr	136.64 (20.45%)	220.70 (33.03%)	313.89 (46.97%)	1.31	65.15 (15.2%)	92.78 (21.65%)	148.55 (34.66%)	2.23
TRIM-Agg	**94.4 (14.13%)**	**190.11 (28.45%)**	**287.17 (42.97%)**	**2.57**	**47.37 (11.05%)**	**89.54 (20.89%)**	**138.67 (32.35%)**	**3.12**

Table 2: Cross-Benchmark Generalization of Routers Trained on AIME

Following Ong et al. (2024), if $s(q;\pi) \in \{0,1\}$ denotes the correctness of query q under policy π, the average performance and the *performance gap recovered (*PGR*)* are defined as

$$r(\pi) = \frac{1}{|\mathcal{Q}|} \sum_{q \in \mathcal{Q}} s(q;\pi), \qquad \text{PGR}(\pi) = \frac{r(\pi) - r(M_w)}{r(M_s) - r(M_w)}, \tag{3}$$

where $r(M_s)$ and $r(M_w)$ denote the accuracies of the M_s and M_w, respectively. $\text{PGR}(\pi)$ quantifies how much of the performance gap between M_w and M_s is recovered by policy π.

To capture the cost required to achieve a desired level of performance, we utilize *cost–performance threshold (*CPT*)*. Specifically, $\text{CPT}(x\%)$ denotes the minimum token cost (in terms of $\bar{C}$ or c) required by policy π to achieve a PGR of $x\%$, providing a measure of efficiency at different target performance levels. Finally, following Aggarwal et al. (2023), we report the *incremental benefit per cost (IBC)* used by the routing system as:

$$IBC(\pi) = \frac{r(\pi) - r(M_w)}{\bar{C}(\pi)}, \quad IBC_{\text{Base}} = \frac{r(M_s) - r(M_w)}{\frac{1}{|\mathcal{Q}|}\sum_{q \in \mathcal{Q}} C_s(q;\pi)}, \quad \Delta_{IBC}(\pi) = \frac{IBC(\pi) - IBC_{\text{Base}}}{IBC_{\text{Base}}} \tag{4}$$

Here $IBC(\pi)$ measures performance improvement per unit expensive-model M_s token usage, IBC_{Base} is the baseline corresponding to always using M_s, and $\Delta_{IBC}(\pi)$ quantifies relative gain. A positive Δ_{IBC} indicates that the router is more cost-effective than querying M_s for every input. For evaluation, we compute Δ_{IBC} across 100 equally sized performance regions between M_w and M_s and report the average.

Experimental setup. For our experimental analysis, we use a two-model setup with Qwen2.5-3B-Instruct as the cheap LLM (M_w) and Claude 3.7 Sonnet as the expensive model (M_s), guided by Qwen2.5-Math-PRM-7B for step-level correctness estimation. For AIME (Di Zhang, 2025), we use an approximately 50–50 train–test split across alternate years and problem sets, while for MATH (Hendrycks et al., 2021), we train on the 7.5k official training set and evaluate on MATH-500. We benchmark three TRIM routing strategies—TRIM-Thr, TRIM-Agg, and TRIM-POMDP—against the query-level routing methods proposed in RouteLLM (Ong et al., 2024), namely the BERT classifier, matrix factorization, and SW ranking models. In TRIM-Agg, the router is parameterized by a simple MLP policy with two hidden layers and is trained with PPO, while TRIM-POMDP uses the SARSOP solver (Kurniawati et al., 2008) to compute policies (see Appendix B for implementation details).

Results. Table 1 reports the evaluation results across benchmarks, and Figure 6 illustrates the performance–cost trade-off curves achieved by different TRIM strategies. To further assess the generalization capability of TRIM-Agg, we evaluate routers trained on AIME in a cross-dataset setting, testing on other math benchmarks of comparable difficulty, namely OlympiadBench and Minerva Math, and report results in Table 2 and the corresponding performance curves in Figure 7.

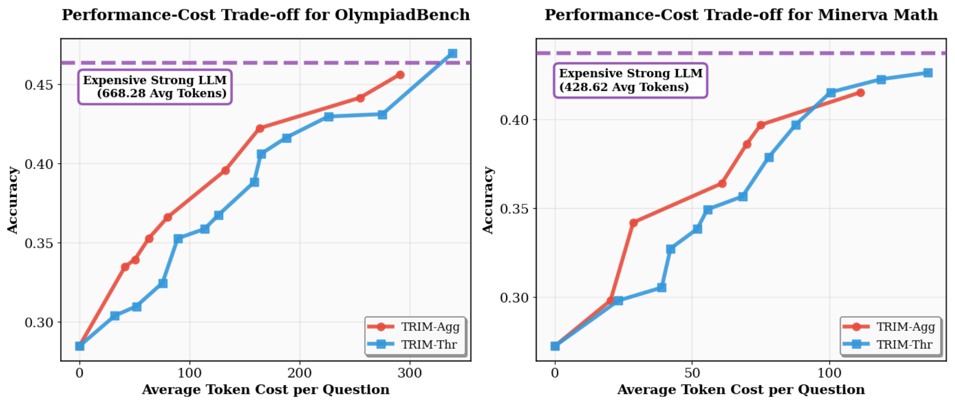

Figure 7: **Performance–cost trade-offs under dataset generalization.** TRIM-Agg routers trained on AIME demonstrate strong performance across benchmarks of similar difficulty

Our experiments reveal distinct strengths across regimes. As can be seen in Figure 6 for the low-budget setting (large λ), TRIM-POMDP achieves superior performance benefiting from principled long-horizon planning under uncertainty. Unlike RL-trained policies, which struggle in this regime due to sparse rewards, modern POMDP solvers efficiently compute policies without being hindered by sparse-reward learning dynamics. In the high-budget regime (small λ), however, RL-trained TRIM-Agg policies show strong performance, achieving 95% of the performance gap for MATH-500 while using approximately 80% fewer expensive tokens, as policy optimization becomes significantly easier. Even our simplest approach, TRIM-Thr, achieves $5\times$ ($\Delta_{IBC} = 4.75$ vs $\Delta_{IBC} = 0.95$) better cost efficiency than baselines. Beyond budget regimes, our cross-dataset evaluations highlight an important distinction: query-level routers can often fit to the intrinsic characteristics of specific datasets, while TRIM captures transferable routing behaviors that generalize across benchmarks of comparable difficulty. For instance, BERT achieves a Δ_{IBC} of 0.44 on AIME but drops dramatically to -0.04 on OlympiadBench and -0.1 on Minerva Math, while SW Ranking similarly degrades from 0.79 to 0.07 and 0.04, respectively. In contrast, TRIM-Agg achieves a Δ_{IBC} of 2.5 on AIME and maintains strong performance with Δ_{IBC} values of 2.57 on OlympiadBench and 3.12 on Minerva Math when trained solely on AIME, demonstrating superior generalization. This suggests that step-level difficulty patterns reflect fundamental properties of multi-step reasoning rather than dataset-specific artifacts.

Despite being trained on fewer than 500 samples from the AIME dataset, TRIM-Agg achieves strong performance on the held-out test set with 38.01% expensive token usage at $\mathrm{CPT}(95\%)$ and exhibits robust generalization to datasets of comparable difficulty, consistently surpassing both TRIM-Thr and query-level baselines. In parallel, TRIM-POMDP shows strong performance across all cost–performance trade-off regimes on both MATH-500 and AIME, despite its observation function being trained on different (but comparably difficult) math datasets (detailed in Appendix B) and the solver requiring only the estimated next-step accuracies of (M_w, M_s) as input.

6 Discussion and Conclusion

In this work, we present TRIM, an approach for targeted stepwise routing that escalates only critical steps to stronger, more expensive LLMs, intervening precisely where the partial reasoning trace risks diverging from a correct solution. Our key insight is that even a small number of well-placed interventions can dramatically boost task accuracy, enabling significant efficiency gains compared to conventional query-level routing. Building on this insight, we designed multiple routing strategies for TRIM that differ in how much trajectory-level information they exploit when making routing decisions. While TRIM already surpasses contemporary routing methods and performs competitively with oracle query-level routers, further improvements may be possible by moving beyond step-level granularity to token-level routing. Since certain tokens disproportionately influence downstream generation (Wang et al., 2025), token-level routing offers a promising direction for achieving even finer-grained and cost-efficient interventions.

7 Reproducibility Statement

In order to foster reproducibility of our work, we outline implementation details of our approach in Appendix B and Section 5.

8 Ethics Statements

All authors have read and agree to abide by the ICLR Code of Ethics. We acknowledge the usage of LLMs for editing (e.g., grammar, spelling, word choice) purposes for polishing writing. However, the use of LLMs was limited to editing and formatting text only.

References

Pranjal Aggarwal, Aman Madaan, Ankit Anand, Srividya Pranavi Potharaju, Swaroop Mishra, Pei Zhou, Aditya Gupta, Dheeraj Rajagopal, Karthik Kappaganthu, Yiming Yang, et al. Automix: Automatically mixing language models. *arXiv preprint arXiv:2310.12963*, 2023.

Eric J Bigelow, Ari Holtzman, Hidenori Tanaka, and Tomer Ullman. Forking paths in neural text generation. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=8RCmNLeeXx.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D Lee, Deming Chen, and Tri Dao. Medusa: Simple llm inference acceleration framework with multiple decoding heads. *arXiv preprint arXiv:2401.10774*, 2024.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

Jasper Dekoninck, Maximilian Baader, and Martin Vechev. A unified approach to routing and cascading for llms, 2025. URL https://arxiv.org/abs/2410.10347.

Di Zhang. Aime_1983_2024 (revision 6283828), 2025. URL https://huggingface.co/datasets/di-zhang-fdu/AIME_1983_2024.

Dujian Ding, Ankur Mallick, Chi Wang, Robert Sim, Subhabrata Mukherjee, Victor Ruhle, Laks VS Lakshmanan, and Ahmed Hassan Awadallah. Hybrid llm: Cost-efficient and quality-aware query routing. *arXiv preprint arXiv:2404.14618*, 2024.

Dujian Ding, Ankur Mallick, Shaokun Zhang, Chi Wang, Daniel Madrigal, Mirian Del Carmen Hipolito Garcia, Menglin Xia, Laks V. S. Lakshmanan, Qingyun Wu, and Victor Rühle. Best-route: Adaptive llm routing with test-time optimal compute, 2025. URL `https://arxiv.org/abs/2506.22716`.

Maxim Egorov, Zachary N. Sunberg, Edward Balaban, Tim A. Wheeler, Jayesh K. Gupta, and Mykel J. Kochenderfer. POMDPs.jl: A framework for sequential decision making under uncertainty. *Journal of Machine Learning Research*, 18(26):1–5, 2017. URL `http://jmlr.org/papers/v18/16-300.html`.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Leng Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, et al. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems. *arXiv preprint arXiv:2402.14008*, 2024.

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

Hyeonbin Hwang, Doyoung Kim, Seungone Kim, Seonghyeon Ye, and Minjoon Seo. Self-explore: Enhancing mathematical reasoning in language models with fine-grained rewards. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 1444–1466, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-emnlp.78. URL `https://aclanthology.org/2024.findings-emnlp.78/`.

Hanna Kurniawati, David Hsu, Wee Sun Lee, et al. Sarsop: Efficient point-based pomdp planning by approximating optimally reachable belief spaces. In *Robotics: Science and systems*, volume 2008. Zurich, Switzerland, 2008.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pp. 19274–19286. PMLR, 2023.

Baohao Liao, Yuhui Xu, Hanze Dong, Junnan Li, Christof Monz, Silvio Savarese, Doyen Sahoo, and Caiming Xiong. Reward-guided speculative decoding for efficient llm reasoning, 2025. *URL https://arxiv. org/abs/2501*, 19324.

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

Keming Lu, Hongyi Yuan, Runji Lin, Junyang Lin, Zheng Yuan, Chang Zhou, and Jingren Zhou. Routing to the expert: Efficient reward-guided ensemble of large language models. *arXiv preprint arXiv:2311.08692*, 2023.

Liangchen Luo, Yinxiao Liu, Rosanne Liu, Samrat Phatale, Meiqi Guo, Harsh Lara, Yunxuan Li, Lei Shu, Yun Zhu, Lei Meng, Jiao Sun, and Abhinav Rastogi. Improve mathematical reasoning in language models by automated process supervision, 2024. URL `https://arxiv.org/abs/2406.06592`.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E Gonzalez, M Waleed Kadous, and Ion Stoica. Routellm: Learning to route llms with preference data. *arXiv preprint arXiv:2406.18665*, 2024.

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

Yuxiao Qu, Matthew YR Yang, Amrith Setlur, Lewis Tunstall, Edward Emanuel Beeching, Ruslan Salakhutdinov, and Aviral Kumar. Optimizing test-time compute via meta reinforcement fine-tuning. *arXiv preprint arXiv:2503.07572*, 2025.

Amrith Setlur, Saurabh Garg, Xinyang Geng, Naman Garg, Virginia Smith, and Aviral Kumar. Rl on incorrect synthetic data scales the efficiency of llm math reasoning by eight-fold. *Advances in Neural Information Processing Systems*, 37:43000–43031, 2024.

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

Kimi Team, Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, Yuankun Chen, Yutian Chen, Zhuofu Chen, Jialei Cui, Hao Ding, Mengnan Dong, Angang Du, Chenzhuang Du, Dikang Du, Yulun Du, Yu Fan, Yichen Feng, Kelin Fu, Bofei Gao, Hongcheng Gao, Peizhong Gao, Tong Gao, Xinran Gu, Longyu Guan, Haiqing Guo, Jianhang Guo, Hao Hu, Xiaoru Hao, Tianhong He, Weiran He, Wenyang He, Chao Hong, Yangyang Hu, Zhenxing Hu, Weixiao Huang, Zhiqi Huang, Zihao Huang, Tao Jiang, Zhejun Jiang, Xinyi Jin, Yongsheng Kang, Guokun Lai, Cheng Li, Fang Li, Haoyang Li, Ming Li, Wentao Li, Yanhao Li, Yiwei Li, Zhaowei Li, Zheming Li, Hongzhan Lin, Xiaohan Lin, Zongyu Lin, Chengyin Liu, Chenyu Liu, Hongzhang Liu, Jingyuan Liu, Junqi Liu, Liang Liu, Shaowei Liu, T. Y. Liu, Tianwei Liu, Weizhou Liu, Yangyang Liu, Yibo Liu, Yiping Liu, Yue Liu, Zhengying Liu, Enzhe Lu, Lijun Lu, Shengling Ma, Xinyu Ma, Yingwei Ma, Shaoguang Mao, Jie Mei, Xin Men, Yibo Miao, Siyuan Pan, Yebo Peng, Ruoyu Qin, Bowen Qu, Zeyu Shang, Lidong Shi, Shengyuan Shi, Feifan Song, Jianlin Su, Zhengyuan Su, Xinjie Sun, Flood Sung, Heyi Tang, Jiawen Tao, Qifeng Teng, Chensi Wang, Dinglu Wang, Feng Wang, Haiming Wang, Jianzhou Wang, Jiaxing Wang, Jinhong Wang, Shengjie Wang, Shuyi Wang, Yao Wang, Yejie Wang, Yiqin Wang, Yuxin Wang, Yuzhi Wang, Zhaoji Wang, Zhengtao Wang, Zhexu Wang, Chu Wei, Qianqian Wei, Wenhao Wu, Xingzhe Wu, Yuxin Wu, Chenjun Xiao, Xiaotong Xie, Weimin Xiong, Boyu Xu, Jing Xu, Jinjing Xu, L. H. Xu, Lin Xu, Suting Xu, Weixin Xu, Xinran Xu, Yangchuan Xu, Ziyao Xu, Junjie Yan, Yuzi Yan, Xiaofei Yang, Ying Yang, Zhen Yang, Zhilin Yang, Zonghan Yang, Haotian Yao, Xingcheng Yao, Wenjie Ye, Zhuorui Ye, Bohong Yin, Longhui Yu, Enming Yuan, Hongbang Yuan, Mengjie Yuan, Haobing Zhan, Dehao Zhang, Hao Zhang, Wanlu Zhang, Xiaobin Zhang, Yangkun Zhang, Yizhi Zhang, Yongting Zhang, Yu Zhang, Yutao Zhang, Yutong Zhang, Zheng Zhang, Haotian Zhao, Yikai Zhao, Huabin Zheng, Shaojie Zheng, Jianren Zhou, Xinyu Zhou, Zaida Zhou, Zhen Zhu, Weiyu Zhuang, and Xinxing Zu. Kimi k2: Open agentic intelligence, 2025. URL `https://arxiv.org/abs/2507.20534`.

Peiyi Wang, Lei Li, Zhihong Shao, RX Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. *arXiv preprint arXiv:2312.08935*, 2023.

Shenzhi Wang, Le Yu, Chang Gao, Chujie Zheng, Shixuan Liu, Rui Lu, Kai Dang, Xionghui Chen, Jianxin Yang, Zhenru Zhang, et al. Beyond the 80/20 rule: High-entropy minority tokens drive effective reinforcement learning for llm reasoning. *arXiv preprint arXiv:2506.01939*, 2025.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL `https://arxiv.org/abs/2505.09388`.

Muru Zhang, Ofir Press, William Merrill, Alisa Liu, and Noah A. Smith. How language model hallucinations can snowball. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

Chujie Zheng, Zhenru Zhang, Beichen Zhang, Runji Lin, Keming Lu, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. Processbench: Identifying process errors in mathematical reasoning. *arXiv preprint arXiv:2412.06559*, 2024.

A Appendix

A.1 TRIM-POMDP

Foundations of POMDPs. A Partially Observable Markov Decision Process (POMDP) provides a principled framework for sequential decision-making under uncertainty when the underlying system state is not directly observable. A POMDP is defined by the tuple $(S, A, T, R, \Omega, \mathcal{O})$, where S is the state space, A the set of actions, and Ω the observation space. The transition function $T(s'|s, a)$ specifies the probability of transitioning from $s \in S$ to $s' \in S$ given $a \in A$, while the observation function $\mathcal{O}$ maps $s \in S$ and $a \in A$ to a probability distribution over the observation space Ω. The reward function $R(s, a)$ assigns a scalar reward to state–action pairs.

Since the true state is hidden, the agent maintains a *belief state* $b \in \Delta(S)$, a probability distribution over latent states, updated recursively via Bayes' rule after each action–observation pair. A policy is a mapping $\pi : b \mapsto a$, and the objective is to maximize expected discounted return by optimizing over policies.

A.1.1 Formalizing TRIM-POMDP

We define the components for TRIM-POMDP as follows:

State space (S): As shown in Figure 4, we categorize the state space into three correctness classes: S_0 (all prior steps correct and current step correct), S_1 (at least one prior step incorrect), and S_2 (prior steps correct but current step incorrect), along with a terminal absorbing state S_{ter}. Each state is augmented with the step index t and the token count c_t of the current step $\mathbf{y}_t$ within the trace $\mathbf{y}_{1:t}$.

Observation Space (Ω): The observation space (noisy state observations) is defined as the set of aggregated features $\tilde{\mathbf{f}}_{1:t} = \big(r_t,\ \min(\mathbf{r}_{1:t-1}),\ c_t,\ t\big)$, identical to the state features used in TRIM-Agg. These observations help us obtain a probability distribution over the state space S.

Action Space (A): Similar to prior router policies, the action set is $A = \{\texttt{continue}, \texttt{regenerate}\}$.

Transition Function (T): The transition dynamics capture the accuracy of M_s and M_w, as well as transitions into the terminal state S_{ter}. Formally:

$$\begin{aligned}
\mathcal{T}(s' \in S_0 \mid s \in S_0 \cup S_2, a = \texttt{regenerate}) &= p_s \quad \text{(next step accuracy of } M_s) \\
\mathcal{T}(s' \in S_0 \mid s \in S_0, a = \texttt{continue}) &= p_w \quad \text{(next step accuracy of } M_w) \\
\mathcal{T}(s' \in S_1 \mid s \in S_2, a = \texttt{continue}) &= 1 \\
\mathcal{T}(s \in S_1 \mid a \in A, s' \in S_1) &= 1 \quad \text{(irrecoverability assumption)}
\end{aligned}$$

Observation Function ($\mathcal{O}$): The observation function is a mapping from $s \in S$ to the probability distribution over the observation space Ω. The probabilities $P(o|s)$, give us the likelihood of observing $o \in \Omega$ (i.e., $\tilde{\mathbf{f}}_{1:t}$) given state $s \in S$.. This corresponds to modeling the distribution of PRM scores conditioned on each state class, which can be learned from process supervision datasets with step-level annotations (e.g., PRM800K).

Reward Function (R): Invoking the strong model incurs a cost proportional to the number of tokens generated, i.e., $R(s, a = \texttt{regenerate}, s') = -\lambda \cdot |M_s(\mathbf{y}_{1:t-1})|$. . Any transition into the terminal state S_{ter} yields the task reward $\mathbf{R}$, if and only if the final state corresponds to a correct solution (i.e., $s \in S_0$).

TRIM-POMDP can be viewed as an extension of Aggarwal et al. (2023) to the setting of multi-step reasoning. In their official implementation, `Automix` employs a greedy approximation to the POMDP, , which is sufficient for task-level routing since it is a single-step decision problem (horizon of one). In contrast, multi-step reasoning requires planning over long horizons, making a full POMDP formulation more appropriate and motivating the use of sophisticated solvers. Furthermore, our formulation introduces key differences that provide additional flexibility. In `Automix`, self-verification probabilities (observations) are used to obtain estimates of model performance metrics (state features), and thus retraining of the observation function $\mathcal{O}$ is required whenever the model pair (M_s, M_w) changes. However, TRIM-POMDP incorporates model accuracies into the transition function, eliminating the need for retraining the observation model when switching model pairs.

	MATH-500					AIME				
Method	10%	15%	20%	25%	30%	10%	15%	20%	25%	30%
BERT	66.60% (8.5%)	67.84% (16.1%)	69.20% (24.4%)	70.00% (29.3%)	70.40% (31.7%)	13.06% (9.7%)	14.28% (16.8%)	15.56% (24.4%)	17.22% (34.1%)	18.37% (40.9%)
MF	68.46% (19.9%)	70.35% (31.4%)	71.40% (37.8%)	71.86% (40.6%)	72.20% (42.7%)	14.94% (20.7%)	16.39% (29.3%)	17.01% (32.9%)	17.43% (35.4%)	18.26% (40.2%)
SW Ranking	67.82% (16.0%)	68.20% (18.3%)	68.80% (22.0%)	70.01% (29.3%)	71.14% (36.2%)	13.90% (14.6%)	15.10% (21.7%)	16.18% (28.0%)	17.43% (35.4%)	18.88% (43.9%)
Smoothie	67.20% (12.2%)	67.77% (15.7%)	68.20% (18.3%)	69.35% (25.3%)	70.00% (29.3%)	13.07% (9.8%)	13.90% (14.6%)	15.15% (22.0%)	16.18% (28.0%)	16.60% (30.5%)
AutoMix-PRM	68.88% (22.4%)	70.54% (32.5%)	72.19% (42.7%)	73.78% (52.3%)	75.26% (61.3%)	12.74% (7.8%)	13.85% (14.3%)	14.95% (20.8%)	15.88% (26.3%)	16.80% (31.7%)
TRIM-Thr	74.22% (55.0%)	77.55% (75.3%)	80.44% (93.0%)	80.76% (94.8%)	82.22% (103.8%)	17.46% (35.6%)	18.12% (39.4%)	19.01% (44.7%)	21.24% (57.8%)	23.00% (68.1%)
TRIM-Agg	76.42% (68.4%)	79.94% (89.9%)	82.15% (103.3%)	84.59% (118.3%)	87.04% (133.2%)	19.14% (45.4%)	20.82% (55.3%)	22.87% (67.4%)	23.23% (69.5%)	25.95% (85.5%)
TRIM-POMDP	76.39% (68.2%)	78.75% (82.6%)	81.21% (97.7%)	81.86% (101.6%)	82.51% (105.5%)	18.80% (43.4%)	19.26% (46.1%)	22.50% (65.2%)	25.76% (84.4%)	28.62% (101.2%)

Table 3: **Budgeted-accuracy comparison** of TRIM across the AIME & MATH-500 benchmarks.

	OlympiadBench					Minerva Math				
Method	10%	15%	20%	25%	30%	10%	15%	20%	25%	30%
BERT	29.63% (6.6%)	30.11% (9.3%)	30.83% (13.3%)	31.41% (16.5%)	32.62% (23.3%)	28.68% (8.9%)	29.04% (11.1%)	29.84% (15.9%)	31.02% (23.1%)	31.25% (24.4%)
MF	29.78% (7.4%)	30.96% (14.0%)	31.11% (14.9%)	31.56% (17.4%)	32.59% (23.1%)	29.89% (16.2%)	30.88% (22.2%)	32.35% (31.1%)	32.35% (31.1%)	32.72% (33.3%)
SW Ranking	30.95% (14.0%)	32.44% (22.3%)	33.04% (25.6%)	33.63% (28.9%)	34.07% (31.4%)	28.31% (6.7%)	29.41% (13.3%)	31.40% (25.4%)	31.25% (24.4%)	31.51% (26.0%)
Smoothie	29.63% (6.6%)	29.93% (8.3%)	30.86% (13.5%)	32.30% (21.5%)	33.33% (27.3%)	28.31% (6.7%)	28.68% (8.9%)	29.15% (11.8%)	30.15% (17.8%)	30.51% (20.0%)
AutoMix-PRM	29.76% (7.3%)	31.34% (16.1%)	32.61% (23.3%)	33.83% (30.0%)	34.75% (35.2%)	33.66% (39.0%)	34.86% (46.3%)	36.48% (56.1%)	37.88% (64.5%)	39.54% (74.6%)
TRIM-Thr	31.91% (19.3%)	35.53% (39.5%)	37.22% (48.9%)	40.70% (68.4%)	42.08% (76.0%)	32.80% (33.8%)	35.43% (49.7%)	39.34% (73.3%)	41.81% (88.3%)	42.49% (92.4%)
TRIM-Agg	35.56% (39.7%)	37.74% (51.8%)	39.67% (62.6%)	42.30% (77.3%)	43.00% (81.2%)	35.17% (48.1%)	37.25% (60.7%)	40.25% (78.8%)	41.33% (85.4%)	42.41% (91.9%)

Table 4: Cross-benchmark generalization performance of AIME-trained routers under budgeted-accuracy evaluation.

B Routing Policies Implementation Details

TRIM-Agg Implementation. The router policy is parameterized by a simple MLP policy with two hidden layers (128 units each, Tanh activations), followed by separate actor and critic heads with a learning rate of 1e-4. We train with PPO using a learning rate of 1×10^{-4}, clipping coefficient 0.2, and entropy coefficient 0.01. We use unnormalized advantages, undiscounted rewards, and a generalized advantage estimate with $\lambda = 0.95$. Training is conducted across performance–cost trade-off parameters λ, ranging from 3×10^{-4} to 8×10^{-5} for AIME and from 8×10^{-4} to 3×10^{-4} for MATH, at regular intervals.

TRIM-POMDP Implementation. For TRIM-POMDP, we learn the observation function using a reflected KDE estimator applied to the ProcessBench (Zheng et al., 2024) dataset. Specifically, we evaluate our PRM on step-by-step solutions in the dataset and align its outputs with human-annotated step-level labels. The observation function is trained on Omni-MATH problems, while evaluation is conducted on AIME and GSM8k for MATH-500. Importantly, the only model-specific information required by TRIM-POMDP is the next-step accuracies of the models, which are estimated from the corresponding training sets. For solving the POMDP, we use SARSOP (Kurniawati et al., 2008), which we implement using the `POMDPs.jl` framework (Egorov et al., 2017). We solve the POMDP using SARSOP (Kurniawati et al., 2008), implemented using the `POMDPs.jl` framework (Egorov et al., 2017), with hyperparameters set to the default values provided in `POMDPs.jl`. While SARSOP can in principle handle large observation spaces, it is sensitive to the choice of the initial state distribution. To achieve the best performance, we therefore recompute the policy at every step using the updated belief distribution as the initial state distribution. To improve efficiency, we employ a simple heuristic: the policy is recomputed only if the belief mass on state S_2 (the case where the most recent step is incorrect but prior steps are correct) lies within 0.35–0.40 of the maximum belief state class; otherwise, we default to continuing with M_w (i.e., $a_t = \texttt{continue}$). For all TRIM routing policies, the reasoning trace is truncated to a maximum of 30 steps during both training and inference, and the solution at this cutoff is returned.

C Budgeted Accuracy

In addition to the Cost–Performance Threshold (CPT) and Δ_{IBC} metrics, we report budgeted accuracy in Table 3 and Table 4. Along with performance accuracy at each token budget, we also report the performance gap recovered (PGR) for improved comparability, with PGR percentages shown in parentheses. The columns indicate the normalized percentage of tokens generated by M_s, expressed relative to the total number of tokens that would be produced when running M_s alone ($c(\pi)$). Budgeted accuracy is a standard metric that evaluates each routing method under a fixed compute or token budget. These results enable direct comparison under matched compute budgets and complement our CPT and IBC analyses.