Not All Errors Are Created Equal: ASCoT Addresses Late-Stage Fragility in Efficient LLM Reasoning

Anonymous ACL submission

Abstract

While Chain-of-Thought (CoT) prompting empowers Large Language Models (LLMs), ensuring reasoning reliability remains an open challenge. Contrary to the prevailing cascading failure hypothesis which posits that early errors are most detrimental, we identify a counter-intuitive phenomenon termed **Late-Stage Fragility**: errors introduced in later reasoning stages are significantly more prone to corrupting final answers. To address this, we introduce ASCoT (Adaptive Self-Correction Chain-of-Thought), a method harmonizing efficiency with robust verification. ASCoT first employs semantic pruning to compress redundant steps, then utilizes an Adaptive Verification Manager (AVM) to prioritize high risk, late-stage steps via a positional impact score, triggering a Multi-Perspective Self-Correction Engine (MSCE) only when necessary. Experiments on GSM8K and MATH-500 demonstrate that ASCoT effectively reallocates computational resources: it reduces token usage by 21%–30% for LLaMA-3.1-8B with negligible accuracy drops ($< 1.8\%$), achieving a superior trade-off between inference efficiency and reasoning fidelity.

1 Introduction

The advent of CoT has endowed LLMs with remarkable multi-step reasoning capabilities (Kojima et al., 2022), enabling them to deconstruct complex problems into a sequence of manageable subproblems (Shinn et al., 2023). Recent studies, such as the o1 from OpenAI (Jaech et al., 2024) and DeepSeekR1 (Guo et al., 2025) have shown that extended CoT reasoning improves LLM performance. However, the success of CoT and its variants has introduced challenges: prohibitive computational costs from long reasoning chains (Chen et al., 2025) and the inherent brittleness of the reasoning process itself (Ke et al., 2025). A single flaw is often sufficient to invalidate an entire process of thought,

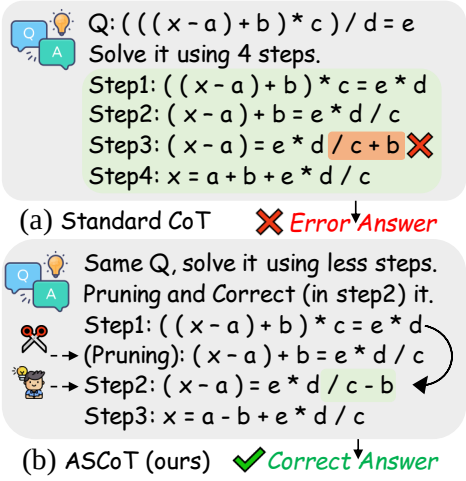

Figure 1: The top part (a) illustrates the standard CoT reasoning process, which often generates redundant outputs and lacks an effective self-correction mechanism when errors occur. The bottom part (b) depicts the ASCoT method, which incorporates a robust mechanism for self-correction, addressing potential reasoning flaws at each stage.

raising significant concerns about the trustworthiness of these models (Pan et al., 2024).

A widely held belief within the research community, often termed the cascading failure hypothesis, posits that the primary threat to CoT integrity lies in early-stage errors (Tyen et al., 2023). The logic is intuitive: an initial mistake in problem comprehension or a miscalculation will inevitably propagate through the logical dependency graph of the reasoning chain, contaminating all subsequent steps. In this work, we directly investigate this foundational assumption, seeking to answer a critical question: Are all errors in a CoT created equal in their impact? To find out this, we conduct a systematic error-injection experiment, which introduces controlled symbolic and numeric errors at varying positions within correct reasoning chains. We then instruct the LLM to resume the reasoning process

from the point at which the error occurred. Our findings are striking and counter-intuitive. We discover that errors introduced in later stages are more likely to lead to an incorrect final answer. We term this phenomenon Late-Stage Fragility. An early error often appears to trigger the model's latent self-correction mechanisms, allowing it to recover and still reach the correct solution.

As the model progresses steadily, its reasoning process gains a form of semantic commitment, reducing relevant information entropy. At the later stages, the model is unable to evaluate the validity of the final calculations. This empirical discovery suggests that a significant portion of efforts to secure CoT reasoning may be misdirected. As illustrated in Figure 1, a standard CoT process can be both inefficient due to redundancy and unreliable due to its inability to correct these late-stage errors. Building on a rethinking of how we evaluate and how we correct LLM reasoning, we introduce the ASCoT method. In contrast to standard methods, ASCoT is designed to address both challenges simultaneously, it first employs a semantic pruning stage to enhance efficiency, followed by a targeted, risk-aware correction mechanism that focuses computational resources precisely on the most vulnerable steps to ensure a correct final answer.

Our contributions are summarized as follows:

- To the best of our knowledge, this is the first to identify and quantify the Late-Stage Fragility phenomenon in CoT reasoning, demonstrating that errors in later steps are significantly more detrimental than those in earlier steps.
- We introduce ASCoT, a novel method specifically designed to counteract this fragility by precisely identifying and robustly correcting the high-impact, late-stage errors.
- Our extensive experiments validate the effectiveness of ASCoT. We demonstrate that ASCoT achieves outstanding accuracy on the GSM8k and MATH benchmarks.

2 Related Work

2.1 LLM Reasoning & Efficient Thinking

CoT prompting significantly advances LLM reasoning by decomposing complex problems into intermediate steps (Liu et al., 2024b; Team, 2024). To mitigate the associated computational costs, recent research explores diverse efficiency-focused strategies, including prompt-based constraints (*e.g.*, token budgets) (Xu et al., 2025; Han et al., 2024), inference-time dynamic early-exit mechanisms (Ding et al., 2025), and CoT distillation (Wadhwa et al., 2024).

2.2 CoT Compression and Pruning

The inherent verbosity of CoT incurs significant computational overhead (Wingate et al., 2022) and can induce the overthinking phenomenon, where excessive reasoning introduces errors (Kumar et al., 2025). To mitigate this, recent research focuses on compression strategies (Xuefei et al., 2023). Approaches vary from token-level pruning based on semantic importance (Zhu et al., 2025; Yang et al., 2025) to structural methods like Prune-on-Logic (Liu et al., 2025), which filters steps via logic graphs. Notably, even heuristic truncation (Wang et al., 2024) has demonstrated efficacy, suggesting that exhaustive reasoning paths are often redundant.

2.3 Self-correction Mechanism

Self-correction mechanisms enhance LLM reliability by iterating through generation, feedback, and refinement phases (Kamoi et al., 2024). Current approaches bifurcate based on the feedback source. Intrinsic self-correction relies on the model's internal critique capabilities without external tools (Yang et al., 2025; Zhu et al., 2025). However, these methods often struggle to identify errors due to the scarcity of correction examples in training data (Liu et al., 2024a). In contrast, extrinsic self-correction leverages external signals, such as verifier models (Kim et al., 2025) or consistency checks, to rigorously assess step-wise correctness.

3 Preliminary

This section establishes the foundational concepts that motivate our work, offering a review of some effective methods for identifying token importance.

Efficient CoT in LLM Reasoning A critical question for CoT efficiency is whether every token in different position contributes equally to the final answer (Xia et al., 2025). Several early approaches (Hou et al., 2022) measure importance based on the self-information or surprisal from a unidirectional language model:

$$I_1(x_i) = -\log P(x_i|x_{<i}; \theta_{\mathcal{M}_L}), \qquad (1)$$

where x_i is a given token, $\theta_{\mathcal{M}_L}$ denotes the LLM used to compute the confidence and $P(\cdot \mid \cdot; \theta_{\mathcal{M}_L})$ is the conditional probability assigned by the causal

LLMs. This metric suffers from known limitations, such as positional dependency, where tokens at the end of sentences often receive lower importance scores due to the nature of perplexity in LLMs.

To address this, more advanced methods like LLMLingua-2 (Pan et al., 2024) utilize a bidirectional language model (Devlin et al., 2019) ($e.g.$, BERT-style) to reframe the task as token classification. In this paradigm, the importance of a token x_i is defined as the probability that a trained classifier assigns it an essential label ($y_i = 1$), conditioned on the entire sequence x:

$$I_2(x_i) = P(y_i = 1|x; \theta_{\mathcal{M}cls}), \tag{2}$$

where $\theta_{\mathcal{M}cls}$ represents the parameters of the trained bidirectional classifier. Such methods confirm that tokens within a CoT sequence are variable in importance. For example, tokens comprising mathematical equations or key entities (Ma et al., 2025) are identified as critical, whereas semantic connectors such as "so" and "since" contribute less to the final outcome of the reasoning (Hou et al., 2025). This highlights that CoT efficiency can be improved by pruning these redundant tokens.

Error Propagation in Reasoning Process Beyond efficiency, the primary challenge for CoT is reliability. A single error can invalidate an entire reasoning process (Guo et al., 2017). The conventional understanding of this problem is rooted in the cascading failure hypothesis, which posits that errors are most damaging when they occur early in the reasoning chain (Jiang et al., 2025). According to this view, an early mistake propagates through the logical dependency graph, contaminating all subsequent steps. In this study, through systematic error-injection experiments, we observe that errors introduced in the later stages of a reasoning chain are significantly more detrimental to the final outcome than identical errors introduced at the beginning.

4 Method

In this section, we detail the architecture of our method. ASCoT generates the $CoT_{initial}$ based on the target LLM. The $CoT_{initial}$ is then processed through the Intelligent Routing Mechanism (IRM) module, reducing the output to a fixed ratio γ. The remaining CoT is passed through the Adaptive Verification Manager (AVM) module, which performs a rigorous verification process. If the model's risk score for any CoT step exceeds a predefined threshold τ, indicating a potential error, the corresponding CoT step is forwarded to the Multi-Perspective Self-Correction Engine (MSCE) module. The MSCE module applies internal and external validation techniques to correct errors, resulting in the corrected CoT_{final}. Ultimately, the refined, non-redundant CoT_{final} is used for fine-tuning. We illustrate the training and inference process in Figure 2, respectively. We also provide Algorithm 1 for ASCoT.

4.1 Intelligent Routing Mechanism

The IRM serves as the initial processing stage, responsible for preliminary CoT pruning and compression. The core idea is inspired by methods TokenSkip (Xia et al., 2025), which recognize that not all tokens in a CoT are equally important for the final reasoning outcome. The mechanism operates by first generating a complete CoT path for a given problem. Then it employs a trained token importance model to assign a semantic importance score to each token in the chain. Based on a predefined compression ratio γ, tokens with scores below the corresponding threshold are pruned, resulting in a compressed CoT, namely the final answer.

4.2 Adaptive Verification Manager

AVM dynamically identifies which steps are most likely to be incorrect and then computes a Risk Score $R(t_k)$ for each step t_k by integrating two key assessments: the step's confidence assessment $Q(t_k)$ and its positional impact $I(k)$.

Confidence Assessment Score To provide a holistic measure of the single reasoning step's quality, we first define a comprehensive quality score $Q(t_k)$, which is a composite metric that evaluates step t_k across four dimensions: Logical Validity $V(t_k)$, Factual Support $G(t_k)$, Semantic Clarity $C(t_k)$ and Process Utility $U(t_k)$.

Logical validity $V(t_k)$ evaluates the internal logical consistency of step t_k. The score is a binary value that indicates if the step t_k is derivable from its context $(P, S_{<k})$, determined by a verification query to the LLM $\mathcal{M}_L$:

$$V(t_k) = \mathbb{I}(o_k^v \in y), \tag{3}$$

where o_k^v is the model's output of step t_k, y is the set of valid responses and $\mathbb{I}(\cdot)$ is the indicator function.

Factual support $G(t_k)$ measures the correctness of all n verifiable assertions within the step. For the mathematical reasoning benchmarks used in

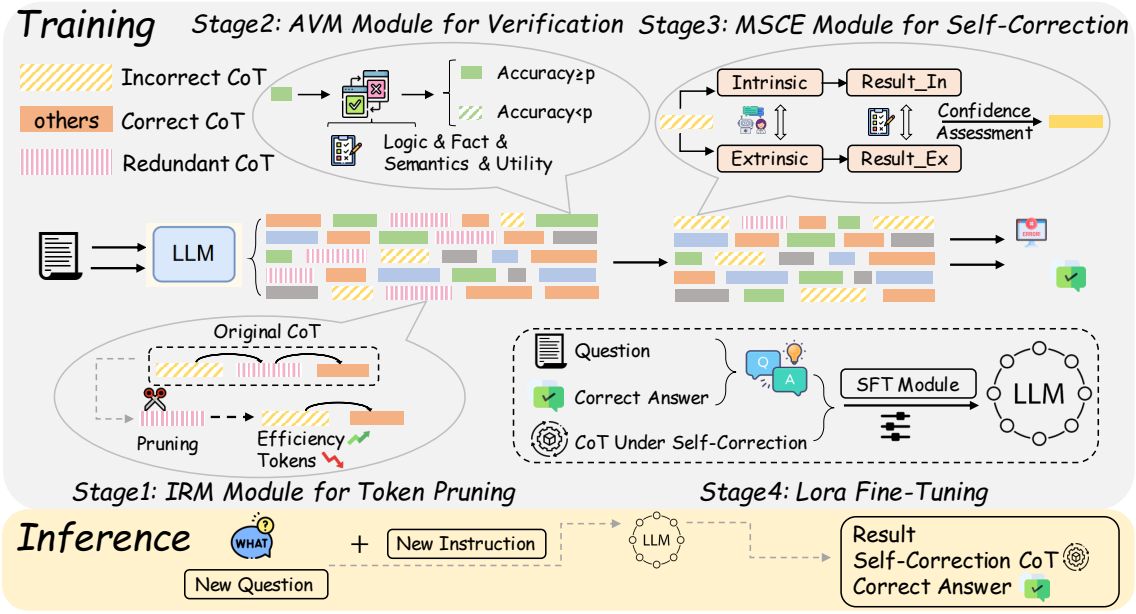

Figure 2: Illustration of ASCoT Pipeline. ASCoT generates the $CoT_{initial}$ from the target LLM, which is then compressed by IRM module to a predefined ratio γ. The compressed CoT is verified by AVM module, and if the confidence score exceeds a threshold τ, the problematic steps are sent to MSCE module for error correction. The corrected CoT_{final} is then used for fine-tuning. Finally, ASCoT enables reasoning on new problems with the specified compression ratio γ.

this work, verifiable assertions are arithmetic calculations. The function is defined as:

$$G(t_k) = \prod_{i=1}^{n} T(a_i), \tag{4}$$

where a_i is the i-th arithmetic operation in step t_k, and $T(a_i) \rightarrow \{0, 1\}$ is a verification function that executes the operation using a computational engine ($e.g.$, a Python interpreter) to confirm its accuracy.

Semantic clarity $C(t_k)$ quantifies the coherence of step t_k. To ensure the score is well-calibrated and bounded within a $[0, 1]$ range, we normalize the average conditional log-probability of its tokens $\{y_1, ..., y_m\}$ using a sigmoid function:

$$C(t_k) = \sigma\left(\frac{1}{m}\sum_{i=1}^{m} \log P(y_i|c_{<i})\right), \tag{5}$$

where $\sigma(\cdot)$ is the sigmoid function, y_i is the i-th token of step t_k, and $c_{<i}$ is the preceding context.

Process utility $U(t_k)$ measures the contribution of step t_k to the final answer $\mathcal{A}^*$. It is defined as the information gain:

$$U(t_k) = \log P(\mathcal{A}^*|S_{\leq k}) - \log P(\mathcal{A}^*|S_{<k}), \tag{6}$$

where $\mathcal{A}^*$ is the ground-truth answer, $S_{\leq k}$ is the full context including step t_k and $S_{<k}$is the context excluding step t_k. $U(t_k)$ relies on the ground-truth answer and is used during the training phase. At inference time, this term is omitted.

Based on these dimensions, we design the single-step comprehensive quality scoring function $Q(t_k)$ with a veto mechanism for foundational criteria:

$$Q(t_k) = V(t_k) \cdot G(t_k) \cdot (w_c \cdot C(t_k) + w_u \cdot U(t_k)), \tag{7}$$

where w_c and w_u are hyperparameters that balance the contribution of clarity and utility. Figure 3 illustrates this evaluation process, where an initial CoT with a calculation error is analyzed.

Positional Impact Score To quantitatively model the Late-Stage Fragility phenomenon observed in our error-injection experiments (Section 4.3), we introduce the Position Impact Function $I(k)$. This function is an empirical model fitted to our experimental data, designed to capture the escalating impact of errors at later reasoning stages. We normalize this position function as:

$$I(k) = w_a \cdot e^{\alpha(k/K)}, \tag{8}$$

where k is the step's index, K is the total number of steps, w_a is a base impact coefficient, and α is a propagation rate. These parameters are determined by fitting the function to our error-injection data. This procedure involves injecting single errors at

different steps k in a set of baseline correct reasoning chains and measuring the probability of final answer failure. The function $I(k)$ is then fitted to these empirical data points. Details of the analysis are available in Appendix E.

Finally, AVM combines these factors into a single Risk Score:

$$R(t_k) = I(k) \times \big(1 - Q(t_k)\big). \tag{9}$$

This score is then compared against a tunable system threshold τ. If $R(t_k) > \tau$, the step is flagged as high risk and passed to the MSCE.

4.3 Multi-Perspective Self-Correction Engine

Once a step t_k is flagged by the AVM, the MSCE is activated to perform a robust and efficient correction. To counteract a model struggles to correct its own mistakes when using a single line of reasoning, the MSCE employs a dual-path correction strategy: Intrinsic Correction, the model is presented with the context up to step $s_{(k-1)}$ and its own flawed step t_k, and is explicitly prompted to review and amend it. Extrinsic Correction, the model is given the context up to $s_{(k-1)}$ but without the flawed step t_k. Upon generating two candidate corrections, the MSCE employs a selection mechanism to choose the superior one. We re-apply the Confidence Assessment Score function $Q(\cdot)$ (from Eq. 7, with $U = 0$) to both candidates. The candidate with the higher quality score is then integrated into the reasoning chain.

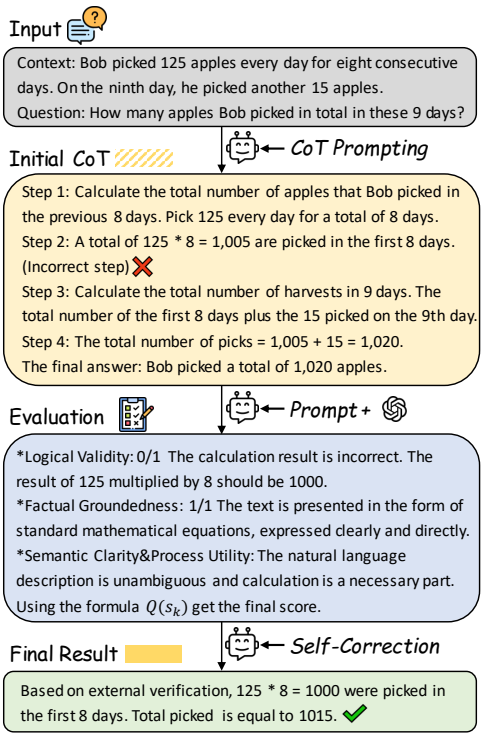

Figure 3: The details of verification and self-correction. AVM and MSCE working together to correct a faulty step t_k. AVM computes $R(t_k)$. MSCE then applies intrinsic and extrinsic correction to produce the final corrected CoT_{final}.

5 Experiment

5.1 Experimental Setup

Models and Datasets We primarily evaluate our method using the LLaMA-3.1-8B-Instruct (Dubey et al., 2024) and Qwen2.5-Instruct series (3B, 7B, 14B) (Team, 2024)models. Our evaluation leverages two notably standard math reasoning benchmarks: GSM8K (Cobbe et al., 2021), which consists of 8.5K grade school math problems requiring 2-8 reasoning steps, and MATH (Hendrycks et al., 2021), a more challenging dataset of 12.5K competition-level math problems. Given the significant computational cost associated with the MATH dataset, we follow recent research practices by evaluating our method on a carefully selected subset, MATH-500 (Lightman et al., 2023), which includes a representative sample of 500 problems.

Implementation Details The ASCoT is implemented in PyTorch 1.13.0 and all experiments are conducted on a server with four NVIDIA GeForce RTX 3090 GPUs (24GB). For the IRM module, we use LLMLingua-2 (Pan et al., 2024) as the token importance metric to generate compressed CoT data for fine-tuning. To optimize the fine-tuning process, we adopt LoRA (Hu et al., 2022) for all fine-tuning procedures, setting the rank $r = 8$ and scaling parameter $\alpha = 16$ to ensure efficient training. Additional implementation details and configurations can be found in the Appendix D.

Evaluation Metrics We evaluate the performance of ASCoT using three primary metrics: Accuracy (%), which represents the percentage of problems for which the final answer is correct. Tokens, the average number of tokens generated in the reasoning chain, measuring computational cost and efficiency during the reasoning process. ActRatio, the actual compression ratio of ASCoT, reflecting the real efficiency of the token compression.

5.2 Main Results

We benchmark ASCoT against two standard length-control baselines: instruction-based Prompting and hard Truncation. The comparative results on LLaMA-3.1-8B are presented in Table 1.

Table 1: Experimental results of ASCoT on LLaMA-3.1-8B-Instruct. We report Accuracy, average CoT token count (Tokens) and actual compression ratio (ActRatio) for comparison. The compression ratio γ is set to $0.5 - 1.0$.

Methods	Ratio	GSM8K			MATH-500		
		Accuracy ↑	Tokens ↓	ActRatio	Accuracy ↑	Tokens ↓	ActRatio
Original	1.0	86.2(0.0↓)	213.17	1.00	48.6(0.0↓)	502.60	1.00
Prompt	0.9	84.1(2.1↓)	226.37	1.06	48.6(0.0↓)	468.04	0.93
	0.7	84.9(1.3↓)	209.39	0.98	48.4(0.4↓)	472.13	0.94
	0.5	83.7(2.5↓)	188.82	0.89	47.8(0.4↓)	471.11	0.94
Truncation	0.9	70.2(26.0↓)	202.06	0.95	47.8(0.8↓)	440.33	0.88
	0.7	25.9(60.3↓)	149.97	0.70	45.0(3.6↓)	386.89	0.77
	0.5	7.0(79.2↓)	103.69	0.49	27.4(21.2↓)	283.70	0.56
ASCoT	1.0	86.9(0.7↑)	214.24	1.00	48.8(0.2↑)	478.80	0.95
	0.9	86.1(0.1↓)	197.65	0.92	47.6(1.0↓)	427.15	0.84
	0.8	85.1(1.1↓)	169.96	0.79	47.2(1.4↓)	376.72	0.74
	0.7	83.7(2.5↓)	151.32	0.71	46.8(1.8↓)	334.26	0.67
	0.6	81.6(4.6↓)	128.38	0.60	41.8(6.8↓)	309.69	0.61
	0.5	79.5(6.7↓)	118.57	0.55	40.1(8.5↓)	288.94	0.57

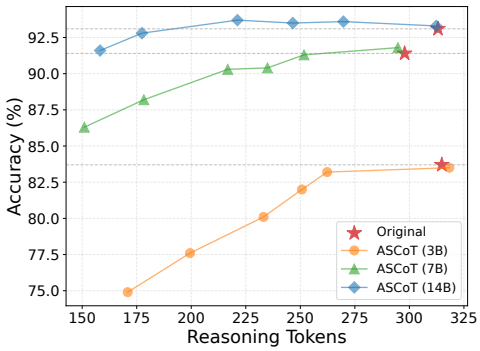

Figure 4: Performance of ASCoT on the Qwen2.5-Instruct Series. The results for the 3B, 7B, and 14B models are shown under various compression ratios and compared against the original, uncompressed baseline.

Comparison with Baselines Existing methods struggle to balance compression controllability with reasoning fidelity. Prompting fails to strictly adhere to token budgets. Truncation also incurs catastrophic performance degradation when it enforces strict length constraints.

In contrast, ASCoT demonstrates precise controllability and robust performance. It aligns closely with target ratios (e.g., ActRatio 0.79 at a 0.8 target) while preserving reasoning accuracy. Notably, at a full budget ($\gamma = 1.0$), ASCoT marginally outperforms the original baseline (+0.7% on GSM8K), suggesting that our verification mechanism effectively filters noise. Even under aggressive compression ($\gamma = 0.5$), ASCoT maintains a respectable 79.5% accuracy on GSM8K, significantly outperforming truncation based methods.

Scalability Analysis Table 2 details the performance across the Qwen2.5-Instruct series (3B, 7B, 14B) to evaluate scalability. ASCoT achieves substantial token reduction with minimal accuracy loss across all scales. For the 3B model, a 0.9 compression ratio reduces token usage by approximately 16% with only a 0.5 point accuracy drop. This resilience becomes more pronounced as model size increases. The 14B model exhibits exceptional robustness. Even when halving the token budget, it retains 91.6% accuracy (a mere 1.5% drop). This trend suggests a positive correlation between model scale and reasoning redundancy, larger models tend to generate more steps that ASCoT effectively identifies and prunes without compromising the final answer, demonstrating its potential to significantly lower inference costs for large-scale deployments without sacrificing reliability.

Figure 4 illustrates ASCoT's robust scalability across the Qwen2.5-Instruct series. While the 3B model incurs a negligible 0.5-point accuracy drop at 17% compression, this resilience amplifies with model size. The 7B variant maintains over 86% accuracy even with a 50% token reduction. Most notably, the 14B model demonstrates exceptional efficiency retention: halving the token budget results in a mere 1.5% accuracy decline. This trend indicates a positive correlation between model capacity

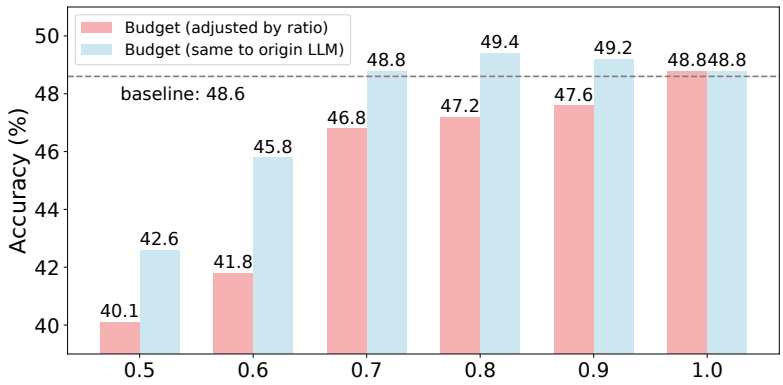

Figure 5: Performance comparison of ASCoT with varying maximum length constraints. We adjust the maximum length budget when evaluating our method on MATH-500.

and reasoning redundancy, suggesting that ASCoT effectively exploits the richer semantic structures inherent in larger models to maximize pruning without information loss. This confirm ASCoT's capability to balance computational efficiency and reasoning fidelity across varying model scales.

5.3 Error Propagation of Controlled Injection

To interrogate the cascading failure hypothesis, we conducted a controlled error-injection study on GSM8K using Qwen2.5-7B. We injected two classes of perturbations mimicking naturally occurring mistakes: Numeric (value deviation) and Symbolic (operator inversion).

Results in Table 4 reveal a Late-Stage Fragility. Contrary to the expectation that early errors propagate most severely, we observe that impact escalates with position. For numeric errors, a miscalculation at the final step (4/4) causes a catastrophic 51.69% accuracy drop, significantly outpacing early-stage errors (14.64% at Step 2). Symbolic errors amplify this trend, with late-stage injections (Step 3/3) causing the most profound degradation (56.78%). These findings suggest that models become increasingly committed to their reasoning trajectory, losing the capacity to self-correct as they approach the solution. Distribution details of the inference steps are provided in Table 3.

5.4 Analysis

Length Budget Analysis To further investigate ASCoT's efficiency, we analyze its performance under a fixed token budget. As illustrated in Figure 5, we assess our approach under two distinct length-budget settings. In the first, the maximum generation length is scaled by the compression ratio (ranging from 0.5 to 1.0). In the second, we retain the original LLM budget (*e.g.*, 512 tokens in GSM8K and 1024 tokens in MATH) regardless of compression. The results reveal that under the unchanged budget, our method consistently surpasses the baseline: accuracy rises from 42.6% to a peak of 49.4%. Notably, at compression ratios of 0.8 and 0.9, we observe absolute accuracy gains of 0.8 and 0.6 percentage points over the baseline in a smaller tokens, underscoring the potential of compressed CoT reasoning within the same budget.

The Late-Stage Fragility of CoT Our findings refine the traditional cascading failure model

Table 2: Experimental results of the Qwen2.5-Instruct series evaluated on GSM8K. We report Accuracy (%), Tokens and ActRatio.

Scale	Methods	Ratio	Accuracy	Tokens	ActRatio
3B	Original	–	83.7(0.0↓)	314.87	–
	ASCoT	1.0	83.5(0.2↓)	318.32	1.01
		0.9	83.2(0.5↓)	262.31	0.83
		0.8	82.0(1.7↓)	250.75	0.79
		0.7	80.1(3.6↓)	233.15	0.74
		0.6	77.6(6.1↓)	199.44	0.63
		0.5	74.9(8.8↓)	170.87	0.54
7B	Original	–	91.4(0.0↓)	297.83	–
	ASCoT	1.0	91.8(0.4↑)	294.74	0.99
		0.9	91.3(0.1↓)	251.71	0.85
		0.8	90.4(1.0↓)	234.95	0.79
		0.7	90.3(1.1↓)	216.73	0.73
		0.6	88.2(3.2↓)	178.28	0.60
		0.5	86.3(5.1↓)	150.98	0.51
14B	Original	–	93.1(0.0↓)	313.11	–
	ASCoT	1.0	93.3(0.2↑)	312.30	0.99
		0.9	93.6(0.5↑)	269.76	0.86
		0.8	93.5(0.4↑)	246.57	0.79
		0.7	93.7(0.6↑)	221.17	0.71
		0.6	92.8(0.3↓)	177.40	0.57
		0.5	91.6(1.5↓)	158.21	0.51

Table 3: Accuracy for Different Inference Steps in the GSM8K Dataset, using Qwen2.5-Instruct 7B.

Steps	Correct	Total	Accuracy
0	615	660	93.18
1	1	1	100.00
2	777	786	98.85
3	2044	2125	96.19
4	1936	2022	95.75
5	1110	1180	94.07
6	440	484	90.91
7	117	137	85.40
8	39	58	67.24
9–11	17	20	85.00

Table 4: Error-injection results on GSM8K using Qwen2.5-Instruct 7B. "All/Err" denotes the total steps vs. injected step. "Drop" denotes the accuracy drop.

All/Err	Type	Ori Acc	Fin Acc	Drop
4/2	numeric	95.75	81.11	14.64
4/3		95.75	66.19	29.56
4/4		95.75	44.06	51.69
2/2	symbolic	98.85	54.88	43.97
3/2		96.19	78.28	17.91
3/3		96.19	39.41	56.78

by uncovering Late-Stage Fragility phenomenon. While early-stage errors often trigger latent self-correction mechanisms due to high contextual entropy, identical errors introduced in the final steps are significantly more prone to persist uncorrected (qualitative illustration provided in Appendix C). This suggests that the model forms a rigid semantic commitment as reasoning progresses. Consequently, while cascading failures undeniably exist, Late-Stage Fragility constitutes a distinct, critical failure mode where the model lacks the flexibility to revisit and rectify terminal errors.

Case Study To provide a clear and detailed qualitative understanding of our method, we present a case study. Figure 3 illustrates a qualitative analysis where a standard model correctly structures the reasoning but commits a critical arithmetic error by calculating 125×8 as 1005. This flaw propagates to an incorrect conclusion. ASCoT effectively addresses this issue as the AVM module flags the step due to low Logical Validity while acknowledging its structural soundness. Crucially, this granular diagnosis prevents the unnecessary discarding of the entire reasoning chain. Instead, the MSCE executes a targeted dual-path repair to synthesize a corrected candidate, rectifying the calculation to 1000, thereby enabling the model to recover and deduce the correct total of 1015.

5.5 Ablation Study

To rigorously evaluate the contribution of each component within ASCoT, we conducted a multi-level ablation study on the GSM8K dataset. We first validated the necessity of the core modules: IRM and AVM. Subsequently, we performed a fine-grained analysis of the MSCE module to disentangle the efficacy of intrinsic versus extrinsic correction strategies. Our results detailed in Appendix F, confirm that the synergistic integration of position-aware verification and dual-path correction is essential for achieving state-of-the-art performance while minimizing token usage.

6 Conclusion

In this work, we critically re-examine the dynamics of error propagation in CoT reasoning, identifying a counter-intuitive phenomenon termed Late-Stage Fragility. Our systematic analysis reveals that contrary to the prevailing cascading failure assumption, errors introduced in the terminal stages of reasoning impose a disproportionately higher penalty on the final outcome. To address this vulnerability, we introduce ASCoT (Adaptive Self-Correction Chain-of-Thought). By integrating an Intelligent Routing Mechanism with a position-aware Adaptive Verification Manager, ASCoT strategically reallocates computational resources to verify high risk, late-stage steps. Extensive experiments on GSM8K and MATH benchmarks demonstrate that ASCoT achieves a superior trade-off between reasoning fidelity and computational efficiency, outperforming strong baselines including Self-Consistency. Ultimately, our findings advocate for a paradigm shift from uniform verification to adaptive, context-sensitive strategies, establishing a new foundation for reliable and resource-efficient LLM reasoning.

Limitations

While ASCoT effectively mitigates Late-Stage Fragility, its performance naturally correlates with the base model's verification capabilities. Additionally, our current study primarily targets mathematical reasoning focusing on numeric and symbolic errors. Extending the framework to broader domains ($e.g.$, code generation) and diverse modalities remains a promising avenue for future exploration.

References

Qiguang Chen, Libo Qin, Jinhao Liu, Dengyun Peng, Jiannan Guan, Peng Wang, Mengkang Hu, Yuhang Zhou, Te Gao, and Wanxiang Che. 2025. Towards reasoning era: A survey of long chain-of-thought for reasoning large language models. *arXiv preprint arXiv:2503.09567*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pages 4171–4186.

Yifu Ding, Wentao Jiang, Shunyu Liu, Yongcheng Jing, Jinyang Guo, Yingjie Wang, Jing Zhang, Zengmao Wang, Ziwei Liu, Bo Du, and 1 others. 2025. Dynamic parallel tree search for efficient llm reasoning. *arXiv preprint arXiv:2502.16235*.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, and 1 others. 2024. The llama 3 herd of models. *arXiv e-prints*, pages arXiv–2407.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Hengdao Guo, Ciyan Zheng, Herbert Ho-Ching Iu, and Tyrone Fernando. 2017. A critical review of cascading failure analysis and modeling of power system. *Renewable and Sustainable Energy Reviews*, 80:9–22.

Tingxu Han, Zhenting Wang, Chunrong Fang, Shiyu Zhao, Shiqing Ma, and Zhenyu Chen. 2024. Token-budget-aware llm reasoning. *arXiv preprint arXiv:2412.18547*.

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

Bairu Hou, Yang Zhang, Jiabao Ji, Yujian Liu, Kaizhi Qian, Jacob Andreas, and Shiyu Chang. 2025. Thinkprune: Pruning long chain-of-thought of llms via reinforcement learning, 2025. *URL https://arxiv.org/abs/2504.01296*.

Le Hou, Richard Yuanzhe Pang, Tianyi Zhou, Yuexin Wu, Xinying Song, Xiaodan Song, and Denny Zhou. 2022. Token dropping for efficient bert pretraining. *arXiv preprint arXiv:2203.13240*.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, and 1 others. 2022. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, and 1 others. 2024. Openai o1 system card. *arXiv preprint arXiv:2412.16720*.

Youhe Jiang, Fangcheng Fu, Wanru Zhao, Stephan Rabanser, Nicholas D Lane, and Binhang Yuan. 2025. Cascadia: A cascade serving system for large language models. *arXiv preprint arXiv:2506.04203*.

Ryo Kamoi, Yusen Zhang, Nan Zhang, Jiawei Han, and Rui Zhang. 2024. When can llms actually correct their own mistakes? a critical survey of self-correction of llms. *Transactions of the Association for Computational Linguistics*, 12:1417–1440.

Zixuan Ke, Fangkai Jiao, Yifei Ming, Xuan-Phi Nguyen, Austin Xu, Do Xuan Long, Minzhi Li, Chengwei Qin, Peifeng Wang, Silvio Savarese, and 1 others. 2025. A survey of frontiers in llm reasoning: Inference scaling, learning to reason, and agentic systems. *arXiv preprint arXiv:2504.09037*.

Minsu Kim, Jean-Pierre Falet, Oliver E Richardson, Xiaoyin Chen, Moksh Jain, Sungjin Ahn, Sungsoo Ahn, and Yoshua Bengio. 2025. Search-based correction of reasoning chains for language models. *arXiv preprint arXiv:2505.11824*.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213.

Abhinav Kumar, Jaechul Roh, Ali Naseh, Marzena Karpinska, Mohit Iyyer, Amir Houmansadr, and Eugene Bagdasarian. 2025. Overthink: Slowdown attacks on reasoning llms. *arXiv preprint arXiv:2502.02542*.

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

Guangliang Liu, Haitao Mao, Jiliang Tang, and Kristen Marie Johnson. 2024a. Intrinsic self-correction for enhanced morality: An analysis of internal mechanisms and the superficial hypothesis. *arXiv preprint arXiv:2407.15286*.

Tengxiao Liu, Qipeng Guo, Xiangkun Hu, Cheng Jiayang, Yue Zhang, Xipeng Qiu, and Zheng Zhang. 2024b. Can language models learn to skip steps? *Advances in Neural Information Processing Systems*, 37:45359–45385.

Yue Liu, Jiaying Wu, Yufei He, Hongcheng Gao, Hongyu Chen, Baolong Bi, Ruihan Gong, Jiaheng Zhang, Zhiqi Huang, and Bryan Hooi. 2025. Efficient inference for large reasoning models: A survey. *arXiv preprint arXiv:2503.23077*.

Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

Yiran Ma, Zui Chen, Tianqiao Liu, Mi Tian, Zhuo Liu, Zitao Liu, and Weiqi Luo. 2025. What are step-level reward models rewarding? counterintuitive findings from mcts-boosted mathematical reasoning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 24812–24820.

Zhuoshi Pan, Qianhui Wu, Huiqiang Jiang, Menglin Xia, Xufang Luo, Jue Zhang, Qingwei Lin, Victor Rühle, Yuqing Yang, Chin-Yew Lin, and 1 others. 2024. Llmlingua-2: Data distillation for efficient and faithful task-agnostic prompt compression. *arXiv preprint arXiv:2403.12968*.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652.

Qwen Team. 2024. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*.

Gladys Tyen, Hassan Mansoor, Victor Cărbune, Peter Chen, and Tony Mak. 2023. Llms cannot find reasoning errors, but can correct them given the error location. *arXiv preprint arXiv:2311.08516*.

Somin Wadhwa, Silvio Amir, and Byron C Wallace. 2024. Investigating mysteries of cot-augmented distillation. *arXiv preprint arXiv:2406.14511*.

Xin Wang, Yu Zheng, Zhongwei Wan, and Mi Zhang. 2024. Svd-llm: Truncation-aware singular value decomposition for large language model compression. *arXiv preprint arXiv:2403.07378*.

David Wingate, Mohammad Shoeybi, and Taylor Sorensen. 2022. Prompt compression and contrastive conditioning for controllability and toxicity reduction in language models. *arXiv preprint arXiv:2210.03162*.

Heming Xia, Chak Tou Leong, Wenjie Wang, Yongqi Li, and Wenjie Li. 2025. Tokenskip: Controllable chain-of-thought compression in llms. *arXiv preprint arXiv:2502.12067*.

Silei Xu, Wenhao Xie, Lingxiao Zhao, and Pengcheng He. 2025. Chain of draft: Thinking faster by writing less. *arXiv preprint arXiv:2502.18600*.

Ning Xuefei, Lin Zinan, Zhou Zixuan, and 1 others. 2023. Skeleton-ofthought: Prompting llms for efficient parallel generation.

Ning Yang, Hai Lin, Yibo Liu, Baoliang Tian, Guoqing Liu, and Haijun Zhang. 2025. Token-importance guided direct preference optimization. *arXiv preprint arXiv:2505.19653*.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyan Luo, Zhangchi Feng, and Yongqiang Ma. 2024. Llamafactory: Unified efficient fine-tuning of 100+ language models. *arXiv preprint arXiv:2403.13372*.

Peijun Zhu, Ning Yang, Jiayu Wei, Jinghang Wu, and Haijun Zhang. 2025. Breaking the moe llm trilemma: Dynamic expert clustering with structured compression. *arXiv preprint arXiv:2510.02345*.

A Appendix

This document provides supplementary material, including pseudocode for the ASCoT mechanism, experimental setup, error-injection experiments, and results analysis. It also covers model configurations, hyperparameters used in the study. Additionally, the document discusses the model's Late-Stage Fragility and outlines current limitations and future directions.

B Algorithmic Details

Algorithm 1 formally outlines the inference pipeline of the proposed ASCoT framework. The process initiates with the Intelligent Routing Mechanism (IRM), which semantically prunes the initial chain $\mathcal{C}_{\text{init}}$ to filter redundant tokens, yielding a compressed sequence $\mathcal{C}_{\text{pruned}}$.

Subsequently, the system iterates through each reasoning step t_k. At each juncture, the Adaptive Verification Manager (AVM) assesses the step against the accumulated context S_{ctx}, computing a risk score based on both content quality and positional impact. If this score exceeds the predefined threshold τ, the step is flagged as fragile. In such cases, the Multi-Perspective Self-Correction Engine (MSCE) is triggered to generate a rectified step t_k^{corr} via dual-path correction. Conversely, low-risk steps are retained directly. This selective intervention ensures that computational resources are reallocated specifically to fix late-stage vulnerabilities while maintaining overall efficiency.

C Qualitative Analysis of Late-Stage Fragility

To visually demonstrate the Late-Stage Fragility phenomenon discussed in Section 4.3, we provide

Algorithm 1 ASCoT: Adaptive Self-Correction Chain-of-Thought

Input: Initial CoT $\mathcal{C}_{\text{init}}$; Compression ratio γ; Risk threshold τ
Output: Corrected CoT $\mathcal{C}_{\text{final}}$

// Stage 1: Semantic Pruning via IRM
$\mathcal{C}_{\text{pruned}} \leftarrow \text{SemanticPruning}(\mathcal{C}_{\text{init}}, \gamma)$
// Stage 2 & 3: Adaptive Verification and Correction
$\mathcal{C}_{\text{final}} \leftarrow \emptyset$; $S_{\text{ctx}} \leftarrow \emptyset$
for each step t_k **in** $\mathcal{C}_{\text{pruned}}$ **do**
  // Calculate risk via AVM (Logic, Fact, Clarity, Position)
  $R_k \leftarrow \text{ComputeRiskScore}(t_k, S_{\text{ctx}})$
  if $R_k > \tau$ **then**
    // High risk detected: Trigger MSCE correction
    $t_k^{\text{corr}} \leftarrow \text{DualPathCorrection}(t_k, S_{\text{ctx}})$
    $\text{Append}(t_k^{\text{corr}}, \mathcal{C}_{\text{final}})$ ▷ *Use corrected step*
    $S_{\text{ctx}} \leftarrow \text{UpdateContext}(S_{\text{ctx}}, t_k^{\text{corr}})$
  else
    $\text{Append}(t_k, \mathcal{C}_{\text{final}})$ ▷ *Keep original step*
    $S_{\text{ctx}} \leftarrow \text{UpdateContext}(S_{\text{ctx}}, t_k)$
  end if
end for
// Final Output Generation
return $\mathcal{C}_{\text{final}}$

a comparative case study in Figure 6. The example contrasts the model's robust recovery from early-stage perturbations against its failure to correct identical errors in the final reasoning steps.

D Experimental Setup

To clearly elucidate the algorithmic pipeline of the ASCoT methodology, we provide its pseudocode. The algorithm commences upon receiving a problem query, which is first processed by the target Large Language Model (LLM) to generate an initial CoT solution, denoted as $CoT_{initial}$. This initial output is subsequently passed to the IRM module for token pruning based on a predefined compression ratio γ, thereby enhancing efficiency. The compressed CoT then undergoes a step-wise risk assessment by the AVM module. For each step t_k, the AVM computes a Risk Score $R(t_k)$, which integrates a multi-dimensional quality assessment $Q(t_k)$ with a Positional Impact score $I(k)$. If this score exceeds a predefined threshold τ, the step is flagged as high risk and is routed to the MSCE module for remediation. The MSCE employs a dual-path strategy, encompassing intrinsic reflection and extrinsic independent generation to correct the identified error, yielding the final, refined CoT_{final}.

Model Configuration and Hyperparameters All experiments were conducted on an Ubuntu 20.04 LTS server equipped with four NVIDIA GeForce RTX 3090 GPUs (24GB each), using PyTorch 1.13.0. We fine-tuned each base LLM via Low-Rank Adaptation (LoRA) (Hu et al., 2022) with rank $r = 8$ and scale factor $\alpha = 16$. We utilize LLMLingua-2 (Pan et al., 2024) as the token importance metric to generate our compressed CoT training data. IRM is evaluated over discrete compression rates in the range $\{0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$. The $I(k) = w_a \cdot e^{\alpha(k-1)}$ was calibrated by fitting to data from controlled error-injection trials. All fine-tuning used best-practice hyperparameters. We train the models for 5 epochs on both datasets. The peak learning rate is set to $5e-5$, following a cosine decay schedule. We use AdamW (Loshchilov and Hutter, 2017) for optimization, with a warmup ratio of 0.1. We implement our training process using the LLaMA-Factory (Zheng et al., 2024) library. During inference, the maximum number of tokens set to 512 for GSM8K and 1024 for MATH.

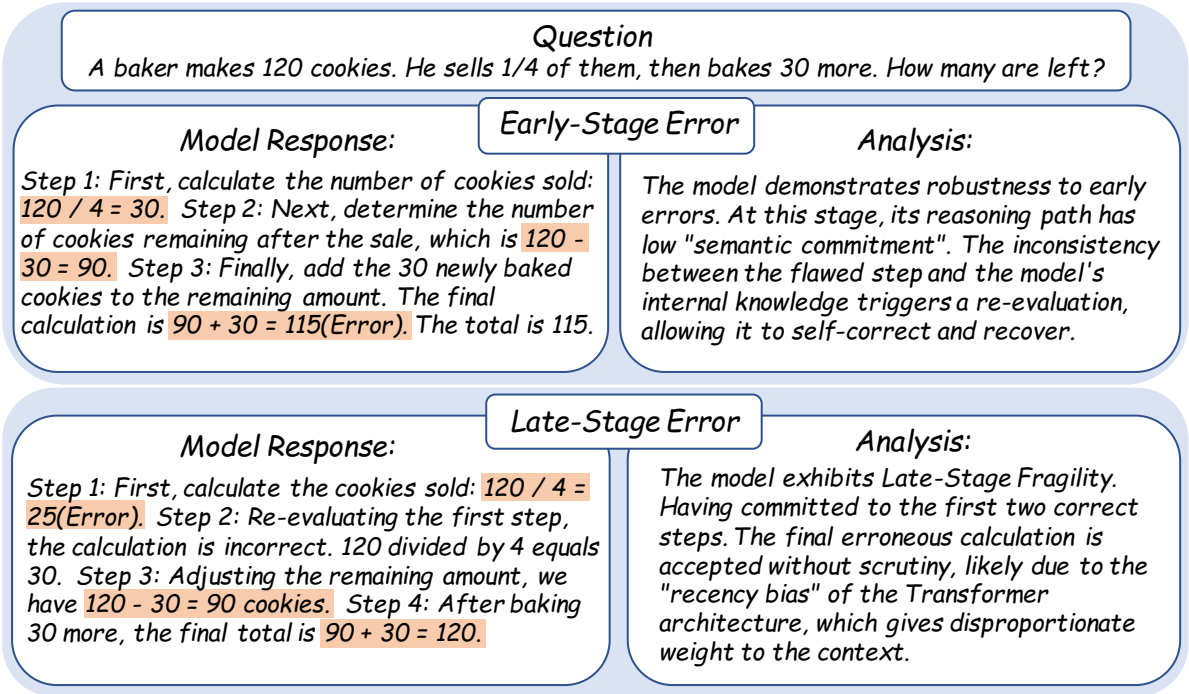

Figure 6: An illustration of Late-Stage Fragility. The model recovers from an early-stage calculation error, but fails when an identical type of error is introduced in the final step. The tokens that occur related errors are highlighted in orange.

E Error-Injection Experiments

E.1 Error Simulation

To probe the model's resilience in a realistic manner, our error-injection methodology does not introduce arbitrary flaws. Instead, it is designed to simulate the spontaneous reasoning failures we observed in the model's own unguided generations, thereby ensuring the ecological validity of our experiment. We focused on two distinct classes of these naturally occurring failure patterns.

The first, Numeric Errors, simulates arithmetic miscalculations. The injection process for this error type involves identifying a numerical value in a correct reasoning step and slightly altering it, for instance, by adding or subtracting a small, fixed constant value to mimic a common calculation mistake. The second, Symbolic Errors, simulates logical inconsistencies. This is achieved by identifying a primary arithmetic operator within a step and systematically replacing it with its mathematical inverse ($e.g.$, + is replaced by -, and * by ÷). To maintain procedural integrity, we preemptively implemented constraints to avoid invalid operations, such as division by zero.

Our overall experimental procedure involved systematically injecting one such controlled error at a specific position k within a baseline of validated, correct reasoning chains. Following this injection, the model was prompted to resume the reasoning process from that point, and the accuracy of the final answer was recorded to precisely measure the positional impact of the error.

Infer Steps	Counts	Accuracy
2	786	98.85
3	2125	96.19
4	2022	95.75
5	1180	94.07
6	484	90.91
7	137	85.40
8	58	67.24
9	7	57.14

Table 5: Performance analysis by reasoning step count on the GSM8K dataset, illustrating both the frequency of problems per step count and the corresponding model accuracy.

E.2 Distribution of Results

Table 5 illustrates the proportion of total steps across all problems. We can conclude from the chart that a significant majority of problems are concentrated in the 2-5 step range, with 3-step and

4-step problems being the most prevalent.

When injecting a numerical error, the accuracy drops are relatively uniform across positions: at step 2 yields a drop of 59.92%, at step 3 a drop of 60.47%, and at step 4 a drop of 61.70%a miscalculation. In contrast, symbolic errors induce more severe degradation, and this severity escalates with later injection points. Replacement of an operator in the first error step (2/2) reduces the accuracy by 69.67%, while an error in the final step (4/4) inflicts a reduction of 80.80%, with symbolic faults becoming increasingly catastrophic when introduced deeper into the chain.

Analysis of Late-Stage Fragility Early-stage errors often activate the model's latent self-correction capability, wherein subsequent reasoning steps can reinterpret and compensate for previous mistakes, leading to a correct final answer. This is facilitated by the model maintaining higher semantic entropy and contextual flexibility during the initial stages of reasoning. However, as the reasoning chain progresses, the model increasingly commits to a specific semantic trajectory, gradually narrowing its generative space and becoming less sensitive to alternative interpretations. We refer to this phenomenon as semantic commitment, a state in which the model's outputs become rigidly anchored to previously generated content. At this point, even minor numerical or symbolic errors near the conclusion are less likely to be detected or corrected, resulting in high failure rates.

Our findings do not aim to refute the cascading failure hypothesis entirely, but rather to refine and contextualize it. Previous work suggests that early-stage errors in CoT reasoning tend to propagate forward, compromising subsequent steps and the final answer. However, our analysis reveals that the impact of reasoning errors is not uniform. We also find that late-stage reasoning steps are disproportionately sensitive to localized perturbations. Errors introduced near the conclusion of reasoning are much less likely to be corrected. In contrast, early-stage errors may still trigger the model's latent self-correction mechanisms. This indicates that cascading failures do occur, but that Late-Stage Fragility constitutes an additional failure mode not previously emphasized.

F Detailed Ablation Studies

In this section, we present a series of ablation experiments to validate the design choices of ASCoT.

Table 6: Ablation of core components. "w/o IRM" denotes removing the pruning stage; w/o Correction denotes using only pruning without any verification or correction.

Configuration	Acc. (%)	Tokens	ActRatio
ASCoT (Full)	**86.9**	**214.24**	**1.00**
w/o IRM	87.1	325.40	1.52
w/o Correction	82.5	198.12	0.92

Table 7: Ablation of the AVM strategy. We compare our position-aware scoring against a uniform weighting scheme.

AVM Strategy	Acc. (%)	Δ Acc.
Position-Aware (Ours)	**86.9**	–
Uniform Weights	84.3	-2.6

All experiments were conducted using the LLaMA-3.1-8B model on the GSM8K benchmark.

F.1 Impact of Core Components

We first investigate the necessity of the two fundamental stages: semantic pruning (IRM) and the correction pipeline (AVM+MSCE). Table 6 summarizes the results. As shown in Table 6, removing the IRM leads to a massive increase in token usage (+52%) with negligible accuracy gain (+0.2%), validating IRM's critical role in efficiency. Conversely, removing the correction pipeline (w/o Correction) causes accuracy to drop significantly to 82.5%, underscoring that semantic pruning alone induces fragility that must be compensated for by our robust verification mechanism.

F.2 Efficacy of Position-Aware Verification

A key innovation of ASCoT is the integration of the Positional Impact Score within the AVM module to address Late-Stage Fragility. Table 7 compares our strategy against a uniform risk assessment. The results in Table 7 reveal a 2.6% performance degradation when position awareness is removed. This empirical evidence supports our hypothesis that errors in later reasoning steps are more detrimental. By assigning higher risk weights to these terminal steps, ASCoT effectively reallocates verification resources to where they are most needed.

F.3 Analysis of Dual-Path Correction

Finally, we dissect the Multi-Perspective Self-Correction Engine (MSCE) to evaluate the contribution of its Dual-Path (Intrinsic + Extrinsic)

User: Natalia sold clips to 48 of her friends in April, and then she sold half as many clips in May. How many clips did Natalia sell altogether in April and May?
Assistant:
"You are a helpful assistant. Please reason step by step, and put your final answer within \\boxed{}. Natalia sold clips to 48 of her friends in April, and then she sold half as many clips in May. How many clips did Natalia sell altogether in April and May?",
Model output:
Let's break down the problem step by step.
Step1: **Determine the number of clips sold in April:** Natalia sold 48 clips in April.
Step2: **Calculate the number of clips sold in May:** According to the problem, Natalia sold half as many clips in May as she did in April. - Therefore, the number of clips sold in May is 24.
Step3: **Find the total number of clips sold in both months:** Add the number of clips sold in April and May: 48 + 24 = 72. Thus, the total number of clips Natalia sold in April and May is boxed\\{72}.
The final answer is: "$\\boxed{72}$", "prediction": "72", "cot_length": 174, "accuracy": true

Figure 7: Example generations by ASCoT, using Qwen2.5-Instruct 7B.

Table 8: Ablation of the MSCE correction strategy. "Intrinsic" relies on self-reflection; "Extrinsic" uses independent regeneration.

Correction Strategy	Acc. (%)	Δ Acc.
Dual-Path (Ours)	**86.9**	–
Extrinsic Only	86.1	-0.8
Intrinsic Only	85.0	-1.9

strategy compared to single-source baselines. Table 8 demonstrates that the Dual-Path strategy outperforms individual approaches. Relying solely on Intrinsic correction yields the lowest accuracy (85.0%), likely because the model struggles to escape its own erroneous reasoning context. Extrinsic correction performs better (86.1%) but is surpassed by the synergistic combination of both, which leverages the strengths of context-aware reflection and objective regeneration.

G Generation Examples

To provide a qualitative illustration of our method's end-to-end functionality, we present representative examples of reasoning chains after processing by the ASCoT pipeline. Figure 7 displays the corrected output after being processed by ASCoT. This transformation from a flawed to a correct reasoning chain highlights the core capabilities of our system. It demonstrates the AVM module successfully identifying a high risk step by analyzing its content quality and positional impact, followed by the MSCE module executing a targeted repair. This example confirms ASCoT's ability to structure reasoning but, more importantly, to dynamically detect and rectify internal inconsistencies, ultimately leading to a more trustworthy final answer.

H Future Work

We propose several promising future directions. A key avenue is the augmentation of MSCE module through integration with external, deterministic oracles like calculators, code interpreters, or knowledge-base APIs. By querying these verifiable tools, the correction engine could access ground-truth feedback, significantly enhancing its reliability and moving beyond the constraints of the model's internal knowledge. Another important direction involves developing online adaptive learning mechanisms. Such a system would enable ASCoT to continuously self-supervise and refine its performance in real time across various domains, evolving from a static framework into a more dynamic and robust reasoning system.