

Rhombus: Incentivizing Coordination in Parallel Thinking through Reinforcement Learning

Anonymous ACL submission

Abstract

Parallel thinking offers a promising avenue for scaling test-time compute in Large Language Models (LLMs), enabling them to explore diverse solution paths simultaneously before aggregating them into a final answer. However, coordinating the exploration and aggregation stages remains challenging, as simple aggregation techniques often incur information loss, failing to preserve the subtle, decision-relevant signals generated during exploration. To overcome this, we propose Rhombus, a parallel thinking framework that explicitly incentivizes coordination between components via end-to-end reinforcement learning. Rhombus employs multiple parallel Proposers to generate compact, decision-focused reasoning cues and a central Synthesizer to integrate them into final predictions, utilizing co-training under a shared task reward to align their interaction. Across challenging mathematical reasoning benchmarks, Rhombus improves accuracy by 6.0% over long chain-of-thought baselines while reducing wall-clock latency by 39.4% under matched token budgets. Our work demonstrates that explicit communication optimization is essential for realizing the accuracy and efficiency gains of parallel reasoning.

1 Introduction

Scaling test-time compute has emerged as a pivotal frontier for enhancing the reasoning capabilities of Large Language Models (LLMs) (Jaech et al., 2024; Guo et al., 2025b). The prevailing paradigm, exemplified by Long Chain-of-Thought (CoT) (Wei et al., 2022) prompting, primarily scales compute along the axis of *depth*: a single model is tasked with generating an extended, sequential reasoning trajectory. While effective, this serial approach is inherently brittle, as local errors committed early in the chain often accumulate without correction (Ghosal et al., 2025; Cuadron et al., 2025; Sui et al., 2025), and the computational cost scales quadrati-

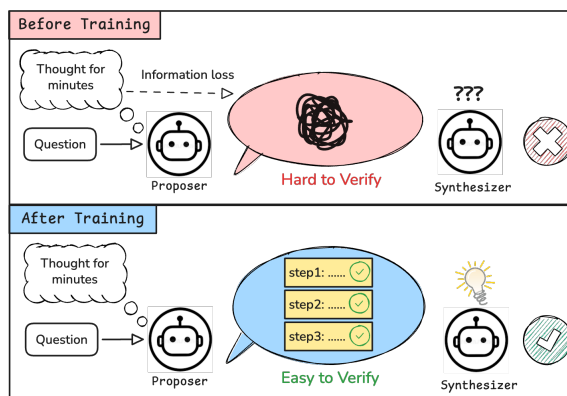


Figure 1: **Impact of Communication.** While uncoordinated agents submit implicit proposals that bury critical reasoning steps and lead to Synthesizer failure due to missing information, Rhombus optimizes the communication to generate explicit, verifiable cues that enable the Synthesizer to accurately aggregate the prediction.

cally with reasoning depth. To mitigate these limitations, *parallel thinking* offers a compelling alternative: decoupling the reasoning process into concurrent exploration followed by centralized aggregation (Wen et al., 2025; Zhao et al., 2025).

However, the efficacy of parallel thinking is currently constrained by a fundamental **coordination gap**. In this paradigm, which typically consists of multiple *Proposers* exploring diverse solution paths and a *Synthesizer* aggregating the results, the two stages are traditionally treated as disjoint processes. Proposers generate reasoning proposals independently, unaware of the downstream Synthesizer’s requirements. Consequently, aggregation relies on heuristic methods that incur severe information loss. As illustrated in Figure 1, uncoordinated Proposers submit *implicit, unstructured proposals*, obscuring critical reasoning steps. In such decoupled systems, these subtle, decision-relevant signals are frequently lost during the hand-off, preventing the Synthesizer from accurately verifying and aggregating the results.

To scale test-time compute effectively, we reformulate parallel reasoning as a communication optimization problem. We posit that simply increasing the breadth of exploration is insufficient, as uncoordinated agents lack the intrinsic capability to format reasoning for downstream aggregation. Consequently, employing Reinforcement Learning to explicitly optimize the communication channel is essential to minimize information loss and realize the theoretical gains of parallel computation.

To this end, we introduce **Rhombus**, a parallel thinking framework designed to incentivize coordination through end-to-end reinforcement learning. Rhombus unifies the inference pipeline into a coherent "propose-and-synthesize" architecture, where multiple parallel Proposers generate compact, decision-focused reasoning cues for a central Synthesizer. Crucially, these components are co-adapted via joint optimization rather than prompted in isolation. To resolve the credit assignment challenge and drive effective communication, we design a hybrid reward structure that incentivizes Proposers for both individual correctness and collective system success. Supported by a role-level advantage normalization scheme that balances reward scales to stabilize training, this joint optimization process aligns the Proposers with the Synthesizer, teaching them to structure their reasoning into explicit, high-signal proposals.

Contributions.

1. We highlight information loss in disjoint exploration-aggregation pipelines as a significant challenge for parallel reasoning. To address this, we propose Rhombus, which reformulates parallel thinking as a communication optimization problem to minimize the coordination gap through a learned interface.
2. We develop an end-to-end reinforcement learning algorithm that co-adapts Proposer and Synthesizer. We address the credit assignment challenge via a hybrid reward structure that incentivizes cooperative behavior, employing Role-Level Advantage Normalization to ensure stable multi-role training.
3. We empirically demonstrate superior efficiency in scaling test-time compute. Rhombus outperforms budget-matched LongCoT baselines by 6.0% in accuracy and 39.4% in latency, while demonstrating robust generaliza-

tion to larger parallel ensembles at inference time.

2 Related Work

2.1 Reinforcement Learning for Reasoning

Reinforcement learning with outcome-based rewards has become the standard for enhancing LLM reasoning, primarily by optimizing Long Chain-of-Thought (LongCoT). This dominant paradigm trains models to generate single, extended trajectories, achieving strong results in math (Yu et al., 2025; Hu et al., 2025; Luo et al., 2025b), coding (Luo et al., 2025a), and logic (Stojanovski et al., 2025). Recent advancements have further improved the efficacy and scaling of these algorithms (Zheng et al., 2025a; Chen et al., 2025; Wang et al., 2025), yet they largely adhere to the single-path assumption. Rhombus fundamentally diverges from this serial constraint. Instead of refining a sequential chain, we apply RL to optimize a parallel inference system. By training the model to coordinate parallel exploration and aggregation, we achieve superior accuracy and latency compared to standard LongCoT baselines under identical token budgets.

2.2 Parallel Thinking

Parallel thinking strategies enhance LLM reasoning by exploring multiple solution paths concurrently prior to final aggregation. Early approaches framed this as structured search (Yao et al., 2023; Wang et al., 2024), typically relying on heuristic consensus mechanisms (Wang et al., 2022; Li et al., 2022a). Recent adaptations extend this to reasoning models via parallel decoding (Yang et al., 2025; Zheng et al., 2025b) or multi-agent architectures (Zhao et al., 2025; Estornell et al., 2025). Rhombus differentiates itself by targeting the coordination gap and information loss inherent in these unaligned processes. Unlike prior works that treat generation and aggregation as disjoint phases, we employ end-to-end reinforcement learning to align the communication interface. This ensures intermediate signals are explicitly optimized for synthesis, allowing the system to robustly surpass serial baselines through fully learned coordination.

3 Methodology

In this section, we describe how Rhombus optimizes parallel thinking by treating the generation and aggregation of reasoning trajectories as a single, end-to-end trainable process. Section 3.1 for-

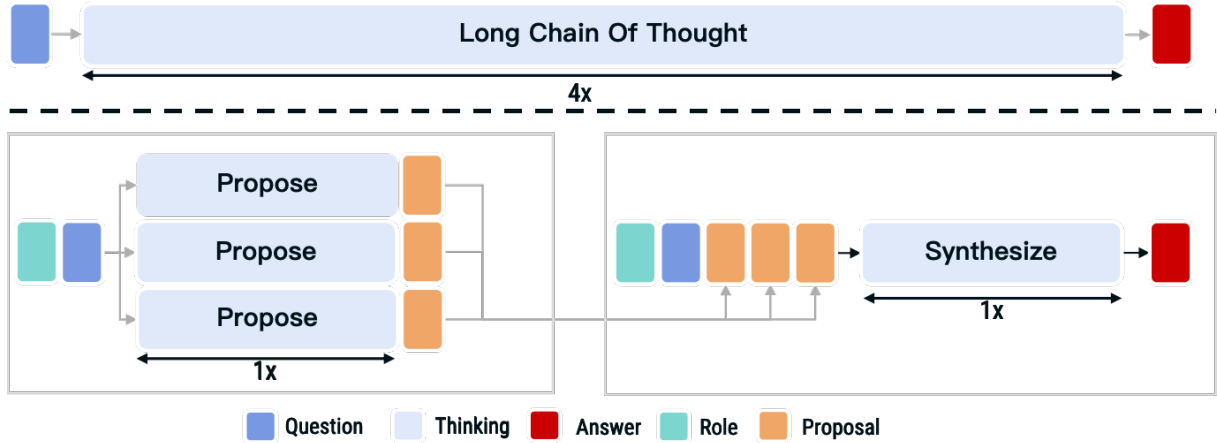


Figure 2: **Overview of Rhombus versus Long Chain-of-Thought (LongCoT)**. The top section illustrates the serial LongCoT baseline, which relies on a single, monolithic reasoning trajectory. The bottom section depicts Rhombus, which decouples inference into *Stage 1 (Exploration)*, where parallel Proposers generate diverse solution paths to cover the reasoning space, and *Stage 2 (Aggregation)*, where a central Synthesizer unifies these proposals. By replacing one extended chain with multiple shorter, concurrent segments, this architecture mitigates the quadratic cost of self-attention, significantly reducing inference latency.

malizes the problem setting and presents the architecture: multiple parallel Proposers generate compact, decision-focused reasoning cues, which are then integrated by a central Synthesizer to produce the final answer. Section 3.2 details the training methodology, specifying the reinforcement learning objective, the credit assignment strategy, and the joint optimization procedure used to align the Proposers and Synthesizer for maximum downstream accuracy.

3.1 Framework Overview

Rhombus operates as a two-stage parallel reasoning framework designed to maximize the efficacy of test-time compute. Given a question q with a ground-truth answer a_{gt} , the system’s objective is to produce a final answer a_{final} via a process of parallel trajectory exploration followed by centralized integration.

Architecture. The framework is powered by a single, shared backbone language model, π_θ . The model operates in one of two distinct modes, controlled by specific system prompts: $r \in \{\text{proposer, synthesizer}\}$. The inference workflow proceeds sequentially: a parallel exploration phase involving multiple Proposers, followed by a single aggregation phase by the Synthesizer. The complete workflow is illustrated in Figure 2, with specific prompts provided in Table 3.

Proposer. The exploration stage employs N parallel Proposer instances. To ensure coverage of

the solution space, each Proposer is executed with the same prompt but utilizes stochastic sampling. Each instance i generates a distinct reasoning path, producing a raw reasoning trace t_i and a specific output proposal p_i .

$$(t_i, p_i) \leftarrow \pi_\theta(q; r = \text{proposer})$$

The proposal p_i functions as a compressed reasoning cue, concluding with a preliminary answer a_i . By explicitly distilling the raw trace into this compact format, the agent filters out the accumulated errors often found in extended reasoning chains, producing a high-fidelity signal optimized for downstream verification and synthesis.

Synthesizer. The Synthesizer functions as the central aggregation module. Its role is to process the set of generated cues $\{p_1, \dots, p_N\}$ to derive the final prediction. Crucially, the Synthesizer does not access the full reasoning traces (t_i); it consumes only the extracted proposals. To enforce a bounded context window and prioritize high-signal information, we apply a strict protocol: proposals are extracted from the raw Proposer output and truncated to a maximum length L_p , with invalid outputs excluded from the set. This structured interface ensures the Synthesizer operates on a consistent, noise-reduced input, allowing it to effectively adjudicate between conflicting signals and consolidate valid reasoning into a final answer, a_{final} .

$$a_{final} \leftarrow \pi_\theta(q, \{p_i\}; r = \text{synthesizer})$$

Inference Efficiency and Dynamics. We define the computational cost in terms of the total token budget $B_{system} = N \cdot L_p + L_s$, where L_p and L_s represent the maximum token limits for the Proposer and Synthesizer, respectively. We align this budget with a serial baseline ($B_{LongCoT}$) to ensure fair comparison. Despite using an identical total number of tokens, Rhombus achieves significantly faster inference through two structural advantages. First, due to the quadratic $O(L^2)$ complexity of self-attention, processing multiple short sequences is computationally cheaper than processing a single monolithic sequence. This reduces the total calculation load even before considering hardware parallelism. Second, given sufficient compute resources, the Proposer instances execute concurrently. This parallelization reduces the end-to-end wall-clock latency to the sum of the individual stages, $T_{system} \approx T(L_p) + T(L_s)$, effectively decoupling the user’s wait time from the total reasoning budget.

3.2 Co-Training with Reinforcement Learning

We co-train the Proposer and Synthesizer end-to-end to explicitly optimize their interaction. Our goal is to adapt the shared backbone π_θ to generate Proposals that are maximally useful for the Synthesizer, and to train the Synthesizer to effectively integrate these cues. We achieve this via a joint optimization framework guided by a shared task reward.

3.2.1 Reinforcement Learning Objective

We adapt Group Relative Policy Optimization (GRPO) (Shao et al., 2024b) to co-optimize the Proposer and Synthesizer. For a given question q , we sample G trajectories. Since each trajectory contains N parallel proposals and one final synthesis, the system generates a total of $G \times (N + 1)$ distinct sequences per optimization step. We treat all sequences as trainable outputs under the shared policy π_θ . The core objective is to maximize:

$$\mathcal{J}(\theta) = \mathbb{E}_{q \sim P(Q)} \frac{1}{|\mathcal{O}_q|} \sum_{o \in \mathcal{O}_q} \sum_{t=1}^{|o|} \left[\min \left(\rho_t(\theta) \hat{A}_t, \text{clip}(\rho_t(\theta), 1 \pm \epsilon) \hat{A}_t \right) \right] \quad (1)$$

where \mathcal{O}_q represents the set of all generated sequences for question q , and ρ_t is the probability ratio between the new and old policies.

Role-Level Advantage Normalization. A standard GRPO implementation would normalize advantages across the entire set \mathcal{O}_q . However, Proposers and Synthesizers operate in different state spaces with distinct reward scales. To ensure stable joint training, we decouple the normalization process by constructing two distinct groups for each question:

1. **Proposer Group (\mathcal{G}_P):** Aggregates all $G \times N$ proposal sequences generated across the G trajectories.
2. **Synthesizer Group (\mathcal{G}_S):** Aggregates all G synthesis sequences.

We compute the mean μ_r and standard deviation σ_r independently for each group $r \in \{\mathcal{G}_P, \mathcal{G}_S\}$. The normalized advantage $\hat{A}_{r,i}$ for the i -th sequence in group r is then calculated as:

$$\hat{A}_{r,i} = \frac{A_{r,i} - \mu_r}{\sigma_r + \epsilon_{norm}} \quad (2)$$

This ensures that the Proposer and Synthesizer updates are scaled relative to their own peers, preventing one role from dominating the gradient.

3.2.2 Reward Modeling

Credit Assignment Strategy. A central challenge is balancing a Proposer’s *local correctness* with its *downstream utility*. Relying solely on individual accuracy causes agents to consume their token budget for internal reasoning, producing terse outputs that starve the Synthesizer of context. Conversely, relying purely on global system rewards generates misleading feedback: an agent receives credit for the team’s success even if its own specific answer was wrong. This noisy signal obscures the link between action and reward, significantly slowing down training. To resolve this, we adopt a hybrid reward structure that combines individual correctness with system correctness. This balance ensures agents remain competent reasoners while explicitly optimizing for communication with the Synthesizer.

Proposer Reward. To balance local accuracy with global system performance, each Proposer i receives a composite reward R_{p_i} :

$$R_{p_i} = c \cdot R_i^C + (1 - c) \cdot R_{sys}^C \cdot R_i^F \quad (3)$$

where:

- R_i^C is the *local correctness reward* (1 if $a_i = a_{gt}$, else 0). This dense signal ensures the model retains basic problem-solving capabilities.
- R_{sys}^C is the *system correctness reward* (1 if $a_{final} = a_{gt}$, else 0). This shared signal propagates the Synthesizer’s success back to the Proposers, encouraging the generation of cues that actually aid integration.
- R_i^F is a binary *format reward*. It acts as a gating function: Proposers only share in the system credit if they successfully adhere to the extractable proposal format.
- $c \in [0, 1]$ controls the trade-off between optimizing for independent accuracy versus downstream helpfulness.

Synthesizer Reward. The Synthesizer is responsible solely for the final aggregation. Its reward R_s is determined entirely by the system output:

$$R_s = R_{sys}^C \quad (4)$$

This asymmetric structure aligns the distinct roles: Proposers maximize a mixture of validity and utility, while the Synthesizer focuses exclusively on accurate aggregation.

4 Experiments

4.1 Experimental Setup

Configuration. We evaluate Rhombus using two open-weight architectures. Our primary analysis employs Qwen3-4B-Instruct-2507 (Team, 2025), optimized on a 22k-sample subset of the POLARIS dataset¹. To assess cross-model generalization, we additionally train DeepSeek-R1-Distilled-1.5B (Guo et al., 2025b) on the DeepScaleR dataset². In both settings, a single shared checkpoint executes both Proposer and Synthesizer roles, conditioned via role-specific system prompts (Table 3). The system is optimized end-to-end using on-policy Group Relative Policy Optimization (GRPO) (Shao et al., 2024a) with $N = 3$ parallel Proposers. Full implementation details, including token budgets and hyperparameters, are provided in Appendix A.

¹<https://github.com/ChenxinAn-fdu/POLARIS/blob/main/parquet/stage1/qwen3-4b-s1.parquet>

²<https://huggingface.co/datasets/agentica-org/DeepScaleR-Preview-Dataset>

Evaluation. To track training dynamics, we monitor the Macro Average Score across AIME 2024 (MAA, 2024), AIME 2025 (MAA, 2025), and OlympiadBench (He et al., 2024). For final evaluation, we report performance on these three benchmarks plus BeyondAIME ([ByteDance-Seed], 2025). We utilize the Pass@k metric (Chen et al., 2021), employing the unbiased estimator to calculate the probability of generating at least one correct solution:

$$\text{Pass@k} = 1 - \frac{\binom{n-c}{k}}{\binom{n}{k}} \quad (5)$$

where n is the total number of samples and c is the number of correct solutions. Answer correctness is automatically verified via the Math-Verify library³.

4.2 Main Results

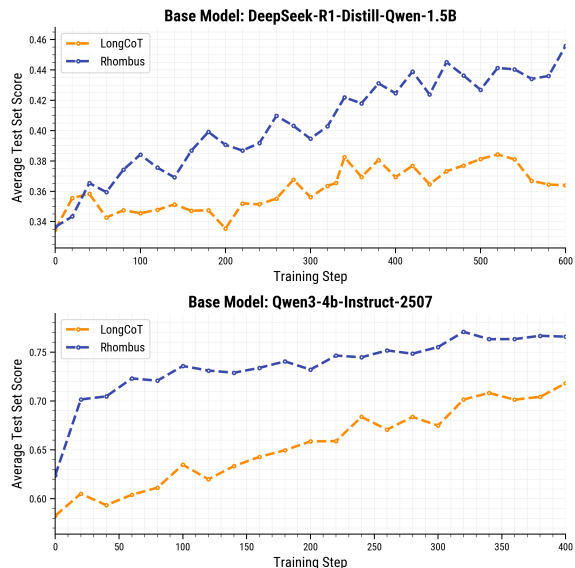


Figure 3: Comparison of Rhombus and LongCoT macro-average scores throughout training.

Baselines and Comparison Setup. We evaluate Rhombus against a Long-Chain-of-Thought (Long-CoT) baseline. To ensure a fair comparison, all experiments are conducted under a matched computational budget. Specifically, the total token budget of the Rhombus system ($N \times L_p + L_s$) is constrained to be equal to the token budget of the corresponding serial LongCoT baseline. This allows us to isolate the architectural benefits of our framework from the effects of simply using more compute.

³<https://github.com/huggingface/Math-Verify>

Model	AIME 2024		AIME 2025		BeyondAIME		Olympiad	Average
	Pass@1	Pass@8	Pass@1	Pass@8	Pass@1	Pass@8	Pass@1	Pass@1
<i>Base Model: DeepSeek-R1-Distilled-1.5B</i>								
Base	27.2	58.1	22.8	42.3	8.6	25.6	49.8	27.1
LongCoT	32.5	59.8	25.4	43.4	10.8	28.2	54.2	30.7
Rhombus	41.0	67.9	29.0	43.4	14.2	31.4	60.4	36.2
<i>Base Model: Qwen3-4B-instruct-2507</i>								
Base	62.2	84.0	47.0	69.0	30.8	53.3	72.1	53.0
LongCoT	70.7	89.4	64.1	79.1	43.3	60.4	76.1	63.6
Rhombus	80.6	90.5	71.3	79.5	48.2	60.6	78.5	69.6

Table 1: Performance comparison between Rhombus and baselines on mathematical benchmarks.

Main Results. As detailed in Table 1, Rhombus consistently outperforms the LongCoT baseline across all evaluated benchmarks and model scales. On our primary model, Qwen3-4B-Instruct-2507, Rhombus achieves a macro-average Pass@1 of 69.6%, a **+6.0%** absolute improvement over an equivalent LongCoT-RL baseline. This advantage holds on the smaller DeepSeek-R1-Distilled-1.5B model, where Rhombus obtains a **+5.5%** absolute gain. These results validate that our approach is a more effective use of a fixed compute budget than serial generation, and this performance gap is maintained consistently throughout the training process (Figure 3).

Generalization to Code Generation. To assess the versatility of Rhombus beyond mathematical reasoning, we extend our evaluation to code generation using the same Qwen3-4B-Instruct-2507 backbone and training recipe. We train on data sourced from inclusionAI (2025) and Luo et al. (2025a) and evaluate on the validation split of the PRIME benchmark (Cui et al., 2025), which encompasses diverse algorithmic tasks from APPS (Hendrycks et al., 2021), CodeContests (Li et al., 2022b), Codeforces (MatrixStudio, 2024), and TACO (Li et al., 2023). As shown in Table 2, Rhombus consistently surpasses the LongCoT baseline across all four benchmarks, achieving a macro-average accuracy gain of **2.4%**. This confirms that our parallel reasoning mechanism is domain-agnostic and generalizes robustly to algorithmic programming challenges.

Inference Efficiency. Rhombus accelerates inference by decomposing single long trajectories into multiple shorter segments, thereby mitigating

Model	APPS	CodeContests	Codeforces	TACO	Average
	Pass@1	Pass@1	Pass@1	Pass@1	Pass@1
<i>Base Model: Qwen3-4B-instruct-2507</i>					
Base	45.0	42.1	38.2	24.3	37.4
LongCoT	58.4	53.9	51.6	32.6	49.1
Rhombus	59.1	55.5	56.3	34.9	51.5

Table 2: Performance comparison between Rhombus and baselines on code benchmarks.

quadratic attention costs. As illustrated in Figure 4, benchmarks on a single H20 GPU demonstrate that Rhombus reduces macro-average inference time from 129.17s to 78.26s. This constitutes a **39.4% speedup** over the budget-matched LongCoT baseline. We provide the full experimental setup and analysis in Appendix B.

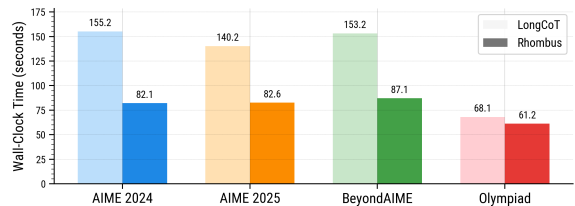


Figure 4: **End-to-End Latency on a Single H20 GPU.** Rhombus achieves a 39.4% speedup over the budget-matched LongCoT baseline by reducing the sequence length overhead.

4.3 Scaling Test-Time Compute

A defining characteristic of Rhombus is its ability to modulate the computational budget at test time by adjusting the number of Proposer agents (N). Figure 5 illustrates the performance scaling across our benchmark suite as we increase N from 0 to 8. Here, the $N = 0$ setting denotes the model answering directly without parallel proposals.

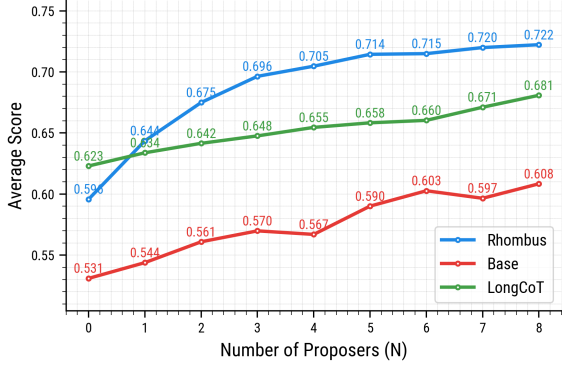


Figure 5: Test-time scaling dynamics. We plot the average Pass@1 accuracy against the number of parallel Proposers (N) across the mathematical benchmarks.

Generalization to Varying Agent Counts. We first evaluate the scalability of Rhombus in isolation. The results demonstrate a consistent performance improvement as the number of Proposers increases from 1 to 8, with no signs of saturation. Crucially, this scaling holds even though the system was trained with a fixed configuration of $N = 3$. This indicates that the Synthesizer has not overfitted to a specific input size but has instead learned a robust integration mechanism that generalizes effectively to larger ensembles at inference time.

Comparative Scaling Efficiency. To rigorously assess the efficiency of this scaling, we compare our specialized Rhombus model against the LongCoT baseline, employing the Rhombus framework for both. We select a baseline checkpoint with comparable initial performance, though slightly stronger than our model: at $N = 0$, the LongCoT baseline outperforms Rhombus by approximately 2.7%. However, despite this initial disadvantage, Rhombus exhibits a significantly superior scaling coefficient. As the number of parallel agents increases, our framework rapidly overtakes the baseline, ultimately achieving a +4.1% improvement at $N = 8$. This crossover confirms that the performance gains are driven by the Rhombus architecture’s ability to effectively aggregate parallel computation, rather than the underlying strength of the base model. Finally, we demonstrate in Appendix C that Rhombus significantly outperforms a Majority Voting baseline, confirming the efficacy of learned synthesis over heuristic consensus.

4.4 Analysis and Ablations

We conduct ablation studies to verify that Rhombus achieves its gains by explicitly optimizing the

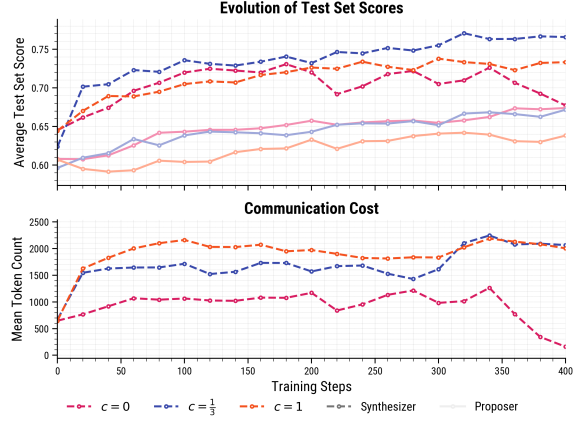


Figure 6: Training dynamics tracking Proposer accuracy, Synthesizer accuracy, and Communication Cost (Proposal Length).

communication between exploration and aggregation. We demonstrate that simply having strong reasoners is insufficient; the system must minimize information loss during the hand-off.

Analysis of Credit Assignment Strategies. We analyze how the reward weighting hyperparameter c impacts the system’s ability to learn effective communication. This parameter balances the Proposer’s internal reasoning quality against its downstream utility. To quantify the information flow, we track the **Communication Cost**, defined as the total accumulated token count of all proposals transmitted to the Synthesizer. We compare three distinct regimes:

- **Individual-Only** ($c = 1$) rewards Proposers solely for local correctness, mimicking independent sampling.
- **System-Only** ($c = 0$) rewards Proposers solely for the final system outcome, ignoring intrinsic validity.
- **Balanced** ($c = 1/3$) represents the Rhombus setting, incentivizing both individual and system correctness.

As shown in Figure 6, the **Individual-Only** setting reveals a critical failure mode known as Communication Collapse. We observe that the Communication Cost drops precipitously, indicating that agents consume their token budget for internal reasoning while producing terse outputs. This behavior starves the Synthesizer of context, resulting in severe information loss at the interface.

In contrast, the **System-Only** setting suffers from Misleading Feedback. Since agents receive credit for the system’s success even if their specific answer is wrong, the learning signal becomes noisy. This ambiguity reduces training efficiency and causes Proposer accuracy to stagnate, as agents struggle to distinguish between effective reasoning and merely benefiting from the group’s success. Only the Balanced approach sustains effective information transfer. It ensures Proposers remain competent reasoners while maintaining a sufficient Communication Cost, explicitly encoding their reasoning into the detailed cues required for accurate aggregation.

Unlocking Exploration Potential. To demonstrate that Rhombus achieves its gains by optimizing the communication between Proposers and Synthesizers, we introduce a metric serve as a proxy for **Coordination Gap** (Δ). This metric quantifies communication efficiency by measuring the divergence between the exploration potential of the Proposers and the realized performance of the final system. We define exploration potential as the Proposer’s Pass@3 metric (the probability that a correct solution exists in the pool) and realized performance as the system’s final Pass@1 accuracy. The gap is defined as:

$$\Delta = \text{Pass}@3_{\text{Proposer}} - \text{Pass}@1_{\text{System}} \quad (6)$$

A minimized Δ indicates an efficient integration process where the Synthesizer successfully retrieves and utilizes the correct reasoning present in the candidate pool.

We compare the fully co-adapted Rhombus system against a Baseline configuration that pairs standard LongCoT-trained Proposers with the fixed Rhombus Synthesizer. To ensure a rigorous evaluation, we selected a LongCoT model with exploration potential comparable to the Rhombus model (72.1% vs. 70.3%).

As shown in Figure 7, the Baseline suffers from a significant coordination gap ($\Delta = 4.6\%$). Although valid solutions exist in the pool with high probability (72.1%), the unoptimized communication prevent effective aggregation, limiting realized accuracy to 67.5%. In contrast, Rhombus minimizes this gap to a negligible $\Delta = 0.7\%$. By bridging the communication disconnect, Rhombus achieves a final accuracy of 69.6%, representing a **+2.1%** system-level gain over the Baseline despite starting with lower intrinsic potential. This

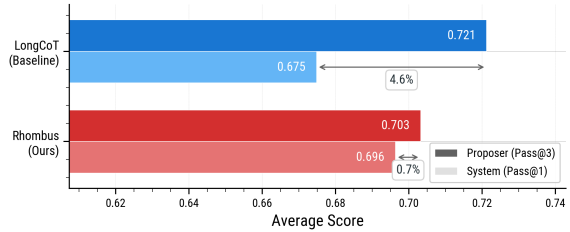


Figure 7: Analysis of the Coordination Gap (Δ). We compare exploration potential (Pass@3) vs. realized accuracy (Pass@1). Rhombus minimizes this gap ($\Delta = 0.7\%$), achieving superior final accuracy (69.6%).

demonstrates that jointly optimizing the communication interface is a prerequisite for unlocking the full potential of parallel thinking, bridging the gap between latent generation capability and realized system performance.

5 Limitations

First, the centralized Synthesizer imposes a context window bottleneck, potentially limiting continuous scaling as the number of parallel agents increases significantly. Second, our joint optimization framework relies on verifiable outcome-based rewards, which currently restricts applicability to domains with objective ground truth, leaving open-ended tasks for future exploration.

6 Conclusion

We presented Rhombus, a parallel reasoning framework designed to resolve the critical information loss bottleneck inherent in uncoordinated aggregation. To bridge the gap between uncoordinated parallel thinking and coherent reasoning, Rhombus employs reinforcement learning to optimize the interaction between Proposers and a Synthesizer, ensuring diverse trajectories are effectively coordinated and integrated. Empirically, this alignment yields a **+6.0%** accuracy improvement over budget-matched LongCoT baselines. Furthermore, Rhombus demonstrates robust test-time scaling capabilities, allowing users to reliably unlock greater reasoning performance by increasing parallel breadth. Future work will focus on enhancing the diversity of the Proposer’s exploration strategies.

References

[ByteDance-Seed]. 2025. Beyondaime: Advancing math reasoning evaluation beyond high school olympiads. [<https://huggingface.co/>]

554	datasets/ByteDance-Seed/BeyondAIME] (https://huggingface.co/datasets/ByteDance-Seed/BeyondAIME).	
555		
556		
557	Aili Chen, Aonian Li, Bangwei Gong, Binyang Jiang,	
558	Bo Fei, Bo Yang, Boji Shan, Changqing Yu, Chao	
559	Wang, Cheng Zhu, Chengjun Xiao, Chengyu Du,	
560	Chi Zhang, Chu Qiao, Chunhao Zhang, Chunhui	
561	Du, Congchao Guo, Da Chen, Deming Ding, and	
562	80 others. 2025. Minimax-m1: Scaling test-time	
563	compute efficiently with lightning attention . <i>ArXiv</i> ,	
564	abs/2506.13585 .	
565	Mark Chen, Jerry Tworek, Heewoo Jun, Qiming	
566	Yuan, Henrique Pondé, Jared Kaplan, Harrison Ed-	
567	wards, Yura Burda, Nicholas Joseph, Greg Brockman,	
568	Alex Ray, Raul Puri, Gretchen Krueger, Michael	
569	Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin,	
570	Brooke Chan, Scott Gray, and 34 others. 2021. Eval-	
571	uating large language models trained on code . <i>ArXiv</i> ,	
572	abs/2107.03374 .	
573	Alejandro Cuadron, Dacheng Li, Wenjie Ma, Xingyao	
574	Wang, Yichuan Wang, Siyuan Zhuang, Shu Liu,	
575	Luis Gaspar Schroeder, Tian Xia, Huanzhi Mao, and	
576	1 others. 2025. The danger of overthinking: Exam-	
577	ining the reasoning-action dilemma in agentic tasks.	
578	<i>arXiv preprint arXiv:2502.08235</i> .	
579	Ganqu Cui, Lifan Yuan, Zefan Wang, Hanbin Wang,	
580	Wendi Li, Bingxiang He, Yuchen Fan, Tianyu Yu,	
581	Qixin Xu, Weize Chen, and 1 others. 2025. Pro-	
582	cess reinforcement through implicit rewards. <i>arXiv</i>	
583	<i>preprint arXiv:2502.01456</i> .	
584	Andrew Estornell, Jean-Francois Ton, Muham-	
585	mad Faaiz Taufiq, and Hang Li. 2025. How to train	
586	a leader: Hierarchical reasoning in multi-agent llms.	
587	<i>arXiv preprint arXiv:2507.08960</i> .	
588	Soumya Suvra Ghosal, Souradip Chakraborty, Avinash	
589	Reddy, Yifu Lu, Mengdi Wang, Dinesh Manocha,	
590	Furong Huang, Mohammad Ghavamzadeh, and Am-	
591	rit Singh Bedi. 2025. Does thinking more always	
592	help? understanding test-time scaling in reasoning	
593	models. <i>arXiv preprint arXiv:2506.04210</i> .	
594	Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song,	
595	Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang,	
596	Shirong Ma, Xiao Bi, and 1 others. 2025a. Deepseek-	
597	r1 incentivizes reasoning in llms through reinforce-	
598	ment learning. <i>Nature</i> , 645(8081):633–638.	
599	Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song,	
600	Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong	
601	Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025b.	
602	Deepseek-r1: Incentivizing reasoning capability in	
603	llms via reinforcement learning. <i>arXiv preprint</i>	
604	<i>arXiv:2501.12948</i> .	
605	Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu,	
606	Zhen Leng Thai, Junhao Shen, Jinyi Hu, Xu Han, Yu-	
607	jie Huang, Yuxiang Zhang, Jie Liu, Lei Qi, Zhiyuan	
608	Liu, and Maosong Sun. 2024. Olympiadbench:	
609	A challenging benchmark for promoting agi with	
610	olympiad-level bilingual multimodal scientific prob-	
611	lems . <i>Preprint</i> , arXiv:2402.14008.	
	Dan Hendrycks, Steven Basart, Saurav Kadavath, Man-	612
	tas Mazeika, Akul Arora, Ethan Guo, Collin Burns,	613
	Samir Puranik, Horace He, Dawn Song, and Jacob	614
	Steinhardt. 2021. Measuring coding challenge com-	615
	petence with apps . <i>Preprint</i> , arXiv:2105.09938.	616
	Jingcheng Hu, Yinmin Zhang, Qi Han, Daxin Jiang,	617
	Xiangyu Tony Zhang, and Heung yeung Shum. 2025.	618
	Open-reasoner-zero: An open source approach to	619
	scaling up reinforcement learning on the base model .	620
	<i>ArXiv</i> , abs/2503.24290 .	621
	inclusionAI. 2025. Areal-boba-2-rl-code .	622
	Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richard-	623
	son, Ahmed El-Kishky, Aiden Low, Alec Helyar,	624
	Aleksander Madry, Alex Beutel, Alex Carney, and 1	625
	others. 2024. Openai o1 system card. <i>arXiv preprint</i>	626
	<i>arXiv:2412.16720</i> .	627
	Rongao Li, Jie Fu, Bo-Wen Zhang, Tao Huang, Zhihong	628
	Sun, Chen Lyu, Guang Liu, Zhi Jin, and Ge Li. 2023.	629
	Taco: Topics in algorithmic code generation dataset .	630
	<i>Preprint</i> , arXiv:2312.14852.	631
	Yujia Li, David Choi, Junyoung Chung, Nate Kushman,	632
	Julian Schrittwieser, Rémi Leblond, Tom Eccles,	633
	James Keeling, Felix Gimeno, Agustin Dal Lago, and	634
	1 others. 2022a. Competition-level code generation	635
	with alphacode. <i>Science</i> , 378(6624):1092–1097.	636
	Yujia Li, David Choi, Junyoung Chung, Nate Kushman,	637
	Julian Schrittwieser, Rémi Leblond, Tom Eccles,	638
	James Keeling, Felix Gimeno, Agustin Dal Lago,	639
	Thomas Hubert, Peter Choy, Cyprien de Mas-	640
	son d’Autume, Igor Babuschkin, Xinyun Chen, Po-	641
	Sen Huang, Johannes Welbl, Sven Gowal, Alexey	642
	Cherepanov, and 7 others. 2022b. Competition-	643
	level code generation with alphacode . <i>Science</i> ,	644
	378(6624):1092–1097.	645
	Michael Luo, Sijun Tan, Roy Huang, Ameen Patel, Al-	646
	pay Ariyak, Qingyang Wu, Xiaoxiang Shi, Rachel	647
	Xin, Colin Cai, Maurice Weber, Ce Zhang, Li Erran	648
	Li, Raluca Ada Popa, and Ion Stoica. 2025a. Deep-	649
	coder: A fully open-source 14b coder at o3-mini	650
	level. Notion Blog.	651
	Michael Luo, Sijun Tan, Justin Wong, Xiaoxiang Shi,	652
	William Y. Tang, Manan Roongta, Colin Cai, Jeffrey	653
	Luo, Li Erran Li, Raluca Ada Popa, and Ion Stoica.	654
	2025b. Deepscaler: Surpassing o1-preview with a	655
	1.5b model by scaling rl. Notion Blog.	656
	MAA. 2024. American invitational mathematics exami-	657
	nation - aime . In <i>American Invitational Mathematics</i>	658
	<i>Examination - AIME 2024</i> .	659
	MAA. 2025. American invitational mathematics exami-	660
	nation - aime . In <i>American Invitational Mathematics</i>	661
	<i>Examination - AIME 2025</i> .	662
	MatrixStudio. 2024. Codeforces-python-submissions .	663

664	Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Jun-Mei Song, Mingchuan Zhang, Y. K. Li, Yu Wu, and Daya Guo. 2024a. Deepseekmath: Pushing the limits of mathematical reasoning in open language models . <i>ArXiv</i> , abs/2402.03300.	721
665		722
666		723
667		724
668		725
669	Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, and 1 others. 2024b. Deepseekmath: Pushing the limits of mathematical reasoning in open language models . <i>arXiv preprint arXiv:2402.03300</i> .	726
670		727
671		728
672		729
673		730
674		731
675	Zafir Stojanovski, Oliver Stanley, Joe Sharratt, Richard Jones, Abdulhakeem Oluwadare Adefioye, Jean Kadour, and Andreas Köpf. 2025. Reasoning gym: Reasoning environments for reinforcement learning with verifiable rewards . <i>ArXiv</i> , abs/2505.24760.	732
676		733
677		734
678		735
679		736
680	Yang Sui, Yu-Neng Chuang, Guanchu Wang, Jiamu Zhang, Tianyi Zhang, Jiayi Yuan, Hongyi Liu, Andrew Wen, Shaochen Zhong, Na Zou, and 1 others. 2025. Stop overthinking: A survey on efficient reasoning for large language models . <i>arXiv preprint arXiv:2503.16419</i> .	737
681		738
682		739
683		740
684		741
685		742
686	Qwen Team. 2025. Qwen3 technical report . <i>Preprint</i> , arXiv:2505.09388.	743
687		744
688	Evan Wang, Federico Cassano, Catherine Wu, Yunfeng Bai, Will Song, Vaskar Nath, Ziwen Han, Sean Hendryx, Summer Yue, and Hugh Zhang. 2024. Planning in natural language improves llm search for code generation . <i>arXiv preprint arXiv:2409.03733</i> .	745
689		746
690		747
691		748
692		
693	Shenzhi Wang, Le Yu, Chang Gao, Chujie Zheng, Shixuan Liu, Rui Lu, Kai Dang, Xionghui Chen, Jianxin Yang, Zhenru Zhang, Yuqiong Liu, An Yang, Andrew Zhao, Yang Yue, Shiji Song, Bowen Yu, Gao Huang, and Junyang Lin. 2025. Beyond the 80/20 rule: High-entropy minority tokens drive effective reinforcement learning for llm reasoning . <i>ArXiv</i> , abs/2506.01939.	
694		
695		
696		
697		
698		
699		
700	Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models . <i>arXiv preprint arXiv:2203.11171</i> .	
701		
702		
703		
704		
705	Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models . <i>Advances in neural information processing systems</i> , 35:24824–24837.	
706		
707		
708		
709		
710		
711	Hao Wen, Yifan Su, Feifei Zhang, Yunxin Liu, Yunhao Liu, Ya-Qin Zhang, and Yuanchun Li. 2025. Parthinker: Native parallel thinking as a new paradigm to scale llm test-time compute . <i>arXiv preprint arXiv:2509.04475</i> .	
712		
713		
714		
715		
716	Xinyu Yang, Yuwei An, Hongyi Liu, Tianqi Chen, and Beidi Chen. 2025. Multiverse: Your language models secretly decide how to parallelize and merge generation . In <i>Advances in Neural Information Processing Systems</i> .	
717		
718		
719		
720		
	Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models . <i>Advances in neural information processing systems</i> , 36:11809–11822.	
	Qiyang Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Wang Zhang, Hang Zhu, and 16 others. 2025. Dapo: An open-source llm reinforcement learning system at scale . <i>ArXiv</i> , abs/2503.14476.	
	Wenting Zhao, Pranjal Aggarwal, Swarnadeep Saha, Asli Celikyilmaz, Jason Weston, and Ilia Kulikov. 2025. The majority is not always right: RL training for solution aggregation . <i>arXiv preprint arXiv:2509.06870</i> .	
	Chujie Zheng, Shixuan Liu, Mingze Li, Xionghui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, Jingren Zhou, and Junyang Lin. 2025a. Group sequence policy optimization . <i>ArXiv</i> , abs/2507.18071.	
	Tong Zheng, Hongming Zhang, Wenhao Yu, Xiaoyang Wang, Xinyu Yang, Runpeng Dai, Rui Liu, Huiwen Bao, Chengsong Huang, Heng Huang, and 1 others. 2025b. Parallel-r1: Towards parallel thinking via reinforcement learning . <i>arXiv preprint arXiv:2509.07980</i> .	

A Implementation Details

Token Budget Allocation. We aligned the computational resources for both **DeepSeek-R1-Distilled-1.5B** and **Qwen3-4B-Instruct-2507** by matching the total inference budget for each model family:

- **DeepSeek-R1-Distilled-1.5B:** We allocated a budget of 8,192 tokens per agent. With $N = 3$ Proposers and 1 Synthesizer, this results in a total system budget of 32,768 tokens, which matches the single-context limit used for its LongCoT baseline.
- **Qwen3-4B-Instruct-2507:** We increased the per-agent budget to 12,288 tokens, resulting in a total system capacity of 49,152 tokens. This expansion was necessary to accommodate the base model’s tendency to generate highly extensive reasoning traces when addressing challenging mathematical benchmarks. For the corresponding LongCoT baseline, we adopted a two-stage training strategy following established practices for long-context optimization (Guo et al., 2025a; Luo et al., 2025b). Specifically, we initiated training with a 32k context window and transitioned to the full context length after 300 steps.

Hyperparameters. Models are optimized via on-policy reinforcement learning without KL regularization, utilizing the AdamW optimizer with a constant learning rate of 1×10^{-6} and $(\beta_1, \beta_2) = (0.9, 0.95)$. We employ a global batch size of 128 with 8 rollouts per prompt. To promote exploration during training, we set the sampling temperature to 1.0 with top-k/p truncation disabled. For final evaluation, we adopt a more stable decoding strategy with temperature 0.6, $top_p = 0.95$, and $top_k = 20$. We estimate performance using $n = 32$ samples for all benchmarks, with the exception of OlympiadBench ($n = 1$).

B Performance Analysis Details

To quantify the efficiency gains of the Rhombus framework, we conducted controlled benchmarks on a single NVIDIA H20 GPU using the vLLM inference engine with a system-level batch size of 1. We recorded both total token consumption (Figure 8) and end-to-end wall-clock latency (Figure 4 in the main text). Rhombus achieves a 39.4% reduction in wall-clock time, despite a higher aggregate

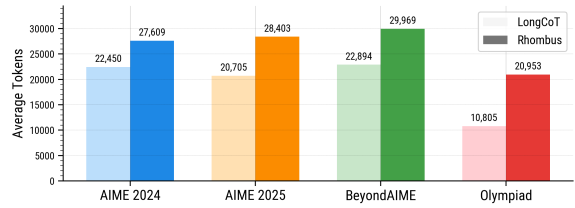


Figure 8: Comparison of token consumption between Rhombus and LongCoT. Although Rhombus generates a higher total volume of tokens, it distributes them across shorter, concurrent contexts, avoiding the high computational cost of long-context generation.

token count. This efficiency stems from distributing reasoning across shorter, parallel chains rather than a single extended sequence. Consequently, Rhombus avoids the latency bottlenecks of long sequential contexts, enabling faster inference even with increased total generation volume.

C Comparison with Majority Voting

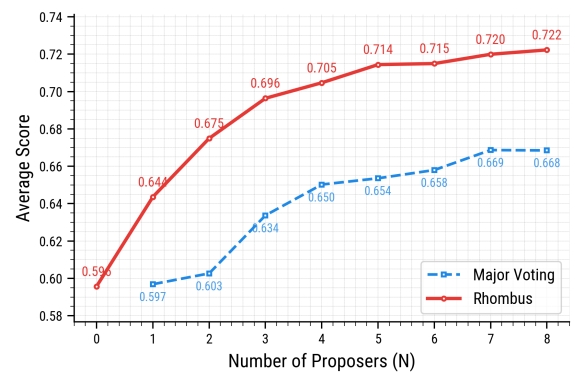


Figure 9: Scaling efficiency of Rhombus versus Majority Voting. We plot Pass@1 accuracy as a function of the number of Proposers (N). Rhombus consistently outperforms the voting baseline, indicating that the Synthesizer extracts signal more effectively than heuristic consensus.

To validate the utility of the learned Synthesizer, we compare Rhombus against a Majority Voting (Self-Consistency) baseline. We implement the baseline by extracting Proposer outputs via the Math-Verify library and selecting the most frequent answer. As shown in Figure 9, Rhombus consistently outperforms Majority Voting across all ensemble sizes. This performance gap confirms that the Synthesizer provides superior integration: instead of discarding the reasoning rationale to simply count votes, it leverages the semantic cues within the proposals to identify correct reasoning,

Role	Prompt
Proposer	<p>Please reason step by step, and put your final answer within $\boxed{\quad}$.</p> <p>Respond using this exact format:</p> <p>### Thought</p> <p>Provide a detailed, step-by-step thought process.</p> <p>### Solution</p> <p>Present a clear step-by-step solution. This should be the "polished" version of your reasoning, formatted for easy understanding.</p> <p>### Final Answer</p> <p>State the final, correct answer.</p>
Synthesizer	<p>Your task is to analyze previous answers to a question and provide one definitive, improved answer.</p> <p>Please reason step by step, and put your final answer within $\boxed{\quad}$.</p> <p>Respond using this exact format:</p> <p>### Thought</p> <p>Provide a detailed, step-by-step thought process.</p> <p>### Final Answer</p> <p>State the final, correct answer.</p>

Table 3: Prompts used for evaluating *Rhombus* with the *Qwen3-4B-Instruct-2507*.

even when it is not the plurality consensus.

Furthermore, Rhombus offers superior generality. Majority Voting relies on rigid, domain-specific normalizers (e.g., math equivalence checkers) which are inapplicable to open-ended tasks. In contrast, Rhombus aggregates via natural language processing, making it a domain-agnostic framework suitable for fields like general QA where rigid answer matching is intractable.

D Qualitative Case Study

Here we present a detailed comparison between the uncoordinated baseline and Rhombus.

Problem Statement

Question: A sequence is defined by the recurrence relation $a_{n+1} = 1 + \frac{1}{1+a_n}$. Given that $a_3 = \frac{41}{29}$, find the value of a_1 .

Baseline (Uncoordinated Proposer)

We are given the recurrence: $a_{n+1} = 1 + \frac{1}{1+a_n}$
 We are given $a_3 = \frac{41}{29}$, and we want a_1 .
 We reverse the recurrence by finding the inverse function. Let $y = 1 + \frac{1}{1+x}$.
 Solving for x :
 $-y - 1 = \frac{1}{1+x} - 1 + x = \frac{1}{y-1} \cdot x = \frac{1}{y-1} - 1 = \frac{2-y}{y-1}$
 So $a_n = \frac{2-a_{n+1}}{a_{n+1}-1}$
 Now compute backwards:
 1. $a_2 = \frac{2-\frac{41}{29}}{\frac{41}{29}-1} = \frac{\frac{17}{29}}{\frac{12}{29}} = \frac{17}{12}$ 2. $a_1 = \frac{2-\frac{17}{12}}{\frac{17}{12}-1} = \frac{\frac{7}{12}}{\frac{5}{12}} = \frac{7}{5}$

Final Answer $\boxed{\frac{7}{5}}$

Analysis: While the answer is correct, the reasoning is unstructured and critical intermediate steps are buried in paragraphs. The Synthesizer receives a noisy signal.

Rhombus

We are given a recurrence:

$$a_{n+1} = 1 + \frac{1}{1+a_n}$$

and that $a_3 = \frac{41}{29}$. We are to find a_1 .
Step 1: Invert the recurrence to express a_n in terms of a_{n+1} .
 Let $y = a_{n+1}$, $x = a_n$. Then:

$$y = 1 + \frac{1}{1+x}$$

$$\Rightarrow y - 1 = \frac{1}{1+x}$$

$$\Rightarrow 1+x = \frac{1}{y-1}$$

$$\Rightarrow x = \frac{1}{y-1} - 1 = \frac{1-(y-1)}{y-1} = \frac{2-y}{y-1}$$

So:

$$a_n = \frac{2-a_{n+1}}{a_{n+1}-1}$$

Step 2: Compute a_2 from $a_3 = \frac{41}{29}$.

$$a_2 = \frac{2-\frac{41}{29}}{\frac{41}{29}-1} = \frac{\frac{58-41}{29}}{\frac{12}{29}} = \frac{17}{12}$$

830

831

Step 3: Compute a_1 from $a_2 = \frac{17}{12}$:

$$a_1 = \frac{2 - \frac{17}{12}}{\frac{17}{12} - 1} = \frac{\frac{24-17}{12}}{\frac{17-12}{12}} = \frac{\frac{7}{12}}{\frac{5}{12}} = \frac{7}{5}$$

Verification

$$- a_1 = \frac{7}{5}$$

$$- a_2 = 1 + \frac{1}{1 + \frac{7}{5}} = 1 + \frac{1}{\frac{12}{5}} = 1 + \frac{5}{12} = \frac{17}{12}$$

$$- a_3 = 1 + \frac{1}{1 + \frac{17}{12}} = 1 + \frac{1}{\frac{29}{12}} = 1 + \frac{12}{29} = \frac{41}{29}$$

All match.

Thus, the value of a_1 is $\boxed{\frac{7}{5}}$.

Analysis: Rhombus structures the response into Logic, Calculation, and Verification. The explicit separation allows the Synthesizer to validate the logic ($a_2 \rightarrow a_3$) without re-deriving the entire chain.