PaCoRe: Learning to Scale Test-Time Compute with Parallel Coordinated Reasoning

Anonymous ACL submission

Abstract

We introduce Parallel Coordinated Reasoning (PaCoRe), a training-and-inference framework designed to overcome a central limitation of contemporary language models: their inability to scale test-time compute (TTC) far beyond sequential reasoning under a fixed context window. PaCoRe departs from the traditional sequential paradigm by driving TTC through massive parallel exploration coordinated via a message-passing architecture in multiple rounds. Each round launches many parallel reasoning trajectories, compacts their findings into context-bounded messages, and synthesizes these messages to guide the next round and ultimately produce the final answer. Trained end-to-end with large-scale, outcome-based reinforcement learning, the model masters the synthesis abilities required by PaCoRe and scales to multi-million-token effective TTC without exceeding context limits. The approach yields strong improvements across diverse domains and notably pushes reasoning beyond frontier systems in mathematics: an 8B model reaches 94.5% on HMMT 2025, surpassing GPT-5's 93.2% by scaling effective TTC to roughly two million tokens. We open-source model checkpoints, training data, and the full inference pipeline to accelerate follow-up work.

1 Introduction

Long-horizon reasoning, requiring sustained exploration, cross-checking, and iterative self-correction, underpins the most demanding intellectual pursuits. Approaching such capabilities in AI demands substantial test-time compute (TTC)—the computation devoted to solving a single challenging instance.

A recurring lesson in the history of deep learning is that progress on hard problems often comes from scaling test-time compute through search (Silver et al., 2016; Li et al., 2022; Trinh et al., 2024) —expanding both in depth (sequentially) and width (in parallel)—rather than relying on a single forward pass. Yet for contemporary language models (Brown et al., 2020), a fixed context window places a hard ceiling on how much such search-driven TTC can be expressed: standard sequential reasoning (Wei et al., 2023) packs every intermediate state into a single expanding chain, strictly coupling reasoning volume to context capacity; once the window fills, the reasoning must stop.

To address this, we introduce **Parallel Coordinated Reasoning (PaCoRe)**, a framework that decouples test-time compute scaling from context limits by shifting the primary driver of inference from solely sequential depth to elevated coordinated breadth. PaCoRe iteratively coordinates parallel reasoning trajectories and compresses their insights into compact messages. The messages from the previous round are then synthesized within the model's context window to guide subsequent exploration, and ultimately, produce the final answer.

Crucially, PaCoRe's success hinges on shifting the model from naive aggregation and solitary problem-solving to active coordination. Vanilla reasoning models tend to rely on **simple heuristics** such as majority voting (Zhao et al., 2025) when problems admit a simple, easily comparable answer. However, on problems with more complex solution structures (*e.g.*, code generation and theorem proving), they often exhibit **Reasoning Solipsism**: despite receiving rich insights from parallel branches, they ignore this context and attempt to solve the problem from scratch, wasting the accumulated test-time compute (shown in Figure 1, right). To overcome this, we employ large-scale, outcome-based reinforcement learning, compelling the model to master **Reasoning Synthesis**: the capacity to scrutinize parallel branches, reconcile conflicting evidence, and synthesize a unified solution that exceeds any individual trajectory.

We evaluate PaCoRe across challenging benchmarks in diverse domains and observe substantial

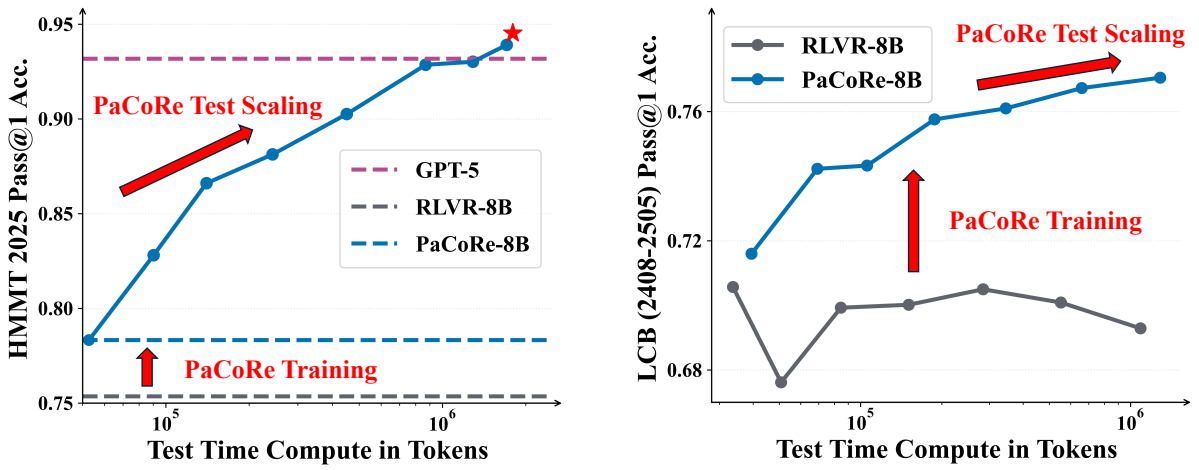

Figure 1: **Pa**rallel **Co**ordinated **Re**asoning (PaCoRe) performance. ***Left:*** On HMMT 2025, PaCoRe-8B demonstrates remarkable test-time scaling by increasing both parallel trajectories and coordinated rounds, yielding steady gains and ultimately surpassing GPT-5. ***Right:*** On LiveCodeBench, the RLVR-8B model fails to leverage increased test-time compute, while PaCoRe-8B model effectively unlocks this synthesis capability, yielding substantial gains as the test-time compute increases.

gains from massively scaling TTC far beyond the limits of standard decoding. The results are especially pronounced in mathematics: on the HMMT 2025 benchmark, our PaCoRe-8B model achieves a score of 94.5%, surpassing the leading proprietary reasoning model, GPT-5, at 93.2%. This performance is achieved by scaling effective TTC up to nearly two million tokens per problem, orchestrated entirely within the model's context window. Our primary contributions are as follows:

1. We introduce **PaCoRe**, a general framework that decouples reasoning volume from model context by coordinating parallel reasoning, enabling multi-million token effective TTC.

2. We demonstrate that large-scale, outcome-based RL is essential for inducing the synthesis capabilities required by this framework in diverse and challenging reasoning settings.

3. We release comprehensive resources including model checkpoints, training data, and the inference code, to enable rigorous evaluation and foster further research.

2 Methods

The framework comprises training and inference pipelines. The inference pipeline coordinates parallel exploration to massively scale test-time compute, growing far beyond the limits of sequential reasoning while remaining independent of context constraints. The training procedure employs large-scale, outcome-based reinforcement learning to teach the model the synthesis skills required to consolidate diverse trajectories and generate high-quality final answers.

2.1 Inference Pipeline

Given a problem instance x, the PaCoRe inference pipeline executes R rounds of **coordinated reasoning**. At each round $r \in \{1, \ldots, R\}$, the system inherits a set of **compact messages** $M_{r-1} = \{m_{r-1}^{(i)}\}_{i=1}^{K_{r-1}}$ from the previous round, and generates a new set of **reasoning trajectories** $\Omega_r = \{\omega_r^{(i)}\}_{i=1}^{K_r}$ that explore the solution space in parallel. The final answer is simply the degenerate case where $K_r = 1$ at the last round. For convenience, we refer to the preceding coordinated-reasoning process as consisting of $\hat{R} = R - 1$ rounds, and refer $\vec{K} = [K_1, \ldots, K_{\hat{R}}]$ as inference configuration.

For each problem x , the model begins each round by taking the combined context (x, M_{r-1}) and generating a set of parallel trajectories $\Omega_r = \{\omega_r^{(i)}\}$. The message set M_{r-1} provides a compacted summary of reasoning trajectories from the previous round, and each trajectory $\omega_r^{(i)}$ contains a full chain of reasoning followed by a final conclusion. Each round then comprises two stages: (i) **Synthesis and Parallel Exploration**, where the model takes the combined context (x, M_{r-1}) and produces the trajectories Ω_r; and (ii) **Message Compaction**, where the trajectories Ω_r are compressed into the next-round message set M_r, allow-

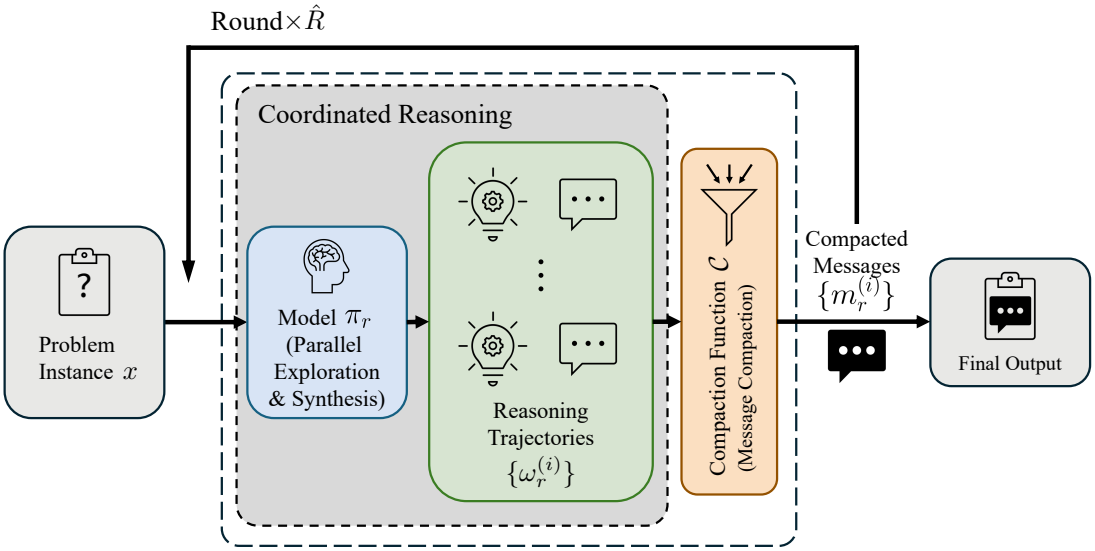

Figure 2: **Inference pipeline of PaCoRe.** Each round launches broad parallel exploration, compacts the resulting trajectories into compacted messages, and feeds these messages together with the question forward to coordinate the next round. Repeating this process $\hat{R}$ times yields multi-million-token effective TTC while respecting fixed context limits, with the final compacted message serving as the system's answer.

ing PaCoRe to massively scale effective test-time compute under a fixed context window. We next detail these two stages of **coordinated reasoning**, outlining how PaCoRe constructs inputs, generates trajectories, and compacts them across rounds.

Synthesis and Parallel Exploration. At the start of round r, the system uses the prompting function $P(x, M_{r-1})$ to serialize the problem and the compact messages into a structured input sequence. Once the input is constructed, the core reasoning model π_r is then invoked in parallel to produce K_r independent trajectories:

$$\omega_r^{(i)} \sim \pi_r(\cdot \mid P(x, M_{r-1})). \tag{1}$$

In our implementation, we employ the same model weights across all rounds for operational convenience. This parallel generation constitutes the main drive of TTC: even though the system can accumulate millions of tokens through parallel expansion, each round's input consumes only (x, M_{r-1}), maintaining an almost constant context cost.

Message Compaction. To enable multi-round coordination while respecting the limited context, the full trajectories in Ω_r must be compressed into short messages M_r. We apply a compaction function to achieve this:

$$M_r = \mathcal{C}(\Omega_r). \tag{2}$$

In our implementation, we leverage the structured nature of the generated trajectories. Reasoning models typically produce detailed intermediate derivations (*e.g.*, chain-of-thought or "reasoning content") followed by a final conclusion. We therefore implement $\mathcal{C}(\cdot)$ as a trajectory-wise extraction function $C(\cdot)$: it parses each $\omega_r^{(i)} \in \Omega_r$, discards the intermediate steps and retains only the final conclusion segment, forming $m_r^{(i)} = C(\omega_r^{(i)})$. This ensures that the input length for synthesis remains bounded (*i.e.*, $|x| + \sum_i |m_r^{(i)}|$ fits within the context), even as the TTC—aggregate token count across all trajectories, $\sum_r \sum_i |\omega_r^{(i)}|$—scales vastly.

Iterative Coordination. The process repeats for R rounds. To ensure convergence, the final round uses a single trajectory ($K_R = 1$), producing a final compact message $y = m_R^{(1)}$, which constitutes the output of the PaCoRe inference pipeline.

2.2 Training Procedure

The efficacy of the PaCoRe framework fundamentally depends on the core model's capacity for ***synthesis***: the ability to critically evaluate diverse perspectives within the input messages M, reconcile conflicting information, and generate novel strategies that surpass the quality of any individual input.

To train this capability, we instantiate the synthesis stage of a single PaCoRe round as an episodic RL environment, with the core model serving as the policy to be optimized. In each training episode, we sample a problem x and a corresponding set of input messages M from a training distribution D,

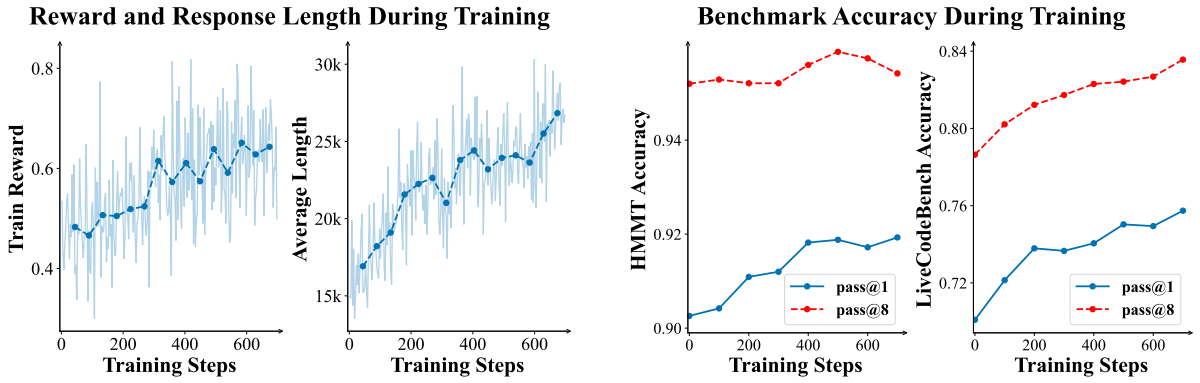

Figure 3: **PaCoRe Training dynamics.** ***Left panels:*** The Training Reward and Response Length steadily increase, demonstrating the training stability and effectiveness. ***Right panels:*** Evaluation on HMMT 2025 and LiveCodeBench (2408-2505). Performance is reported using single round coordinated reasoning in PaCoRe inference setting with $\vec{K} = [16]$.

where M can be either generated online or drawn from a cached pool. Given the sampled pair (x, M), the policy π_θ generates a reasoning trajectory ω from the formatted input $P(x, M)$, and receives a sparse terminal reward $R(\omega) \in [0, 1]$ at the end.

We then apply large-scale, outcome-based reinforcement learning to elicit these advanced synthesis behaviors. Training is performed on challenging domains, where each trajectory is evaluated by the correctness of its extracted message. In order to ensure that naive strategies like majority voting or random selection are insufficient and compel the policy to develop synthesis to succeed, we discard training instances where the average accuracy of the message set M exceeds a predefined threshold.

Compared to standard RL on static tasks with verifiable rewards (Lambert et al., 2025), PaCoRe represents a significant departure. Because the input context M is composed of model-generated messages, the policy must function within an implicitly multi-agent environment rather than a fixed single-agent setting. Although the outcome-based reward aligns with conventional RL formulations for chain-of-thought optimization, PaCoRe demands higher-level synthesis and coordination across agents' outputs, introducing opportunities to examine emergent collective behaviors.

3 Experiments

In this section, we present comprehensive experimental results and analysis of PaCoRe framework. We begin by detailing the training protocol to develop our PaCoRe-8B model, followed by in-depth analysis of training results. We then present the main evaluation results, comparing PaCoRe against frontier reasoning models on challenging benchmarks. Finally, we conduct analysis and ablations to validate key design choices and investigate the properties of the learned synthesis capabilities.

3.1 Training

Training Recipe. We initialize PaCoRe-8B from an internal, reasoning-oriented post-trained version of Qwen3-8B-Base (Yang et al., 2025) (denoted as RLVR-8B). We use RLVR-8B to generate a message cache pool of size 24 for each problem in the training dataset. During training, we randomly sample the message set M from this pool with $|M| \sim U(16, 24)$. We employ strict on-policy PPO (Schulman et al., 2017) with GAE ($\lambda = 1, \gamma = 1$) following the ORZ setup (Hu et al., 2025), utilizing a batch size of 16 instances (64 responses each), a maximum sequence length of 131,072, and generation temperature/top-p of 1.0. We leverage competition-level mathematics and coding tasks as the primary substrate to cultivate model capabilities. To incentivize the model to learn synthesis, the training data distribution is refined over two stages. In **Stage 1** (250 iterations), we target scenarios where naive aggregation fails by filtering for low message set accuracy ($0 < \text{mean}(\text{message_acc}) < 9/24$ for math; $< 15/24$ for code), alongside quality filtering. In **Stage 2** (450 additional iterations), we further refine the distribution. We use an intermediate Stage 1 checkpoint to evaluate synthesis accuracy on the Stage 1 data, retaining only instances where $0 < \text{synthesis_acc} < 1$. The final PaCoRe-8B model is obtained after 700 total iterations.

Training Results. We present the training dynamics by tracking training rewards, response lengths, and performance on key benchmarks. As

Benchmark	AIME 2025	HMMT 2025	IMO AnswerBench	Apex	LiveCodeBench	HLE_{text}	Multi Challenge
GPT-5	93.5 (13k)	93.2 (16k)	72.9 (26k)	1.0 (33k)	**83.5** (13k)	**26.0** (14k)	**71.1** (5.0k)
Qwen3-235B-Thinking	91.6 (26k)	82.3 (32k)	71.7 (34k)	**3.3** (46k)	74.5 (21k)	18.2 (23k)	60.3 (1.6k)
GLM-4.6	92.3 (20k)	88.7 (25k)	73.5 (37k)	0.7 (53k)	79.5 (19k)	17.2 (21k)	54.9 (2.2k)
DeepSeek-v3.1*	90.2 (16k)	86.1 (20k)	63.0 (27k)	1.4 (36k)	74.9 (11k)	19.3 (18k)	54.4 (1.1k)
Kimi-K2-Thinking	**95.3** (25k)	86.5 (33k)	76.5 (44k)	0.8 (60k)	79.2 (25k)	23.9 (29k)	66.4 (1.6k)
RLVR-8B	84.1 (50k)	75.4 (48k)	64.6 (56k)	0.0 (65k)	70.6 (34k)	9.3 (35k)	33.3 (1.7k)
PaCoRe-8B (low)	89.7 (255k)	88.1 (243k)	76.1 (306k)	0.7 (362k)	75.8 (188k)	13.0 (196k)	41.8 (13k)
PaCoRe-8B (medium)	92.5 (908k)	92.9 (869k)	77.3 (1080k)	1.4 (1280k)	76.7 (659k)	14.6 (694k)	45.7 (45k)
PaCoRe-8B (high)	93.7 (1873k)	**94.5** (1796k)	**78.4** (2258k)	2.3 (2679k)	78.2 (1391k)	16.0 (1451k)	48.0 (95.3k)

Table 1: **Benchmark performance and TTC spent per problem instance.** For each benchmark, we report accuracy together with total TTC (in thousands). *DeepSeek-V3.1 refers to the Terminus version. For *Low*, *Medium*, and *High*, we apply the inference configuration as $\vec{K} = [4]$, $[16]$, and $[32, 4]$ separately.

illustrated in Figure 3, both the training reward and response length steadily increase throughout the process. Correspondingly, the performance on both HMMT 2025 and LiveCodeBench (2408-2505) improves significantly. In these evaluations, the PaCoRe inference setting is $\vec{K} = [16,]$. In order to improve experimental efficiency, we utilize cached responses from the RLVR-8B model for each problem instance to directly seed the current checkpoint for the next round generation. All these results indicate stable and effective learning.

3.2 Evaluation

Evaluation Setup. We evaluate PaCoRe on representative benchmarks across mathematics, coding, science, and open-ended generation: AIME 2025, HMMT 2025 (Feb.), Apex, IMOAnswerBench, LiveCodeBench (2408-2505), Humanity's Last Exam (text subset), and MultiChallenge (Balunović et al., 2025; Luong et al., 2025; Jain et al., 2024; Team, 2025a; Sirdeshmukh et al., 2025). We compare against frontier reasoning models including GPT-5 ("high reasoning effort) (OpenAI, 2025), Kimi-K2-Thinking (AI, 2025), DeepSeek-V3.1-Terminus (DeepSeek, 2025), GLM-4.6 (Z.ai, 2025), and Qwen3-235B-A22B-Thinking-2507 (Team, 2025b). We report pass@1 accuracy for all approaches based on average performance of multiple independent generations per problem: 64 for AIME 2025, HMMT 2025, and APEX; 8 for IMOAnswerBench, LiveCodeBench, and MultiChallenge; and 1 for HLE. For both PaCoRe and RLVR evaluations, We maintain a maximum generation sequence length of 131,072 tokens with a temperature and top-p of 1.0, and we apply YaRN (Peng et al., 2023) (scale 2.0) without modifying attention temperature. We evaluate our PaCoRe inference pipeline under three coordinated reasoning levels: *Low* ($\vec{K} = [4]$), *Medium* ($\vec{K} = [16]$), and *High* ($\vec{K} = [32, 4]$). To optimize experimental efficiency without compromising validity, we utilize a caching strategy for the first round for PaCoRe evaluation: we pre-generate a pool of 512 trajectories for each problem, from which the model randomly samples K_1 items to seed the next round. We empirically validated that this method yields results equivalent to generating all trajectories from scratch.

Evaluation Results. The main results are summarized in Table 1. Across all test-time effort settings, PaCoRe-8B consistently outperforms the starting checkpoint, RLVR-8B, on every benchmark. Even in the *Low* setting, PaCoRe-8B demonstrates robust capabilities, achieving 88.1% on HMMT2025 and 75.8% on LiveCodeBench, significantly outperforming the RLVR-8B baseline and surpassing considerably larger models like Qwen3-235B-A22B-Thinking-2507 and DeepSeek-V3.1-Terminus. With further test-time scaling in the *High* setting, PaCoRe-8B reaches a remarkable 94.5% on HMMT 2025 and 78.4% on IMOAnswerBench, surpassing the leading proprietary model, GPT-5, by scaling effective TTC to approximately 2 million tokens. A standout result is observed on the extremely challenging Apex benchmark: while the RLVR-8B fails to answer any questions correctly (0.0%), PaCoRe-8B achieves 2.3% in the *High* setting. On LiveCodeBench, PaCoRe-8B achieves a strong 78.2%, remaining competitive with frontier models like GLM-4.6 and Kimi-K2-Thinking. These findings demonstrate that parallel coordinated reasoning allows an 8B model to bridge the gap with, and often surpass, state-of-the-art systems on complex reasoning tasks.

3.3 Analysis and Ablations

Ablation of Core Design Principles. We study the importance of the two central components of

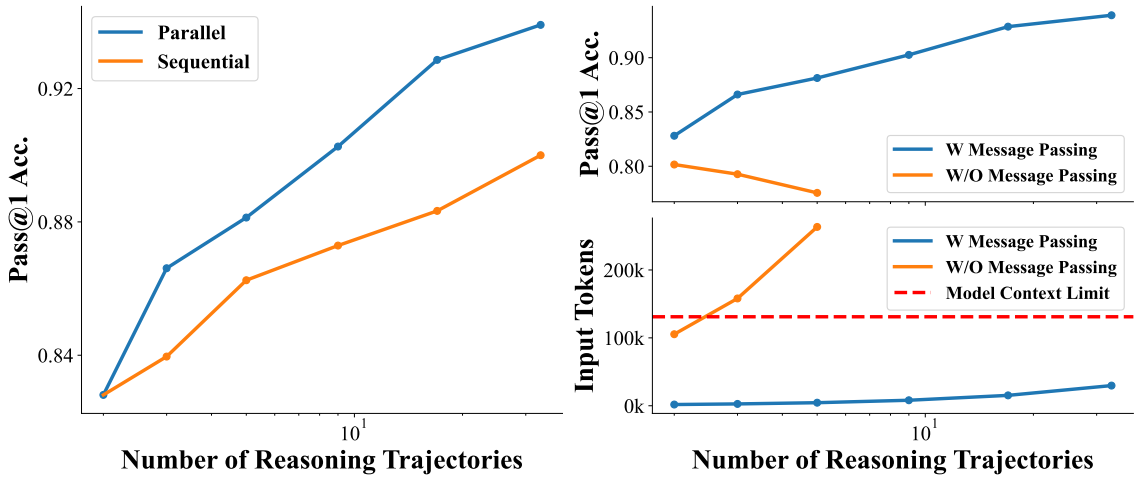

Figure 4: **Ablation of parallel reasoning and message passing.** ***Left:*** **Parallel** scaling ($\vec{K} = [N,]$) utilizes test-time compute more effectively than **Sequential** scaling ($\vec{K} = [1, \ldots, 1]$). ***Right:*** **Message Passing** is essential for test-time scaling. Without compaction ("W/O Message Passing"), performance degrades as test time scales and fundamentally limited by model context length, whereas standard PaCoRe ("W Message Passing") scales unboundedly and robustly. Pass@1 accuracy is evaluated on HMMT 2025.

PaCoRe: parallel exploration and message passing. First, we compare parallel scaling (PaCoRe with $\vec{K} = [N,]$) against sequential scaling (equivalent to $\vec{K} = [1, \ldots, 1]$) under the same total number of generated trajectories. As shown in Figure 4 (Left), parallel coordinated reasoning utilizes TTC much more effectively than purely sequential approach. Second, we examine the role of message passing. We compare PaCoRe (W Message Passing) with a variant where compaction is disabled (W/O Message Passing), feeding the full trajectories into the next round context. Figure 4 (Right) shows that without compaction, performance degrades as TTC scales and is fundamentally limited by the context length. In contrast, PaCoRe, with message passing, scales robustly and without bound.

Evolution of Synthesis Capabilities. We probe the mechanism underlying PaCoRe by tracking the evolution of synthesis behaviors during training. First, the frequency of "cross-checking" linguistic markers rises steadily across both domains (Figure 5, Left). Notably, the near-zero initial frequency in the Code domain aligns with the poor test-time scaling of untrained models (Figure 1), confirming that training fundamentally transforms the model into a coordinated reasoner. To distinguish genuine synthesis from simple aggregation or refinement, we track the *Emergent Correctness Rate* (Figure 5, Right), defined as the probability of generating a correct solution given input messages that are all incorrect. The upward trend in both domains demonstrates that the model transcends naive strategies like majority voting or random selection (which fail given all wrong inputs), learning instead to reconstruct valid solutions from erroneous partial evidence. This capability is empirically confirmed in Table 5, where PaCoRe demonstrates robust test-time scaling while the voting-based Self-Consistency baseline saturates rapidly.

Model	SWE-Verified
RLVR-8B	29.8%
PaCoRe-8B (low)	**34.0%**

Table 2: **Performance on SWE-Verified.** PaCoRe-8B (low) markedly surpasses the RLVR-8B baseline, indicating generalization to software engineering scenarios without task-specific tuning.

Generalization Across Domains. To assess cross-domain generalization beyond mathematical and logical reasoning benchmarks, we evaluate PaCoRe on software engineering and multi-turn conversation tasks using the SWE-Verified (OpenAI, 2024a) and MultiChallenge (Sirdeshmukh et al., 2025), respectively. We note that the starting checkpoint (RLVR-8B) underwent no specialized post-training for these tasks; thus, this study assesses the intrinsic transfer of reasoning capabilities rather than fully optimized performance. For SWE-Verified, we compare a compute-efficient variant, PaCoRe-8B (low), against the RLVR-8B baseline, both under Agentless (Xia et al., 2024) framework. As shown in Table 2, PaCoRe-8B (low) achieves a resolve rate of 34.0%, substantially out-

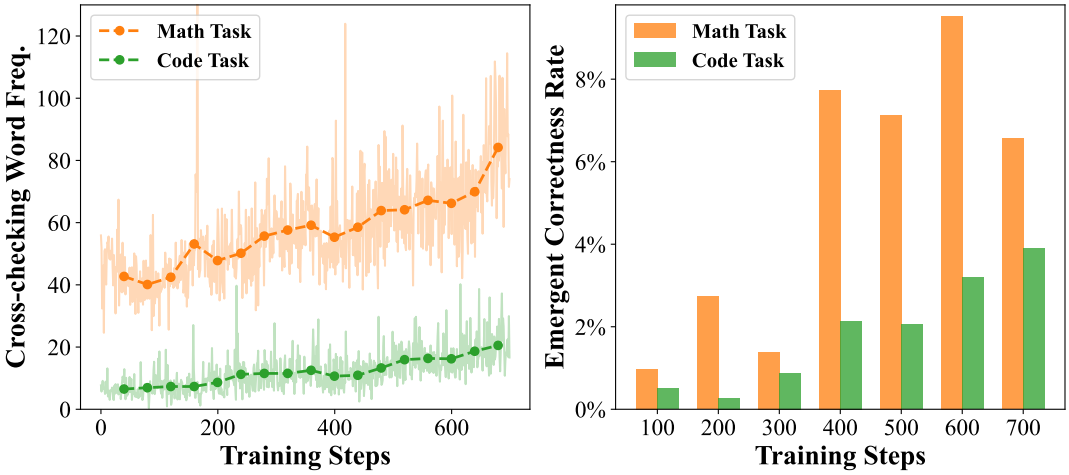

Figure 5: **Evolution of synthesis-related linguistic features and emergent correctness across training steps.** ***Left:*** Frequency of cross-checking words (including 'reference', '参考', 'Ref <number>', 'ref <number>') in generated solutions. Training elicits and magnifies this capability across domains; notably, the near-zero initial frequency in Code corroborates the poor test-time scaling of untrained models (Figure 1). ***Right:*** The **Emergent Correctness Rate** tracks the probability of generating a correct solution given input messages that are all incorrect, averaged over 100-step intervals. The upward trend in both domains demonstrates that through large scale RL training, the model transcends naive strategies like majority voting or random selection to achieve genuine synthesis, recovering valid solutions even from entirely erroneous contexts.

performing the baseline (29.8%). Furthermore, PaCoRe again exhibits strong generalization on MultiChallenge, with PaCoRe-8B (high) improving the RLVR-8B baseline from 33.3% to 48.0% under higher TTC budgets as shown in Table 1

Model	AIME 2025	LiveCodeBench
Ours-SFT-Qwen3-30B-A3B	81.4%	66.0%
+ RLVR with PaCoRe Data	**83.2%**	**74.0%**

Table 3: **General Effectiveness of PaCoRe Data.** RLVR trained on PaCoRe data, with only 200 update iterations, achieves significant gains on benchmarks

General Effectiveness of PaCoRe Data. Beyond the framework itself, we discovered that the training corpus curated for PaCoRe constitutes a general applicable learning resource of exceptional density. As shown in Table 3, we observe that employing our released dataset as the primary substrate for standard RLVR yields a robust performance boost. The RLVR stage uses high-quality, carefully curated training data and applies an off-policy PPO algorithm with GAE ($\gamma = 1, \lambda = 1$) for only 50 iterations and 4 mini-batches per iteration on ours-SFT-Qwen3-30B-A3B. Despite this minimal compute budget, RLVR delivers substantial gains over the SFT model on both AIME 2025 and LiveCodeBench. This indicates that our problem set—carefully filtered to demand genuine synthesis—serves as a highly effective catalyst for training strong reasoning models in general.

Setting	HMMT 2025	LiveCodeBench
$\lvert M\rvert = 4$	83.6	63.3
$\lvert M\rvert = 8$	83.7	64.4
$\lvert M\rvert = 16$	84.2	64.1
$\lvert M\rvert \sim U(1, 16)$	84.3	64.3
$\lvert M\rvert \sim U(8, 16)$	**85.2**	**65.1**

Table 4: Ablation of how the message-set size $|M|$ is determined during training.

Ablation on Message Sampling Strategy in Training. We investigate the impact of message set size $|M|$ during training by comparing fixed versus uniformly sampled sizes. As shown in Table 4, training with diverse and relatively large message sets ($|M| \sim U(8, 16)$) yields the best performance when evaluated under a fixed inference configuration ($\vec{K} = [16,]$). This randomized sampling strategy improves robustness by familiarizing the model with varying parallel coordination settings, thereby facilitating superior test-time scaling.

4 Related Work

Classical approaches to sequentially scaling TTC, such as CoT (Wei et al., 2023), pack intermediate steps into a single expanding chain. Even when boosted by large-scale RL (OpenAI, 2024b; DeepSeek-AI, 2025), these methods inevitably saturate the context, strictly ceiled on achievable TTC.

A shift towards parallel scaling has shown immense potential. Early methods (Wang et al., 2023;

Method	Setting	Benchmark			
		AIME 2025	HMMT 2025	IMO AnswerBench	Apex
Self Consistency	@4	87.0 (194k)	82.3 (193k)	69.7 (226k)	0.3 (249k)
	@16	88.5 (776k)	85.9 (773k)	72.1 (904k)	0.0 (995k)
	@64	89.5 (3106k)	85.1 (3094k)	72.8 (3615k)	0.0 (3981k)
	@256	90.0 (12422k)	84.7 (12376k)	73.0 (14458k)	0.0 (15924k)
PaCoRe	low	89.7 (255k)	88.1 (243k)	76.1 (306k)	0.7 (362k)
	medium	92.5 (908k)	92.9 (869k)	77.3 (1080k)	1.4 (1280k)
	high	93.7 (1873k)	94.5 (1796k)	78.4 (2258k)	2.3 (2679k)

Table 5: **Test-time scaling comparison with Self-Consistency sampling (Majority voting).** We compare PaCoRe against standard Self-Consistency (SC) on representative short-answer benchmarks. Token counts (TTC) are reported in parentheses in thousands (k). Self-Consistency sampling are conducted using RLVR-8B and repeated sampling from a 512 size of pool of responses for each problem for 64 times to estimate the accuracy.

Trinh et al., 2024; El-Kishky et al., 2025) use simple rules to integrate language models with extensive parallel search, achieving strong performance in specific domains. However, the coordination in these systems often hinges on task-specific scaffolds or priors, hindering their general applicability.

This highlights the need for a general framework to coordinate massive parallel exploration within model context limits. Attempts at such coordination (Chen et al., 2023; Wang et al., 2024; Wen et al., 2025; Zheng et al., 2025; Venkatraman et al., 2025; Wu et al., 2025) synthesize parallel outputs but lack mechanisms for message passing or context management, thus facing the same context limitations. Alternative approaches focused on context management (Yan et al., 2025; Aghajohari et al., 2025) use compression rules but remain rooted in the sequential paradigm, limiting TTC scaling efficiency. AggLM (Zhao et al., 2025) focuses on learning aggregation strategies that surpass majority voting and reward-model ranking, while we introduce parallel coordinated reasoning with context compaction to scale test-time compute beyond context limits. Thus, scaling trajectory configurations and rounds of coordinated reasoning yields substantial gains, outperforming AggLM and even GPT-5 in the mathematics domain.

Concurrent work PDR primarily focuses on refinement behavior to optimize the latency–accuracy trade-off, whereas PaCoRe is motivated by scaling test-time compute far beyond context limitations for general-purpose reasoning. Despite architectural similarities, PaCoRe adopts a specialized data curation that explicitly targets instances requiring complex synthesis, rather than iterative correction or refinement. This design encourages reasoning behaviors beyond naive aggregation. As a result, PaCoRe exhibits a broader range of emergent behaviors, including systematic cross-checking and the synthesis of correct solutions even when all individual input messages are incorrect.

Recently, frontier proprietary systems (xAI, 2025; DeepMind, 2025; OpenAI, 2025) have demonstrated exceptional reasoning performance, reportedly via utilizing massive parallel TTC. While these systems showcase the power of this paradigm, their undisclosed technical details hinder broader scientific progress. PaCoRe bridges this gap by providing an effective, and open-sourced framework for parallel coordinated reasoning, accelerating future research in this direction.

5 Conclusion

In this work, we introduce Parallel Coordinated Reasoning (PaCoRe), a test-time scaling framework that enables massive test-time compute (TTC) without exceeding the context window constraints of contemporary language models. PaCoRe addresses the limitations of sequential reasoning by shifting the primary driver of inference to coordinated parallel breadth. It operates via a message-passing architecture that iteratively launches many parallel reasoning trajectories, compacts their findings into context-bounded messages, and synthesizes these messages to guide subsequent exploration. Trained end-to-end with large-scale, outcome-based reinforcement learning, the model develops the synthesis capabilities required for effective coordination. Our empirical results demonstrate significant improvements across diverse domains. Notably, our PaCoRe-8B model achieves 94.5% on HMMT 2025, surpassing GPT-5 by scaling effective TTC to roughly two million tokens, all orchestrated within a standard context window.

Limitations

While PaCoRe successfully decouples reasoning performance from the constraints of sequential decoding, our current study operates within specific boundaries that invite further exploration. First, we rely on scaling the *volume* of test-time compute via massive parallelism rather than strictly optimizing *intelligence density*—the utility derived per unit of compute. Our current implementation employs homogeneous search without explicit division of labor. Future work could design strategies that significantly reduce redundancy by orchestrating specialized endeavors and cooperative planning among trajectories. Second, we validate our framework exclusively on pure-text domains to focus on intrinsic reasoning. We have not yet integrated external tool usage or multi-modal perception. Extending PaCoRe to agentic settings where synthesis must coordinate active tool interactions and multimodal inputs remains a critical step to establish the framework's broader generality. Finally, while PaCoRe constitutes a distinct training stage within our overall model pipeline, we have not yet investigated how to co-optimize it with the foundational pre-training and post-training processes. Leveraging the framework to drive an advanced synthetic data generation pipeline offers a promising path toward such integration.

References

Aime problems and solutions. https://artofproblemsolving.com/wiki/index.php/AIME_Problems_and_Solutions. Accessed: 2025-12-07.

Berkeley math tournament (bmt). https://berkeley.mt/. Accessed: 2025-12-07.

Brown university math olympiad (brumo). https://www.brumo.org/. Accessed: 2025-12-07.

Caltech harvey mudd math competition (chmmc). http://chmmc.caltech.edu/problems.html. Accessed: 2025-12-07.

Carnegie mellon informatics and mathematics competition (cmimc). https://cmimc.math.cmu.edu/. Accessed: 2025-12-07.

Duke math meet (dmm). https://dukemathmeet.org/. Accessed: 2025-12-07.

Harvard–mit mathematics tournament (hmmt). https://www.hmmt.org/. Accessed: 2025-12-07.

Math prize for girls. https://mathprize.atfoundation.org/. Accessed: 2025-12-07.

Minnesota youth math outreach (mnymo). https://www.mnyouthmathoutreach.org/. Accessed: 2025-12-07.

Princeton university mathematics competition (pumac). https://jason-shi-f9dm.squarespace.com/. Accessed: 2025-12-07.

Stanford math tournament (smt). https://www.stanfordmathtournament.org/. Accessed: 2025-12-07.

Milad Aghajohari, Kamran Chitsaz, Amirhossein Kazemnejad, Sarath Chandar, Alessandro Sordoni, Aaron Courville, and Siva Reddy. 2025. The markovian thinker: Architecture-agnostic linear scaling of reasoning. *Preprint*, arXiv:2510.06557.

Moonshot AI. 2025. Introducing kimi k2 thinking.

Alon Albalak et al. 2025. Big-math: A large-scale, high-quality math dataset for reinforcement learning in language models. *arXiv preprint arXiv:2502.17387*.

aslawliet. 2024a. Cn-k12. https://huggingface.co/datasets/aslawliet/cn-k12. Hugging Face dataset of Chinese K-12 math problems.

aslawliet. 2024b. Olympiads. https://huggingface.co/datasets/aslawliet/olympiads. Hugging Face dataset.

Mislav Balunović, Jasper Dekoninck, Ivo Petrov, Nikola Jovanović, and Martin Vechev. 2025. MathArena: Evaluating LLMs on uncontaminated math competitions.

Mislav Balunović, Jasper Dekoninck, Ivo Petrov, Nikola Jovanović, and Martin Vechev. 2025. Matharena: Evaluating llms on uncontaminated math competitions. *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmark*.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. *Preprint*, arXiv:2005.14165.

Xinyun Chen, Renat Aksitov, Uri Alon, Jie Ren, Kefan Xiao, Pengcheng Yin, Sushant Prakash, Charles Sutton, Xuezhi Wang, and Denny Zhou. 2023. Universal self-consistency for large language model generation. *Preprint*, arXiv:2311.17311.

Google DeepMind. 2025. Gemini 2.5 deep think model card.

DeepSeek. 2025. Deepseek-v3.1-terminus.

DeepSeek-AI. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *Preprint*, arXiv:2501.12948.

Ahmed El-Kishky, Alexander Wei, Andre Saraiva, Borys Minaiev, Daniel Selsam, David Dohan, Francis Song, Hunter Lightman, Ignasi Clavera, Jakub Pachocki, Jerry Tworek, Lorenz Kuhn, Lukasz Kaiser, Mark Chen, Max Schwarzer, Mostafa Rohaninejad, Nat McAleese, o3 contributors, Oleg Mürk, Rhythm Garg, Rui Shu, Szymon Sidor, Vineet Kosaraju, and Wenda Zhou. 2025. Competitive programming with large reasoning models. *Preprint*, arXiv:2502.06807.

Etash Guha et al. 2025. Openthoughts: Data recipes for reasoning models. *arXiv preprint arXiv:2506.04178*.

X. He et al. 2025. Deepmath-103k: A large-scale, challenging math qa benchmark. *arXiv preprint arXiv:2504.11456*.

Jingcheng Hu, Yinmin Zhang, Qi Han, Daxin Jiang, Xiangyu Zhang, and Heung-Yeung Shum. 2025. OpenReasoner-Zero: An open source approach to scaling up reinforcement learning on the base model. *arXiv preprint arXiv:2503.24290*.

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

Yunjie Ji, Xiaoyu Tian, Sitong Zhao, Haotian Wang, Shuaiting Chen, Yiping Peng, Han Zhao, and Xiangang Li. 2025. Am-thinking-v1: Advancing the frontier of reasoning at 32b scale. *Preprint*, arXiv:2505.08311.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V. Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, Yuling Gu, Saumya Malik, Victoria Graf, Jena D. Hwang, Jiangjiang Yang, Ronan Le Bras, Oyvind Tafjord, Chris Wilhelm, Luca Soldaini, Noah A. Smith, Yizhong Wang, Pradeep Dasigi, and Hannaneh Hajishirzi. 2025. Tulu 3: Pushing frontiers in open language model post-training. *Preprint*, arXiv:2411.15124.

Jia Li et al. 2024. Numinamath. `https://huggingface.co/datasets/AI-MO/NuminaMath-CoT`. Technical report and dataset card.

Rongao Li, Jie Fu, Bo-Wen Zhang, Tao Huang, Zhihong Sun, Chen Lyu, Guang Liu, Zhi Jin, and Ge Li. 2023. Taco: Topics in algorithmic code generation dataset. *Preprint*, arXiv:2312.14852.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. 2022. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097.

LIMO Authors. 2025. Less is more for reasoning: Semi-parametric math reasoners. *arXiv preprint arXiv:2502.03387*.

Michael Luo, Sijun Tan, Roy Huang, Ameen Patel, Alpay Ariyak, Qingyang Wu, Xiaoxiang Shi, Rachel Xin, Colin Cai, Maurice Weber, Ce Zhang, Li Erran Li, Raluca Ada Popa, and Ion Stoica. 2025. Deepcoder: A fully open-source 14b coder at o3-mini level. `https://www.together.ai/blog/deepcoder`. Technical Blog.

Thang Luong, Dawsen Hwang, Hoang H. Nguyen, Golnaz Ghiasi, Yuri Chervonyi, Insuk Seo, Junsu Kim, Garrett Bingham, Jonathan Lee, Swaroop Mishra, Alex Zhai, Clara Huiyi Hu, Henryk Michalewski, Jimin Kim, Jeonghyun Ahn, Junhwi Bae, Xingyou Song, Trieu H. Trinh, Quoc V. Le, and Junehyuk Jung. 2025. Towards robust mathematical reasoning. *Preprint*, arXiv:2511.01846.

Arindam Mitra, Hamed Khanpour, Corby Rosset, and Ahmed Awadallah. 2024. Orca-math: Unlocking the potential of slms in grade school math. *arXiv preprint arXiv:2402.14830*.

Niklas Muennighoff et al. 2025. s1: Simple test-time scaling. https://arxiv.org/abs/2501.19393. *Preprint*, arXiv:2501.19393.

Open-R1 Team. 2025. Openr1-math-220k. https://huggingface.co/datasets/open-r1/OpenR1-Math-220k. Open-source distilled math reasoning dataset.

OpenAI. 2024a. Introducing SWE-bench verified we're releasing a human-validated subset of swe-bench that more.

OpenAI. 2024b. Learning to reason with llms.

OpenAI. 2025. Introducing gpt-5.

Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. 2023. Yarn: Efficient context window extension of large language models. *Preprint*, arXiv:2309.00071.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *Preprint*, arXiv:1707.06347.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489.

Ved Sirdeshmukh, Kaustubh Deshpande, Johannes Mols, Lifeng Jin, Ed-Yeremai Cardona, Dean Lee, Jeremy Kritz, Willow Primack, Summer Yue, and Chen Xing. 2025. Multichallenge: A realistic multi-turn conversation evaluation benchmark challenging to frontier llms. *Preprint*, arXiv:2501.17399.

HLE Team. 2025a. Humanity's last exam. *Preprint*, arXiv:2501.14249.

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

Qwen Team. 2025b. Qwen3-235b-a22b-thinking-2507.

Trieu H Trinh, Yuhuai Wu, Quoc V Le, He He, and Thang Luong. 2024. Solving olympiad geometry without human demonstrations. *Nature*, 625(7995):476–482.

Siddarth Venkatraman, Vineet Jain, Sarthak Mittal, Vedant Shah, Johan Obando-Ceron, Yoshua Bengio, Brian R. Bartoldson, Bhavya Kailkhura, Guillaume Lajoie, Glen Berseth, Nikolay Malkin, and Moksh Jain. 2025. Recursive self-aggregation unlocks deep thinking in large language models. *Preprint*, arXiv:2509.26626.

Junlin Wang, Jue Wang, Ben Athiwaratkun, Ce Zhang, and James Zou. 2024. Mixture-of-agents enhances large language model capabilities. *Preprint*, arXiv:2406.04692.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. *Preprint*, arXiv:2203.11171.

Zihan Wang, Siyao Liu, Yang Sun, Hongyan Li, and Kai Shen. 2025. Codecontests+: High-quality test case generation for competitive programming. *Preprint*, arXiv:2506.05817.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-thought prompting elicits reasoning in large language models. *Preprint*, arXiv:2201.11903.

Hao Wen, Yifan Su, Feifei Zhang, Yunxin Liu, Yunhao Liu, Ya-Qin Zhang, and Yuanchun Li. 2025. Parathinker: Native parallel thinking as a new paradigm to scale llm test-time compute. *Preprint*, arXiv:2509.04475.

Tong Wu, Yang Liu, Jun Bai, Zixia Jia, Shuyi Zhang, Ziyong Lin, Yanting Wang, Song-Chun Zhu, and Zilong Zheng. 2025. Native parallel reasoner: Reasoning in parallelism via self-distilled reinforcement learning. *Preprint*, arXiv:2512.07461.

xAI. 2025. Grok 4.

Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. 2024. Agentless: Demystifying llm-based software engineering agents. *Preprint*, arXiv:2407.01489.

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

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. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

Feng Yao, Liyuan Liu, Dinghuai Zhang, Chengyu Dong, Jingbo Shang, and Jianfeng Gao. 2025. Your efficient rl framework secretly brings you off-policy rl training.

Qiying 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, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Weinan Dai, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Mu Qiao, Yonghui Wu, and Mingxuan Wang. 2025. DAPO: An open-source LLM reinforcement learning system at scale. *Preprint*, arXiv:2503.14476.

Z.ai. 2025. Glm-4.6: Advanced agentic, reasoning and coding capabilities.

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. *Preprint*, arXiv:2509.06870.

Tong Zheng, Hongming Zhang, Wenhao Yu, Xiaoyang Wang, Runpeng Dai, Rui Liu, Huiwen Bao, Chengsong Huang, Heng Huang, and Dong Yu. 2025. Parallel-r1: Towards parallel thinking via reinforcement learning. *Preprint*, arXiv:2509.07980.

In this Appendix, we provide more elaboration on the implementation details, experiment results, and qualitative results.

A Initial Checkpoint Derivation

To facilitate effective test-time scaling, we first implement a reasoning-oriented post-training pipeline for Qwen3-8B-Base, comprising Supervised Fine-Tuning (SFT) and Reinforcement Learning with Verifiable Rewards (RLVR).

Reasoning-Oriented SFT. We collect millions of prompts from the open-source community, spanning diverse domains including mathematics, coding, science, software engineering, tool use, logical reasoning, and creative writing. Leveraging these prompts, we distill data from multiple frontier models. Following a rigorous correctness verification process, we curate a dataset comprising 10.4M samples, totaling 61.1B tokens.

To ensure data quality and integrity, we apply two pipes to further filter the dataset. First, we use predefined rules to eliminate low-quality data with degenerate patterns, such as infinite repetition, harmful content, and personally identifiable information. Second, we conduct comprehensive benchmark decontamination to prevent leakage. This involves both exact matching (with digit masking to catch questions featuring only numerical modifications) and N-gram matching ($N = 64$). This two-pipe filtration process yields a refined dataset of 10.2M samples, totaling 59.5B tokens.

We perform SFT on Qwen3-8B-Base using the refined dataset. The training is configured with a maximum sequence length of 64k and a global batch size of 64. We employ a cosine learning rate scheduler with a 200-step warmup phase, where the learning rate peaks at 1×10^{-4} and anneals to a final value of 1×10^{-5}. We implement domain-specific sampling weights for the dataloader, which translate to varying epochs for different domains. In total, the model is trained over $\sim$165B tokens.

Reasoning-Oriented RLVR. Similarly, we source $\sim$500k prompts paired with ground truth from the open-source community, which encompass domains including mathematics, coding, science, logical reasoning, and instruction following. We implement a robust internal RL framework utilizing vLLM for inference and Megatron for training. During each step, we sample 256 prompts and generate 16 responses

per prompt (max length 64k). To address long-tail inference latency, we enable the partial rollout strategy (Team et al., 2025).

For non-coding domains, we employ LLM-as-judge (gpt-oss-120b [1]) to verify response consistency against the ground truth. For coding tasks, we utilize sandboxes to validate code execution against test cases with soft reward. We omit KL divergence constraints (both as reward penalties and loss terms) and entropy loss during training. We use an off-policy PPO algorithm with GAE ($\gamma = 1, \lambda = 1$), while omitting importance sampling. Samples are split into 4 mini-batches per iteration. We set the actor and critic learning rates to 2×10^{-6} and 5×10^{-6}, respectively. Furthermore, we apply the Truncated Importance Sampling ratio (threshold $C = 8$) proposed by Yao et al. (Yao et al., 2025) to mitigate training-inference inconsistency issues. The entire RL phase consists of 200 training iterations.

B PaCoRe Synthesis Prompt Template

In the *Synthesis and Parallel Exploration* stage of the PaCoRe inference pipeline, we map the problem instance x and the set of compact messages M into a structured natural language input using a prompting function $P(x, M)$. The exact serialization template is provided in Table 6. By framing the compact messages as "Reference Responses," we explicitly encourage the model to critically evaluate and synthesize the diverse perspectives accumulated from the previous round.

While the method is described in Section 3.1 in the context of a single-turn problem, this framework is naturally extensible to multi-turn, interactive environments (e.g., dialogues or agentic tool-use loops). In such settings, the pipeline is invoked at any decision point where the model must generate an action. The "Original Problem" slot in the template is populated by the most recent observation or user message, while the preceding interaction history (previous turns, tool outputs) remains intact in the context. This allows PaCoRe to maintain exactly same interface as standard chat or reasoning models, ensuring seamless compatibility with existing ecosystem.

[1] https://huggingface.co/openai/gpt-oss-120b

C PaCoRe Training Data Preparation

C.1 Math Data

Math Competition and Open Source Data. We aggregate math problems from a mix of open-source datasets and competition archives. On the dataset side, we include open source datasets from (Yu et al., 2025; Hu et al., 2025; Guha et al., 2025; Li et al., 2024; Albalak et al., 2025; Mitra et al., 2024; aslawliet, 2024b,a; Open-R1 Team, 2025; LIMO Authors, 2025; Muennighoff et al., 2025; He et al., 2025). On the competition side, we collect all available historical problems (including official solutions where available) from AIME, HMMT, SMT, CMIMC, BRUMO, BMT, CHMMC, DMM, MNYMO, PUMAC, and Math Prize for Girls, using their official online archives (aim; hmm; smt; cmi; bru; bmt; chm; dmm; mny; pum; mat). We exclude AIME 2024/2025, HMMT Feb 2025, BRUMO 2025, SMT 2025, CMIMC 2025, and HMMT Nov 2025 from the training pool because these contests are used as part of the MathArena benchmark (Balunović et al., 2025).

Synthetic Data. Beyond mined data, we add a synthetic component to strengthen large-integer arithmetic, a basic skill that underlies many competition-style solutions and naturally produces diverse numeric answers. We generate 13k synthetic large-integer arithmetic problems using a small set of simple hand-written text templates. For each problem, we independently sample integers A and B uniformly between 10^{11} and 10^{13}, randomly choose one of four operation types—addition, subtraction, multiplication, or modular exponentiation modulo a fixed large prime—and substitute A, B, and the chosen operation into a template to form the natural-language question. We compute the corresponding exact integer answer with standard arithmetic and record it as the ground-truth label. By restricting to integer-valued outputs, each synthetic problem has a single unambiguous label and can be checked easily.

Data Quality Control. We apply several stages of quality control to ensure that the math data are well-posed and rule-checkable. First, we run deterministic rule-based filters that remove problems with embedded images or external links; multi-part questions or prompts that request multiple final answers; and items whose solutions depend on vague prose or open-ended discussion rather than a single numerical or algebraic target. Next, in-house math

```
You are given a problem and a list of reference responses. Your job is to analyze these references and provide your own response.
Original Problem:
{{ original_prompt }}

Reference Responses:
{% for response in ref_responses %}
Reference {{ loop.index }}:
{{ response }}
{% endfor %}

Now, based on the original problem and reference responses above, please provide your own comprehensive solution.
```

Table 6: **Input serialization template for PaCoRe synthesis.** We use this template to embed the current problem x (denoted as `original_prompt`) and the compact message set M (denoted as `ref_responses`) into the model's context. In the degenerate case where the message set is empty ($M = \emptyset$), this template is bypassed, and the original problem input is passed to the model unmodified.

experts spot-check samples and mark common failure types (incorrect official solutions, internally inconsistent statements, or ambiguous wording). Based on these annotations, we construct a 100-sample evaluation set with a 1:1 mix of valid and invalid QA pairs and iteratively refine a prompt for an LLM-based judge. To accommodate the strict formatting requirements of our verifiable reward environment (e.g., single-part questions with a unique final answer, as in math-verify-style checkers), we use gpt-oss-120b with this prompt in a 4-pass unanimity scheme, keeping only items that receive 4/4 positive votes on the QA-validity classification. We tune the prompt until the F1 score on the evaluation set saturates, and in practice this configuration reliably filters out most ill-posed or non-checkable problems. The filtered open-source datasets and competition problems are then combined with the synthetic arithmetic problems described above. We summarize the problem counts from each source after rule-based and model-based filtering in Table 7. As discussed in Section 3.1, we additionally apply an accuracy-based filter using a strong proposer model to select problems that are neither trivial nor degenerate; we treat this as part of curriculum design rather than basic data cleaning.

Table 7: Problem counts by source after rule-based and model-based filtering.

Source	Before Filtering	Stage 1	Stage 2
Open-source datasets	695k	0.7k	0.5k
Competition archives	8.4k	0.2k	0.3k
Synthetic arithmetic	13.4k	0.8k	1.6k
Total	717k	1.7k	2.4k

C.2 Competitive Code Data

Competitive Programming Contest and Open Source Data. About 29k problems are gathered from competitive programming problem sources (for example, subsets of TACO (Li et al., 2023), USACO [2], etc) and undergo a rigorous cleaning process. The main metrics in validation are format checks in problem statements, number of test cases, and full judging of provided code submissions. We use testlib [3] library to help judge a submission code. We apply an auto-matching method on test case outputs of problems to decide a pre-defined testlib checker to judge. We make some modification to the checker to handling broader cases, including change 'icmp' function to support 64-bit integer comparision. For problems requiring a special-judge checker, we use LLMs to generate it, or simply skip this problem. There are about 14k additional problems sampled from recent open-source competitive programming datasets, including am-thinking-v1 (Ji et al., 2025) and deepcoder (Luo et al., 2025). These problems are filtered with deduplication and validation processes.

Synthetic Data. To address the shortage of test cases in open-source datasets, we employ a generator-validator pipeline inspired by CodeContests+ (Wang et al., 2025). This process utilizes LLMs to produce new test cases, which are subsequently verified against ground-truth solutions and incorrect submissions. In RL experiments, we found that a fine-grained reward function based on test case pass rates significantly outperforms a simple binary reward. Our final code data source comprises approximately 5k problems curated from

[2] https://usaco.org/index.php?page=training
[3] https://github.com/MikeMirzayanov/testlib

Table 8: Ablation over the intermediate round width K_2 during inference, using a fixed multi-round configuration of $\vec{K} = [32, K_2]$.

K_2	**HMMT 2025**	**LiveCodeBench(2408-2505)**
2	94.0	77.4
4	**94.6**	**78.4**
8	94.0	77.5

CodeForces [4].

D More Ablation Studies

Multi-round Inference Recipe. We study the impact of the inference trajectory configuration $\vec{K}$ in a multi-round setting. Fixing the first round width $K_1 = 32$, we ablate the second round width K_2 (using a configuration $\vec{K} = [32, K_2]$). Table 8 indicates that $K_2 = 4$ yields the best performance on both benchmarks.

E Potential Risks

Although we deliberately filtered harmful content from our training data, our reasoning-oriented model may still generate harmful or biased outputs. Furthermore, since we apply no direct supervision to the intermediate reasoning steps, the model's internal chain of thought remains unconstrained and could potentially contain harmful or biased content, even if the final answer is correct.

[4] https://codeforces.com