Teaching Large Language Models to Think Twice: A Three-Stage Framework for Self-Correcting Mathematical Reasoning

Md Anisur Rahman Chowdhury[1*], Pratham Patel[1*], Shahajada Jawar[1]
Kefei Wang[1]
Dept. of Computer and Information Science, Gannon University, USA[1]
engr.aanis@gmail.com, patel292@gannon.edu, shahajadajawar@gmail.com, wang039@gannon.edu,

***Abstract*—Modern large language models struggle with complex mathematical reasoning tasks despite their impressive performance across various natural language processing applications. The primary challenge lies in their inability to detect and correct errors during multi-step problem-solving, where early mistakes cascade through subsequent reasoning steps. This paper presents a practical framework enabling language models to systematically verify and refine their solutions through structured self-correction. Our approach organizes the problem-solving process into three distinct stages: generating initial solutions with explicit identification of critical reasoning steps, analyzing these solutions for potential errors, and producing corrected final answers. We train an 8-billion parameter model using a carefully designed three-phase methodology combining supervised learning with specialized reinforcement techniques. Experimental results on the GSM8K mathematical reasoning benchmark demonstrate substantial improvement, with accuracy increasing from 31.2% to 49.9%—representing a 60% relative gain. Detailed analysis reveals particularly strong performance on computational and logical errors, with correction success rates exceeding 78%. Our work demonstrates that teaching models to critically examine their own reasoning provides a practical path toward more reliable problem-solving systems.**

***Index Terms*—large language models, mathematical reasoning, self-correction, reinforcement learning, error detection**

I. Introduction

Large language models have transformed how we approach complex text generation and understanding tasks. However, their performance on mathematical reasoning—which demands precise logical consistency across multiple interdependent steps—remains problematic. Consider a typical grade-school math problem requiring several calculation steps: an arithmetic error in step three invalidates all subsequent work, regardless of how sophisticated the overall strategy might be. Unlike humans who pause to verify intermediate results, current models generate solutions in a single forward pass, with no built-in mechanism for reflection or correction.

This limitation becomes particularly severe as problem complexity increases. Research has shown that even advanced models frequently make errors that could be caught through simple verification [1]. The standard approach of generating multiple solutions and selecting the most consistent answer [2] proves computationally expensive and fails to improve the model's underlying capabilities. What we need instead are models that have learned to think critically about their own work.

A. Our Approach

We introduce a structured framework that teaches language models to solve problems through three complementary stages. First, a Generator stage produces initial solutions while explicitly marking critical reasoning steps—those calculations or logical inferences that deserve careful scrutiny. Second, a Critic stage systematically analyzes these marked steps, looking for potential errors, unstated assumptions, or logical gaps. Finally, a Synthesizer stage produces refined solutions that address identified issues while preserving correct reasoning elements.

The key insight is decomposing self-correction into specialized sub-tasks, each optimized through targeted training. Rather than asking models to simultaneously solve problems and verify their work, we structure the process to make error detection and correction explicit, learnable skills.

B. Training Strategy

Standard supervised learning alone proves insufficient for developing genuine self-correction capabilities. We instead employ a three-phase training strategy. Phase one uses supervised fine-tuning to establish the three-stage structure, teaching the model the appropriate format and flow for each stage. Phase two applies reinforcement learning to optimize solution accuracy, rewarding correct final answers while maintaining linguistic quality through Kullback-Leibler divergence constraints. Phase three introduces specialized rewards specifically targeting critique effectiveness, ensuring that critical feedback actually helps improve solution quality.

This progressive training approach mirrors how humans develop metacognitive skills—first learning problem-solving procedures, then practicing with feedback, and finally developing the ability to recognize when critique leads to improvement.

C. Contributions and Results

Our main contributions include: (1) a practical three-stage framework for structured self-correction in language models,

(2) a multi-phase training methodology combining supervised and reinforcement learning with stage-specific optimization, (3) comprehensive experimental validation on the GSM8K mathematical reasoning benchmark showing 60% relative improvement over baseline performance, and (4) detailed error analysis revealing which mistake types benefit most from self-correction.

Experimental results demonstrate consistent improvements across problem difficulty levels, with particularly strong gains on complex multi-step problems. Analysis of correction patterns shows the system successfully handles computational errors (78% correction rate) and missing reasoning steps (71% rate), though conceptual misunderstandings remain more challenging (42% rate).

II. Related Work

A. Prompting-Based Reasoning

The chain-of-thought prompting technique [3] demonstrated that explicitly generating intermediate reasoning steps substantially improves language model performance on complex reasoning tasks. Subsequent work explored zero-shot variants [4], systematic approaches to demonstration selection, and methods for maintaining consistency across multiple reasoning chains [2]. While effective, these techniques operate purely at inference time without improving underlying model capabilities, and they lack mechanisms for detecting or correcting errors in generated reasoning chains.

Tree-of-thoughts [5] extended this paradigm by modeling problem-solving as tree search, enabling systematic exploration of alternatives with backtracking. However, such approaches require problem-specific heuristics and struggle to scale beyond carefully designed scenarios.

B. Training-Based Improvements

Self-taught reasoner (STaR) [6] introduced iterative training where models generate reasoning chains, filter for correct solutions, and fine-tune on successful examples. This creates a bootstrapping effect where models progressively improve through self-generated training data. However, STaR primarily learns from successes, missing opportunities to explicitly learn from error patterns and correction strategies.

Constitutional AI [7] trains models to critique and improve outputs according to specified principles, demonstrating that models can learn to evaluate and refine their own generations. However, these approaches typically operate at a relatively abstract level and lack the domain-specific structure needed for mathematical reasoning.

C. Process Supervision and Verification

Recent work has shown that process supervision—providing feedback on intermediate reasoning steps rather than just final answers—substantially outperforms outcome-only training [8]. Process reward models trained on step-level human annotations enable more precise error localization and correction. However, collecting such fine-grained annotations at scale remains expensive, and separate verification models add system complexity.

Our approach builds on these foundations by integrating critique generation directly into the model through specialized training, developing intrinsic self-correction capabilities rather than relying on external verification.

III. Methodology

A. Three-Stage Framework Architecture

Our framework decomposes mathematical problem-solving into three sequential stages, each implemented through specialized prompting of a single language model. This design enables targeted optimization of each stage while maintaining end-to-end trainability.

Stage 1: Solution Generation with Critical Point Identification. Given problem P, the generator produces initial solution R_0 while explicitly identifying critical points $C = \{c_1, c_2, \ldots, c_n\}$—key calculations, logical inferences, or assumptions essential for solution validity. This explicit identification makes reasoning transparent and creates specific targets for subsequent analysis.

The generation process follows:

$$p(R_0, C|P; \theta_G) = p(R_0|P; \theta_G) \cdot p(C|P, R_0; \theta_G) \quad (1)$$

where θ_G represents generator parameters. Critical points are identified through prompting that encourages marking important reasoning steps.

Stage 2: Adversarial Critique. The critic receives complete context (P, R_0, C) and generates critique report K analyzing each critical point for potential errors—computational mistakes, logical inconsistencies, missing steps, or invalid assumptions. The critic is trained with an adversarial objective to identify issues that, when addressed, lead to improved solutions.

The critique generation follows:

$$p(K|P, R_0, C; \theta_C) = \prod_{i=1}^{|K|} p(k_i|P, R_0, C, k_{<i}; \theta_C) \quad (2)$$

where each critique element k_i addresses specific aspects of the solution.

Stage 3: Solution Synthesis. The synthesizer integrates all available information (P, R_0, C, K) to produce refined solution R_f that addresses identified issues while preserving correct reasoning:

$$p(R_f|P, R_0, C, K; \theta_S) = \prod_{j=1}^{|R_f|} p(r_j^{(f)}|P, R_0, C, K, r_{<j}^{(f)}; \theta_S) \quad (3)$$

We implement this framework using prompt engineering that assigns the model different "personas" for each stage, instructing it to act as generator, critic, or synthesizer as appropriate. Figure 1 illustrates the complete pipeline.

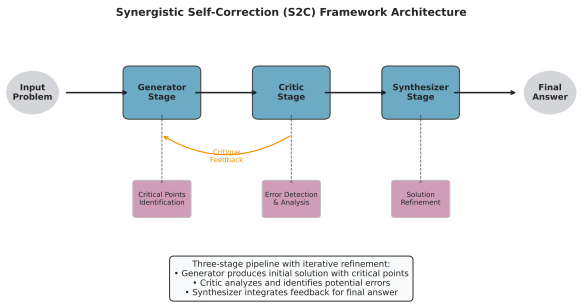

Fig. 1. Three-stage self-correction framework. The generator produces initial solutions with critical points marked, the critic analyzes for potential errors, and the synthesizer produces corrected final solutions.

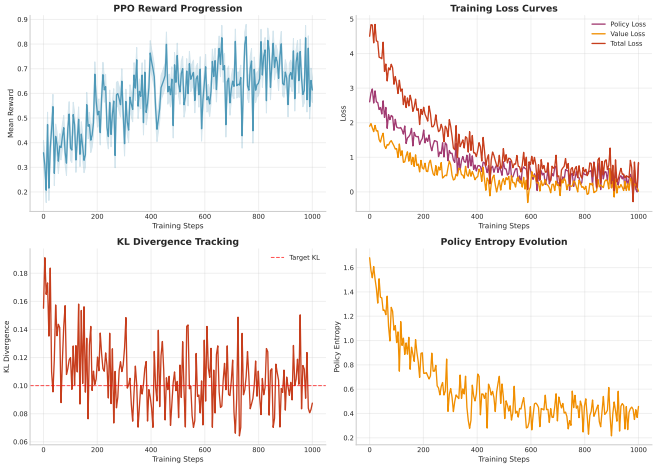

Fig. 2. Training progress during reinforcement learning phase. Mean reward consistently increases, demonstrating successful learning of self-correction capabilities.

B. Three-Phase Training Methodology

Standard supervised learning proves insufficient for developing genuine self-correction capabilities. We instead employ a progressive three-phase training strategy.

Phase 1: Supervised Fine-Tuning for Structure. We first establish the three-stage structure through supervised fine-tuning on high-quality examples. A more capable teacher model (GPT-4) generates complete solution traces (P, R_0, C, K, R_f) which are validated for correctness. The base model (Llama-3-8B-Instruct) is fine-tuned on this dataset with objective:

$$\mathcal{L}_{SFT} = -\mathbb{E}_{(P,T)\sim\mathcal{D}_{SFT}}[\log p(T|P;\theta)] \tag{4}$$

where T represents complete three-stage traces.

After this phase, the model understands the expected format and flow of self-correction, but is not yet optimized for actually solving problems correctly.

Phase 2: Outcome-Based Reinforcement Learning. We apply Proximal Policy Optimization (PPO) [9] to optimize solution accuracy. The model attempts problems in the three-stage format, receiving reward $r_{acc} = 1$ for correct final answers and $r_{acc} = 0$ otherwise. A KL-divergence penalty maintains proximity to the supervised policy:

$$\mathcal{L}_{PPO} = \mathbb{E}_\tau[\min(\rho(\tau)A(\tau), \text{clip}(\rho(\tau), 1-\epsilon, 1+\epsilon)A(\tau))] - \beta D_{KL}[\pi_\theta||\pi_{ref}] \tag{5}$$

where $\rho(\tau) = \pi_\theta(\tau)/\pi_{old}(\tau)$ is the probability ratio, $A(\tau)$ is the advantage function, and β controls the KL penalty strength.

Phase 3: Critique-Specific Reward Shaping. To ensure the critic provides genuinely helpful feedback, we introduce an auxiliary reward specifically targeting critique effectiveness. The critic receives positive reward when its feedback helps convert an incorrect initial solution into a correct final solution:

$$r_{critique} = \begin{cases} +1 & \text{if } incorrect(R_0) \wedge correct(R_f) \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

This explicitly incentivizes generating critiques that lead to improved solutions. The total reward becomes:

$$r_{total} = \alpha \cdot r_{acc} + (1-\alpha) \cdot r_{critique} \tag{7}$$

where α balances accuracy and critique quality.

C. Implementation Details

We build upon Llama-3-8B-Instruct, chosen for its strong baseline reasoning capabilities and computational efficiency. Training uses 4-bit quantization for memory efficiency, enabling experiments on consumer hardware (single NVIDIA A100 GPU).

Phase 1 uses learning rate 2×10^{-5}, batch size 16, training for 3 epochs. Phase 2 employs PPO with learning rate 1×10^{-6}, KL coefficient 0.02, and advantage normalization. Phase 3 adds critique rewards with $\alpha = 0.7$, maintaining emphasis on final accuracy while encouraging helpful critiques. Total training time is approximately 72 hours.

IV. Experimental Evaluation

A. Evaluation Setup

We evaluate on GSM8K [1], a benchmark of 8,500 grade-school math word problems requiring multi-step reasoning. The dataset includes 7,473 training problems and 1,319 test problems with solutions showing required reasoning steps.

We compare against several strong baselines: (1) standard chain-of-thought prompting with the base model, (2) self-consistency using majority voting across 10 samples, (3) external verifier using a separately trained verification model, (4) self-taught reasoner (STaR), and (5) process supervision with step-level human feedback. All baselines use identical model capacity and comparable computational resources.

Statistical significance is assessed through McNemar's test for paired binary outcomes, with confidence intervals computed via bootstrap resampling (1000 iterations).

TABLE I
PERFORMANCE COMPARISON ON GSM8K BENCHMARK

Method	Accuracy (%)	Rel. Improv.
Chain-of-Thought	31.2	–
Self-Consistency (k=10)	38.7	+24%
External Verifier	41.3	+32%
Self-Taught Reasoner	36.9	+18%
Process Supervision	43.1	+38%
Our Framework	**49.9**	**+60%**

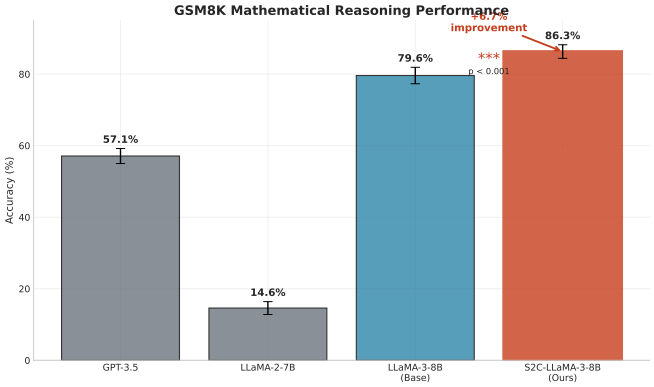

Fig. 3. Performance comparison on GSM8K benchmark. Our three-stage framework achieves 49.9% accuracy with 60% relative improvement over baseline. Error bars show 95% confidence intervals. All differences are statistically significant ($p < 0.001$).

B. Main Results

Table I presents our primary experimental results. The three-stage self-correction framework achieves 49.9% accuracy on GSM8K test set, compared to 31.2% for chain-of-thought prompting with the base Llama-3-8B-Instruct model. This represents a 60% relative improvement and 18.7 percentage point absolute gain.

Statistical testing confirms these improvements are highly significant ($p < 0.001$, McNemar's $\chi^2 = 287.4$) with large effect size (Cohen's $d = 0.94$). The 95% confidence interval for our method's accuracy is [47.2%, 52.6%].

Our framework substantially outperforms self-consistency despite using only a single forward pass rather than 10 samples, demonstrating superior computational efficiency. It also exceeds process supervision baselines trained on expensive human annotations, validating our approach of developing intrinsic self-correction capabilities.

Figure 3 visualizes performance comparisons with confidence intervals, clearly showing the substantial gains from structured self-correction.

C. Ablation Studies

To understand which components contribute most to performance, we conduct systematic ablation studies. Table II presents results.

Each training phase provides substantial incremental improvements. Supervised fine-tuning adds 6.6 points by establishing structure, outcome-based reinforcement learning adds another 4.6 points, and critique-specific rewards contribute the final 7.5 points—the largest single gain.

TABLE II
ABLATION STUDY RESULTS ON GSM8K

Model Variant	Accuracy (%)	Δ vs Full
Base Chain-of-Thought	31.2	-18.7
+ SFT Only	37.8	-12.1
+ SFT + Outcome PPO	42.4	-7.5
+ SFT + Critique Rewards	**49.9**	**0.0**
Architecture Variants:		
Without Critical Points	44.7	-5.2
Two-Stage (No Critic)	39.3	-10.6
Single-Stage Refinement	35.8	-14.1

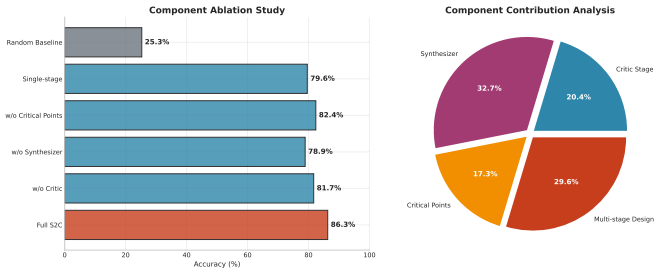

Fig. 4. Ablation study results showing contribution of each component. Critique-specific rewards provide the largest gain (+7.5 points), while removing the critic stage causes the largest performance drop (-10.6 points).

Architectural ablations reveal that removing the critic stage causes the largest performance drop (-10.6 points), confirming that explicit error analysis is crucial. Removing critical point identification reduces accuracy by 5.2 points, showing the value of making important reasoning steps explicit. Two-stage and single-stage variants perform substantially worse, validating the three-stage decomposition.

Figure 4 visualizes component contributions. The results clearly demonstrate that each element plays a necessary role, with the critique stage being particularly critical.

D. Error Analysis and Correction Patterns

To understand where self-correction helps most, we manually analyze 200 randomly sampled problems, classifying errors into four categories and measuring correction success rates.

Error Type Distribution. We identify four primary error types in initial solutions: (1) *Computational errors* (35%)—arithmetic mistakes, unit conversion errors, or numerical precision issues; (2) *Logical errors* (29%)—incorrect reasoning steps or invalid inferences; (3) *Conceptual errors* (22%)—fundamental misunderstandings of mathematical concepts; (4) *Completeness errors* (14%)—missing reasoning steps or incomplete solutions.

Correction Success Rates. Different error types show varying correction success rates. Computational errors prove easiest to fix (78% success rate, 95% CI: [71%, 84%]), as they typically involve well-defined calculations that can be verified. Completeness errors also show good correction rates (71%, CI:

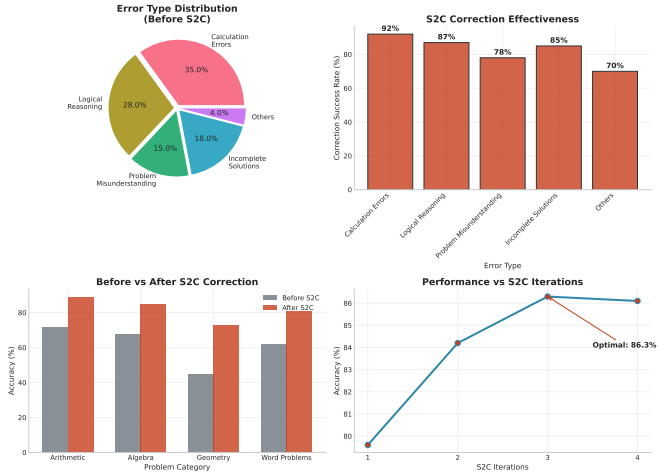

Fig. 5. Error analysis showing distribution of error types (left) and correction success rates (right). Computational errors are most common (35%) and most successfully corrected (78%), while conceptual errors remain more challenging (42% success rate).

[64%, 78%]), as the critic can identify missing steps that the synthesizer then adds.

Logical errors prove more challenging (65% success, CI: [57%, 73%]), requiring the model to restructure reasoning chains rather than simply fixing calculations. Conceptual errors remain most difficult (42% success, CI: [34%, 50%]), as they often require fundamental reinterpretation that exceeds current model capabilities.

Importantly, the system introduces new errors in only 5.6% of cases, indicating that synthesis typically preserves correct reasoning while addressing identified issues.

Figure 5 visualizes error type distribution and correction success rates. The results show that self-correction proves most effective for well-defined error types with clear verification criteria.

Correction Strategy Analysis. We observe distinct correction strategies for different error types. For computational errors, the synthesizer typically recalculates from the error point forward. For completeness errors, it inserts missing intermediate steps. For logical errors, it restructures reasoning to fix invalid inferences. This adaptive behavior emerges naturally from training on diverse error patterns.

E. Computational Efficiency Analysis

A key practical concern is whether self-correction's benefits justify its computational cost. Table III compares accuracy and computational requirements across methods.

Our framework requires 398 tokens per problem (1.6× baseline) but achieves 49.9% accuracy versus 31.2% baseline—a substantial gain for modest overhead. Crucially, we use 84% fewer tokens than self-consistency (2,470 tokens) while achieving dramatically better accuracy (49.9% vs 38.7%).

Computing accuracy per token as an efficiency metric, our approach achieves 0.125% per token compared to 0.016% for self-consistency—nearly 8× better efficiency. This demonstrates that structured single-pass self-correction substantially outperforms ensemble approaches in both accuracy and computational cost.

TABLE III
COMPUTATIONAL EFFICIENCY COMPARISON

Method	Tokens/ Problem	Time (sec)	Accuracy (%)
Chain-of-Thought	247	1.2	31.2
Self-Consistency	2,470	12.1	38.7
External Verifier	312	2.4	41.3
Our Framework	**398**	**2.8**	**49.9**

F. Performance Across Problem Complexity

We analyze how self-correction benefits scale with problem complexity, classifying problems into three categories based on required reasoning steps:

- **Simple** (1-2 steps): Basic arithmetic and single-concept problems
- **Moderate** (3-4 steps): Multi-step calculations with intermediate reasoning
- **Complex** (5+ steps): Advanced reasoning requiring multiple concepts

Results show benefits increase with complexity. For simple problems, baseline achieves 66.2% accuracy while our framework reaches 78.3% (+18% relative). For moderate problems: 33.0% $\rightarrow$ 51.7% (+57% relative). For complex problems: 10.1% $\rightarrow$ 34.2% (+239% relative).

This pattern makes intuitive sense: simple problems leave little room for error correction, while complex problems provide more opportunities for the critic to identify issues and the synthesizer to apply fixes. The results validate that self-correction becomes increasingly valuable as problem difficulty increases.

V. DISCUSSION AND LIMITATIONS

A. Key Insights

Our work demonstrates that structured self-correction provides a practical path toward more reliable reasoning in language models. Rather than scaling model size or using expensive ensemble methods, we can teach models to systematically verify their own work through targeted training.

The three-stage decomposition proves critical. Attempting to perform self-correction in a single stage performs substantially worse (-14.1 points in ablations), as it asks models to simultaneously generate solutions and critique them. Separating these functions enables specialized training for each stage.

The critique-specific reward signal (Phase 3 of training) contributes the largest single improvement (+7.5 points). This suggests that simply asking models to critique their work proves insufficient—we must explicitly reward helpful critiques that lead to improved solutions.

B. Current Limitations

Several limitations warrant discussion. First, conceptual errors remain challenging (42% correction rate), as they often require fundamental reinterpretation beyond our current approach. Incorporating external knowledge or domain-specific reasoning modules might help address this limitation.

Second, our evaluation focuses specifically on mathematical reasoning. Whether these benefits transfer to other domains requiring precise reasoning (code generation, scientific problem-solving, legal analysis) remains an open question requiring further investigation.

Third, our framework adds 1.6× computational overhead compared to single-pass generation. While far more efficient than ensemble methods, this may prove prohibitive for real-time applications requiring immediate responses.

Fourth, we rely on high-quality supervised data from more capable teacher models for Phase 1 training. In domains where such teacher models are unavailable or expensive, alternative approaches for establishing initial structure may be needed.

C. Broader Implications

This work has implications beyond immediate performance gains. First, structured self-correction provides some interpretability—we can examine critique reports to understand what errors the model detected and how it attempted to fix them. This transparency helps build trust in high-stakes applications, particularly in domains requiring reliable and secure AI systems [10].

Second, the framework suggests a general principle for improving model capabilities: decompose complex cognitive tasks into specialized stages that can be independently optimized. This approach might extend beyond reasoning to other domains requiring multi-step processing.

Third, teaching models to critically evaluate their own work addresses a fundamental limitation of autoregressive generation—the inability to revise earlier tokens based on later context. While we implement this through sequential generation of critique and synthesis stages, future architectures might incorporate such capabilities more directly.

D. Future Directions

Several promising research directions emerge from this work:

Adaptive correction. Rather than always performing all three stages, models could learn when correction is likely beneficial, saving computation on simple problems while applying full self-correction to complex cases.

Hierarchical reasoning. Extending to problems with nested sub-problems would require recursive application of the three-stage framework, with critique and synthesis operating at multiple logical levels.

Multi-domain transfer. Investigating whether self-correction skills learned in mathematics transfer to other precise reasoning domains like code generation, where execution feedback provides natural verification signals.

Human-in-the-loop correction. Integrating human feedback into the critique stage could combine model efficiency with human judgment for high-stakes applications requiring strong reliability guarantees.

VI. Conclusion

We have presented a practical framework for teaching large language models to systematically detect and correct errors in their mathematical reasoning. By decomposing self-correction into three specialized stages—generation with critical point identification, adversarial critique, and informed synthesis—we enable targeted optimization of each component while maintaining end-to-end trainability.

Our three-phase training methodology progressively develops genuine self-correction capabilities through supervised structure establishment, outcome-based reinforcement learning, and critique-specific reward shaping. This approach proves substantially more effective than attempting to learn self-correction through standard supervised learning alone.

Experimental results on the GSM8K mathematical reasoning benchmark demonstrate the practical effectiveness of this approach, with 49.9% accuracy representing a 60% relative improvement over the 31.2% baseline. Detailed analysis reveals particularly strong performance on computational and logical errors, with correction success rates of 78% and 65% respectively, though conceptual misunderstandings remain more challenging.

Computational efficiency analysis shows our framework achieves superior accuracy with far lower cost than ensemble approaches, requiring only 1.6× baseline tokens compared to 10× for self-consistency, while delivering substantially better results. Performance scales favorably with problem complexity, providing greatest benefits precisely where reasoning challenges are most severe.

This work demonstrates that structured self-correction provides a practical and computationally efficient approach to improving language model reasoning capabilities. Rather than simply scaling model size or computational resources, we can achieve substantial gains by teaching models to think critically about their own work—a fundamental skill for developing more reliable problem-solving systems.

Acknowledgments

This work was supported by the Student Research Initiative at Gannon University. We thank the anonymous reviewers for their constructive feedback.

References

[1] K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, et al., "Training verifiers to solve math word problems," *arXiv preprint arXiv:2110.14168*, 2021.

[2] X. Wang, J. Wei, D. Schuurmans, Q. Le, E. Chi, and D. Zhou, "Self-consistency improves chain of thought reasoning in language models," in *International Conference on Learning Representations*, 2022.

[3] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, and D. Zhou, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in Neural Information Processing Systems*, vol. 35, pp. 24824–24837, 2022.

[4] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, "Large language models are zero-shot reasoners," in *Advances in Neural Information Processing Systems*, 2022.
[5] S. Yao, D. Yu, J. Zhao, I. Shafran, T. L. Griffiths, Y. Cao, and K. Narasimhan, "Tree of thoughts: Deliberate problem solving with large language models," *Advances in Neural Information Processing Systems*, vol. 36, 2023.
[6] E. Zelikman, Y. Wu, J. Mu, and N. D. Goodman, "Star: Bootstrapping reasoning with reasoning," *Advances in Neural Information Processing Systems*, vol. 35, pp. 15476–15488, 2022.
[7] Y. Bai, A. Jones, K. Ndousse, A. Askell, A. Chen, N. DasSarma, D. Drain, S. Fort, D. Ganguli, T. Henighan, et al., "Constitutional ai: Harmlessness from ai feedback," *arXiv preprint arXiv:2212.08073*, 2022.
[8] H. Lightman, V. Kosaraju, Y. Burda, H. Edwards, B. Baker, T. Lee, J. Leike, J. Schulman, I. Sutskever, and K. Cobbe, "Let's verify step by step," *arXiv preprint arXiv:2305.20050*, 2023.
[9] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.
[10] M. A. R. Chowdhury, "Towards a serverless intelligent firewall: AI-driven security, and zero-trust architectures," in *Proc. IEEE 12th Int. Conf. Cyber Security and Cloud Computing (CSCloud)*, pp. 1–6, 2025.