ONCE-MORE: CONTINUOUS SELF-CORRECTION FOR LARGE LANGUAGE MODELS VIA PERPLEXITY-GUIDED INTERVENTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Large Language Models (LLMs) often experience compounding errors during long text generation. Early errors could propagate and lead to drift, faulty reasoning, or repetition. Self-correction is a promising technique for addressing this issue. However, the existing main approaches have limitations. Supervised training methods can build self-correcting behaviours into models, but require training data collection and lack cross-domain generalizability. Current post-hoc iterative refinement methods operate only at inference time, but have to wait for substantial portions of the draft to be generated before providing feedback. Such feedback can not guarantee effective guidance, and the same error patterns can reappear. In this paper, we propose Once-More, a model-agnostic post-hoc self-correction framework that intervenes during generation. Once-More leverages token-level perplexity and verifier feedback to provide continuous, guided steering of the generation path through a logit suppression mechanism, and therefore helps accumulate "more correct" steps throughout the generation process. Evaluation on multiple benchmarks demonstrates that the proposed Once-More achieves state-of-the-art results when compared to representative self-correction methods. To our knowledge, Once-More is the first post-hoc method to leverage token perplexity and external feedback for continuous, guided self-correction.

1 INTRODUCTION

Large Language Models (LLMs) have demonstrated remarkable capabilities in text generation and complex reasoning tasks (Sun et al., 2025; Ferrag et al., 2025). However, their autoregressive nature poses a fundamental challenge: during generation, LLMs can produce errors or inaccuracies that further propagate through subsequent tokens, making errors to compound and eventually drive the generation away from the target (Arbuzov et al., 2025; Wang et al., 2023). This error-compounding issue raises serious concern when deploying LLMs in critical decision-making processes. Many LLM-based agentic workflows suffer from reliability issues stemming from this concern, which limits their viability for real-world deployment. (Pan et al., 2025; Gabison & Xian, 2025).

To address such concerns, researchers have developed various self-correction methods which enable LLMs to modify or revise their outputs at inference (Kamoi et al., 2024; Pan et al., 2024). Such methods identify errors during generation and guide models toward better responses through targeted feedback mechanisms (Paul et al., 2023; Tyen et al., 2023; Shinn et al., 2023). The feedback on the final output is provided via external knowledge bases (Gou et al., 2024; Zheng et al., 2024) or by more capable models (e.g., GPT-4) (Koutcheme et al., 2024). However final-output feedback is too coarse to provide effective steering, and thus these methods often fail to prevent recurring error patterns and lead to non-convergent refinement loops (Kamoi et al., 2024; Xu et al., 2024). Some methods use supervised fine-tuning as an alternative to build self-correction behaviour directly into models (Yan et al., 2025; Liu et al., 2025; Wang et al., 2024a); however, they are inherently limited by the distribution of their training data. They are susceptible to performance degradation on out-of-distribution tasks, where uncorrected errors can cascade through the generation.

Rather than building domain-specific models with built-in self-correction, we believe that iterative refinement remains the most general and practical approach. However, it requires fundamental

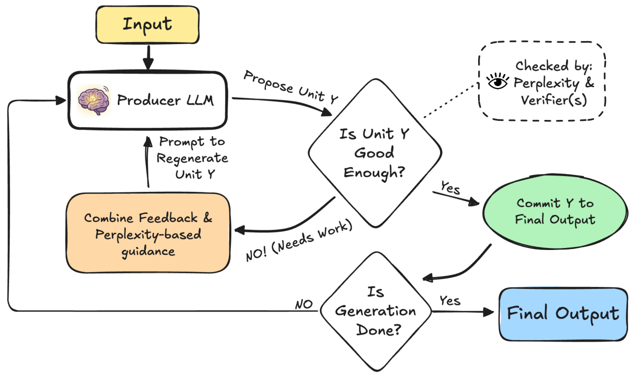


Figure 1: Overview of the proposed Once-More framework (Sec. 3.1). The Producer generates adaptive units, which are checked by Perplexity and Verifier(s). Rejected units trigger regeneration via the combined feedback and perplexity-driven guidance, creating a continuous self-correction process that intervenes before errors propagate.

improvements in guidance granularity. Effective self-correction should function continuously throughout generation, progressively steering models along better trajectories, and eventually allow "more correct" incremental decisions to accumulate into better final outputs. Motivated by this vision, in this paper, we propose an inference-time framework, **Once-More**, which performs continuous self-correction on any LLM generation process. It transforms the LLM generation process into a multi-agent interaction process: a Producer generates content while Verifier(s) evaluate and provide continuous guidance. Once-More operates at the level of units of generation (e.g., clauses, sentences, paragraphs, or code blocks). A complete generation is a sequential composition of unit generations. When the Producer generates a unit, the framework computes its perplexity to assess potential errors. High perplexity triggers verification for goal alignment, correctness, and consistency. Rejected units are regenerated using evaluation feedback combined with perplexity-driven logit redistribution until the Verifier(s) accept the unit. Figure 1 shows the high-level overview of the proposed Once-More. The framework also supports LLM, non-LLM, or tool-augmented Verifiers.

The proposed Once-More are evaluated on various benchmarks, including Olympiad mathematics (AIME 2024 (Art of Problem Solving, 2024a;b) and 2025 (Art of Problem Solving, 2025a;b)), graduate-level science questions (GPQA) (Rein et al., 2024), LiveBench (reasoning) (White et al., 2024), SVAMP (Patel et al., 2021), and GSM8K (Cobbe et al., 2021). It achieves state-of-the-art performance when compared to other self-correction methods. Our contributions are as follows: (1) We propose the Once-More framework to perform continuous, fine-grained self-correction, which yields State-of-the-Art performance while being more token-efficient than previous iterative refinement methods; (2) We present a method to uniquely fuse perplexity-based uncertainty signals with external verifier feedback. It is employed in Once-More to prevent repeated errors while preserving model beliefs during generation. To our knowledge, the proposed Once-More is the first post-hoc method to leverage token perplexity and external feedback for continuous, guided self-correction; (3) Our proposed continuous guidance mechanism in Once-More enables LLMs to perform reasoning-like corrections dynamically during generation, which helps to detect early errors and move to better generation trajectories.

2 Related Work

Error accumulation in LLMs. The problem of error propagation in autoregressive models has been extensively studied. Classical analyses of exposure bias demonstrate that models trained on teacher-forced prefixes often struggle at test time, leading to error compounding in long generations (Arora et al., 2022; Schmidt, 2019). LLMs also suffer from the error accumulation issue, although LLM's error accumulation patterns are more nuanced than simple exponential decay (Arbuzov et al., 2025). While scaling up LLMs enables the emergence of self-correction behaviours (Liu et al., 2024; Wang et al., 2024b), they remain unpredictable and uncontrollable at inference. To address this concern, we propose making self-correction explicit and controllable through perplexity-guided intervention. Our proposed approach is model-agnostic and can be used with smaller-sized LLMs, making it more practical for local or edge use cases.

Self-correction via iterative refinement. Several approaches can enable LLMs to revise their outputs through iterative processes. For instance, CRITIC uses external tools to verify model outputs and provides corrective feedback (Gou et al., 2024), Self-Refine employs models as both generators and evaluators (Madaan et al., 2023), and Verify-and-Edit incorporates knowledge bases for factual correction (Zhao et al., 2023). While these approaches can improve response quality, they operate on completed drafts or coarse-grained steps, and therefore could allow errors to compound before intervention occurs. The prompt-only feedback also occasionally fails to prevent recurring error patterns. Our proposed Once-More addresses the above concerns by intervening at a more granular level with continuous monitoring, enabling more direct and localized feedback that has a more effective impact on the model's generation process.

Self-correction via supervised fine-tuning. Another line of work builds self-correction behaviours directly into the models through training. For instance, S^3c-MATH trains spontaneous self-correction for mathematical reasoning (Yan et al., 2025), LLaVA-SCo extends this to vision-language models (Liu et al., 2025), and the learning-from-failure approach finetunes models to internalize correction behaviours (Wang et al., 2024a). However, their gains remain modest, and these methods require extensive training data and are susceptible to performance degradation on out-of-distribution tasks. To tackle such issues, our proposed framework remains model-agnostic and training-free, while providing continuous guidance for self-correction behaviours.

Multi-agent and role-playing approaches. Formulating generation as a multi-agent interaction has been shown to be effective for complex tasks. E.g., Reflexion equips agents with verbal feedback and episodic memory to improve over trials (Shinn et al., 2023), MAgICoRe combines solver, reviewer, and refiner roles with reward models for targeted step-wise error correction (Chen et al., 2024), and ReAct interleaves reasoning with external actions to reduce hallucinations (Yao et al., 2023). These methods leverage role specialization to provide feedback in different perspectives, and inspired by their success, our proposed Once-More adopts this idea in its Producer-Verifier architecture. However, their coarse-grained feedback and prompt-only interactions cannot guarantee effective steering. We therefore plan to enhance the multi-agent system with continuous monitoring and direct probability intervention.

Decoding time steering. The approach that manipulates sampling at inference time offers another avenue for controlling generation behaviour. E.g., GeDi guides generation toward desired attributes with discriminative models (Krause et al., 2020), Contrastive Decoding reduces hallucination by subtracting weaker model logits (Li et al., 2022; O'Brien & Lewis, 2023), and DoLa sharpens factuality through layer-wise contrast (Chuang et al., 2023). Recent work on Cautious Next Token Prediction demonstrates that perplexity-based mechanisms can effectively guide generation decisions (Wang et al., 2025b), providing a theoretical basis for our approach. However, existing methods only operate globally or target high-level attributes, lacking fine-grained control. Inspired by these methods, our proposed method manipulates sampling at inference time, but also employs localized logit manipulation to perform a more effective generation steering.

3 ONCE-MORE FRAMEWORK

We now present Once-More, a model-agnostic framework that performs continuous self-correction during LLM generation. The key idea is simple: rather than waiting for a complete output before providing feedback, we monitor generation quality at intermediate steps and intervene immediately when potential errors are detected. This section describes the framework's architecture (Sec. 3.1), how it monitors generation quality (Sec. 3.2), and how it guides corrections (Sec. 3.3).

3.1 FRAMEWORK ARCHITECTURE

As shown in Figure 1, Once-More transforms standard LLM generation into a monitored multi-agent process with three core components:

Producer. A frozen, pre-trained LLM that generates content incrementally in *units* (adaptive chunks such as sentences, paragraphs, or code blocks). The Producer operates at inference time without

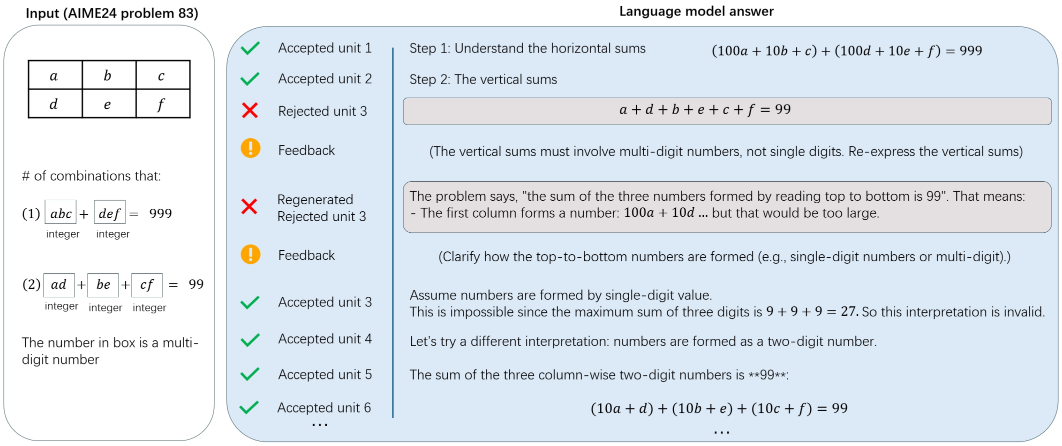


Figure 2: Illustration example: The proposed Once-More corrects a mathematical interpretation error on AIME 2024 Problem 83. It rejects an incorrect single-digit interpretation (unit 3), provides feedback, and guides the model to reconsider. The model then explicitly reasons why the initial is impossible and pivots to the correct two-digit interpretation, demonstrating intermediate reasoning.

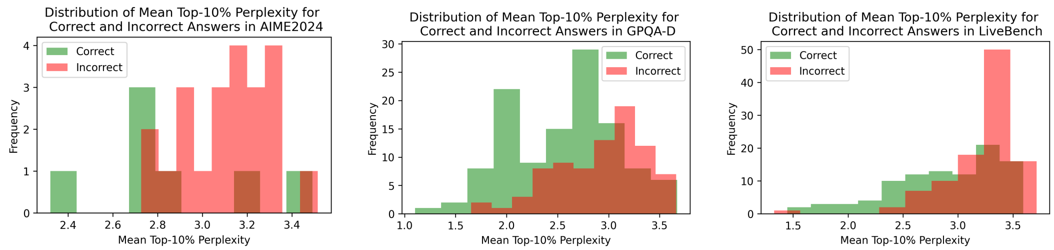
any training or fine-tuning. The only requirement is access to token-level probability scores during decoding, which many providers already expose through their APIs. This requirement matters only for our regeneration mechanism and doesn't constrain the choice of model architecture.

Verifier(s). One or more agents that evaluate each provisional *unit* against the task **Goal** and constraints, returning a binary decision (accept or reject) along with optional feedback F in natural language or structured format. Verifiers may be LLMs, non-LLM programs, or tool-augmented modules as in tool-using agents (Yao et al., 2023; Shen et al., 2023; Shen, 2024).

Generation Units. Rather than working with fixed token windows, Once-More adapts its intervention granularity to the task context. For mathematical reasoning, a unit might be a single equation or derivation step; for code generation, a function or block; for prose, a sentence or paragraph. Units are identified using syntactic markers (punctuation, indentation) or learned boundary predictors. This adaptivity enables intervention at the most appropriate granularity for each task.

The Generation-Verification Loop. The proposed Once-More framework operates through continuous generation, monitoring, and correction loops. Figure 2 demonstrates this on a mathematical problem, showing how the framework catches and corrects an incorrect interpretation mid-solution. Let **Goal** denote the task and **Context** the generation history that has been verified and accepted. The framework operates through the following loop:

1. **Generate:** The Producer generates a provisional unit $Y = [y_1, \ldots, y_n]$ conditioned on (Goal, Context), where y_i is one output token at time i.
2. **Monitor:** Compute the unit's perplexity $\mathrm{PPL}_{\text{unit}}(Y)$ as an uncertainty signal (refsubsec:perplexity).
3. **Decide:**
   - If $\mathrm{PPL}_{\text{unit}}(Y) \leq P_{\text{th}}$ (low uncertainty), trust the unit and append to Context.
   - If $\mathrm{PPL}_{\text{unit}}(Y) > P_{\text{th}}$ (high uncertainty), invoke Verifier for explicit checking.
4. **Verify & Correct:**
   - If accepted: Append to Context and create a checkpoint.
   - If rejected: Trigger guided regeneration (Sec. 3.3) using verifier feedback and probability adjustments.
   - If regeneration also fails: Roll back to the previous checkpoint and regenerate from there, as the error may have originated earlier.



(a) AIME24 (b) GPQA-Diamond (c) LiveBench(Reasoning)

Figure 3: Distributions of the mean top-10% perplexity for correct vs. incorrect answers across benchmarks. Incorrect units consistently exhibit higher perplexity.

This continuous monitoring enables early error detection and correction before mistakes compound into larger failures.

3.2 Monitoring Generation Quality via Perplexity

To detect potential errors during generation, we need a real-time quality signal that doesn't require external verification for every token. We leverage *perplexity*, which is a standard language modeling metric that measures prediction uncertainty (Jelinek et al., 1977; Christopher Manning, 2021; Hugging Face, 2025). Recent work shows that entropy-based metrics provide reliable signals for response quality (Fu et al., 2025; Yang et al., 2025b; Wang et al., 2025a), which motivates our approach.

Intuition. When a model confidently predicts the next token (high probability on one candidate), perplexity is low. When probability mass spreads across multiple candidates, perplexity increases, signalling uncertainty. High uncertainty often correlates with potential errors, indicating the model is likely unsure what to generate next, and it's probably on the wrong path.

Token-Level Perplexity. At each position t, the Producer outputs a probability distribution $q_t(v)$ over the vocabulary $\mathcal{V}$. Rather than computing perplexity over the entire vocabulary (expensive), we approximate using the top-K most likely tokens $S_t = \{v_{t,1}, \dots, v_{t,K}\}$ (Holtzman et al., 2018; Fan et al., 2018):

$$\mathrm{PPL}_t^{(K)} \;=\; \exp\!\left(\frac{1}{K}\sum_{i=1}^{K}(-\log q_t(v_{t,i}))\right) \tag{1}$$

This efficiently captures local uncertainty: $\mathrm{PPL}_t^{(K)} \approx 1$ when one token dominates, and increases as mass spreads across alternatives.

Unit-Level Aggregation & Verification Trigger. For a unit $Y = [y_1, \dots, y_n]$, we average token-level perplexities:

$$\mathrm{PPL}_{\text{unit}}(Y) \;=\; \frac{1}{n}\sum_{t=1}^{n}\mathrm{PPL}_t^{(K)} \tag{2}$$

Verification is triggered when $\mathrm{PPL}_{\text{unit}}(Y) > P_{\text{th}}$, where P_{th} is calibrated on a small held-out set to target a desired verification rate (e.g., check the top 25% most uncertain units). Details of the calibration procedure are in Appendix A.2. Figure 3 empirically validates this approach: on our benchmarks, incorrect units consistently show substantially higher perplexity than correct ones, demonstrating clear separation between reliable and problematic generations.

3.3 Guided Regeneration via Probability Adjustment

When a unit is rejected, simply asking the model to "try again" often produces the same mistake, since the model's learned biases can lead it down the same path (Xu et al., 2024). The model may acknowledge the feedback but still sample the same tokens due to strong learned priors. To break this cycle, Once-More performs *guided regeneration*: it (1) incorporates verifier feedback into the prompt, and (2) directly adjusts the token probability distribution to suppress previously chosen tokens and explore alternatives.

Overview of the Adjustment Mechanism. The adjustment process operates at the token level during regeneration:

1. **Identify high-uncertainty tokens:** Convert the rejected unit's token-level perplexities into position-wise suppression strengths; therefore, tokens with higher uncertainty receive stronger suppression.
2. **Align attempts:** Match tokens between the rejected attempt and the current regeneration attempt to determine which tokens to suppress at each position.
3. **Redistribute probability mass:** At each position, decrease the probability of the previously chosen token and proportionally increase alternatives, ensuring a valid probability distribution.

We now detail each step.

Step 1: Converting Perplexity to Suppression Strength. Let $[x_1, \ldots, x_{n_-}]$ denote the rejected unit with token-level perplexities $\{\mathrm{PPL}_i\}_{i=1}^{n_-}$. We normalize these into suppression weights:

$$\hat{u}_i^{(1)} \;=\; \frac{\mathrm{PPL}_i - \min(\mathrm{PPL})}{\max(\mathrm{PPL}) - \min(\mathrm{PPL}) + \varepsilon} \;\in [0,1] \tag{3}$$

where ε is a small constant for numerical stability. This maps perplexity to a [0,1] scale: tokens with high perplexity (high uncertainty) get suppression weights near 1, while confident tokens get weights near 0.

Step 2: Aligning Regeneration with Previous Attempt. When regenerating, the new token sequence $[y_1, \ldots, y_n]$ may differ in length and content from $[x_1, \ldots, x_{n_-}]$. We establish a monotone alignment to transfer suppression weights: for each position j in the new attempt, find the first matching token x_i from the previous attempt (if any), creating an alignment matrix $A \in \{0,1\}^{n_- \times n}$. This identifies which token should be suppressed at each regeneration position:

$$\text{target}_j = \begin{cases} x_i, & \text{if a match exists at position } i \\ x_j, & \text{otherwise (use position-based fallback)} \end{cases} \tag{4}$$

To avoid over-suppressing tokens that matched only by coincidence (e.g., common words appearing at distant positions), we apply distance decay:

$$\hat{u}_j^{\rightarrow} \;=\; A_{ij} \cdot \exp\Big(-\Big(\tfrac{|i-j|}{\tau}\Big)^{\gamma}\Big) \cdot \hat{u}_i^{(1)} \tag{5}$$

where τ controls the decay rate and $\gamma \in \{1, 2\}$ the decay shape. This reduces suppression strength for matches far from their original position.

We further apply Gaussian smoothing to prevent over-localization, spreading suppression signals across nearby positions (details in Appendix A.3). The final effective suppression at position j is:

$$\alpha_j \;=\; \alpha \cdot u_j^{\star}, \quad \alpha \in (0,1) \tag{6}$$

where $u_j^{\star}$ blends the direct and smoothed suppression signals, and α is a global scaling factor.

Step 3: Probability Redistribution. Let $q_j^{(2)}(v)$ denote the Producer's original next-token distribution at position j during regeneration. We adjust this distribution to suppress the target token while boosting alternatives:

$$s_j(v) \;=\; \begin{cases} 1 - \alpha_j\, r_j(v), & v = \text{target}_j \quad \text{(suppress)} \\ 1 + \kappa_j\, r_j(v)^{\beta}, & v \neq \text{target}_j \quad \text{(boost)} \end{cases} \tag{7}$$

where $r_j(v) = q_j^{(2)}(v) / \max(\varepsilon, \bar{q}_j^{(1)}(v))$ measures how much the model's belief about token v has shifted between attempts. The coefficient κ_j is computed to preserve probability mass:

$$\kappa_j \;=\; \frac{\alpha_j\, r_j(\text{target}_j)\, q_j^{(2)}(\text{target}_j)}{\displaystyle\sum_{u \neq \text{target}_j} q_j^{(2)}(u)\, r_j(u)^{\beta}} \tag{8}$$

ensuring that the total probability removed from the target equals the total added to alternatives (derivation in Appendix A.4). The parameter $\beta \geq 0$ controls how redistributed mass is allocated and is set to 1 for all our experiments.

The final adjusted distribution is:

$$\tilde{q}_j(v) \;=\; \frac{q_j^{(2)}(v)\, s_j(v)}{\sum_{u \in \mathcal{V}} q_j^{(2)}(u)\, s_j(u)} \tag{9}$$

This is a valid probability distribution (non-negative, sums to 1) that explores alternatives while respecting the model's learned beliefs.

Regeneration Process. With the adjusted distribution $\tilde{q}_j(\cdot)$, the Producer samples a new unit. The verifier feedback is also appended to the prompt context, providing semantic guidance alongside the probability adjustment. If this regenerated unit is accepted, it becomes the new checkpoint. If rejected again, the framework rolls back to the previous checkpoint (before the problematic unit) and regenerates from there, recognizing that the error may have originated earlier in the generation. By combining perplexity-driven uncertainty detection with probability-level intervention, Once-More achieves continuous, fine-grained self-correction that prevents error accumulation during generation.

4 Experiments

4.1 Setup and baselines

Due to the rapid development of the LLM field, prior self-correction studies employ diverse experimental setups across different models and benchmarks. To establish reproducible comparisons, we selected the popular open-sourced Qwen3 family of models (4B, 8B, and 14B parameters) as our base models (non-thinking) (Yang et al., 2025a). These model sizes were chosen for three key reasons: (1) they span small to medium scales, revealing how self-correction scales with capacity; (2) they represent practical compute-constrained scenarios; (3) Qwen3 offers state-of-the-art performance at these sizes, ensuring our results are relevant to current practice.

We evaluated Once-More on six benchmarks: AIME 2024 (Art of Problem Solving, 2024a;b) and 2025 (Art of Problem Solving, 2025a;b) for complex mathematical reasoning, LiveBench (reasoning subset) (Rein et al., 2024) for general reasoning, GPQA Diamond (Rein et al., 2024) for graduate-level science questions, and SVAMP (Patel et al., 2021) and GSM8K (Cobbe et al., 2021) for arithmetic word problems. This benchmark selection covers diverse reasoning types and difficulty levels.

We compare the proposed Once-More with four representative baselines:

- **Raw**: Direct generation without self-correction.
- **Self-Refine** (Madaan et al., 2023): The most widely-cited iterative refinement method, representing prompt-based self-correction approaches.
- **CRITIC** (Gou et al., 2024): Incorporates external tools for verification, representing tool-augmented self-correction methods and most recent post-hoc method.
- **S^{3}c-MATH** (Yan et al., 2025): A supervised fine-tuning approach, representing training-based self-correction methods.

Here, we implemented the Self-Refine and CRITIC baselines according to their references. For Self-Refine, the number of refinement iterations was set to $k = 3$. For CRITIC, since no suitable resources were specified in the original work, we adapted the method by leaving the external evidence field empty in math experiments. On the GPQA Diamond benchmark, CRITIC uses the first 1000 words from web search results as external evidence. S^{3}c-MATH results are taken from their published papers on comparable models. All reported results represent means over three independent runs. For Once-More's default configuration, we set K = 5 for top-K perplexity calculation, with suppression factor $\alpha = 1.0$, redistribution sharpness $\beta = 1.0$, distance decay $\tau = 1$, diffusion bandwidth $\sigma = 1$ and perplexity threshold $\eta = 25\%$ fixed. The verifier feedback is generated from an agent using the same LLM as the producer agent across all experiments.

Table 1: Accuracy performance (mean accuracy % ± std. dev. over 3 runs) comparisons between the proposed Once-More and baseline self-correction methods across multiple benchmarks. Results are for Qwen3 models of varying sizes (4B, 8B, 14B parameters). Once-More consistently outperforms iterative refinement approaches (Self-Refine, CRITIC) and raw generation.

		AIME24	**AIME25**	**LiveBench(Reason.)**	**GPQA Diamond**
Qwen3 4B	Raw	13.3 ±3.3	20.0 ±3.3	20.3 ±1.2	43.9 ±1.3
	SelfRefine	13.3 ±3.3	18.9 ±3.8	21.0 ±0.9	43.9 ±1.0
	CRITIC	14.4±1.9	20.0±3.3	20.7±1.3	**48.9**±0.8
	ours	**16.7**±3.4	**23.3**±3.4	**33.0**±1.0	47.5±0.9
Qwen3 8B	Raw	24.4±1.9	16.7±3.4	30.5±0.9	45.4±0.9
	SelfRefine	25.6±3.9	15.5±3.9	31.2±1.3	47.0±1.7
	CRITIC	24.4±1.9	18.8±3.9	31.5±1.3	48.9±1.5
	ours	**33.3**±3.3	**24.4**±1.9	**39.5**±1.3	**49.5**±1.7
Qwen3 14B	Raw	26.7±3.3	18.8±3.8	44.0±1.3	48.0±1.5
	SelfRefine	28.8±3.8	23.3±3.3	46.5±1.5	49.2±1.0
	CRITIC	28.8±1.9	22.2±3.9	45.5±1.5	50.5±1.3
	ours	**36.7**±3.3	**26.7**±3.3	**52.5**±1.8	**55.6**±1.6

Table 2: Accuracy performance comparisons with supervised fine-tuning methods on mathematical benchmarks. Once-More achieves competitive or superior performance without any training.

	Llama 3 8B				**Qwen2 Math 7B**			
Benchmark	MetaMath SFT	$\mathbf{S^3C}$ w/o R&I	$\mathbf{S^3C}$ MathQA	Ours	MetaMath SFT	$\mathbf{S^3C}$ w/o R&I	$\mathbf{S^3C}$ MathQA	Ours
SVAMP	81.4	80.5	81.8	**82.0**	85.6	86.7	87.4	**89.0**
GSM8K	81.1	81.7	**82.9**	79.0	84.1	84.4	84.8	**85.2**

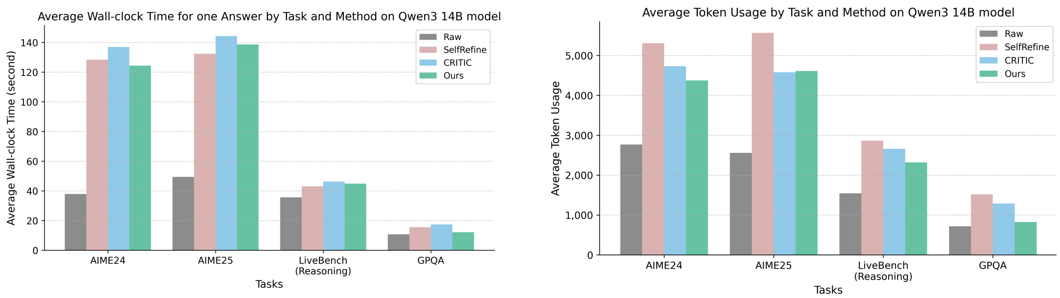
4.2 Main Results

Table 1 presents performance comparisons across the AIME24/25, LiveBench (reasoning), and GPQA Diamond benchmarks. Our proposed Once-More consistently outperforms all baselines across model sizes and task types. On mathematical reasoning tasks (AIME 2024/2025), Once-More achieves 3.4 to 10 point gains over raw generation, with the 14B model reaching 36.7% on AIME 2024 and 26.7% on AIME 2025. In contrast, Self-Refine and CRITIC show minimal improvements, gaining at most 2.1 to 4.5 points. For general reasoning (LiveBench), Once-More delivers 9.0 to 12.7 point improvements, while baselines achieve less than 1 point gain. On GPQA Diamond, all model sizes show consistent 3.6 to 7.6 point improvements with Once-More.

Table 2 compares Once-More against supervised fine-tuning approaches, including MetaMath-SFT and S^3c-MATH. Once-More achieves the best performance on SVAMP across both model families (82.0% for Llama 3 8B, 89.0% for Qwen2 Math 7B) without any training. While S^3c-MATH outperforms Once-More on GSM8K for Llama (82.9% vs 79.0%), Once-More leads on Qwen2 Math (85.2% vs 84.8%). The results demonstrate that Once-More can match or exceed specialized training methods while maintaining model-agnostic deployment.

4.3 Analysis and Ablation Study

Scaling effects in self-correction. On AIME 2024, Once-More's improvements scale progressively: a 25.6% relative gain for 4B, 36.4% for 8B, and 37.4% for 14B models. This scaling pattern suggests that self-correction benefits from richer internal representations in larger models. They can generate more informative feedback and possess more nuanced token distributions for effective logit redistribution. The same pattern holds across AIME 2025, where gains increase from 16.5% (4B) to 42% (14B). This indicates that Once-More can scale with model capacity, unlike post-hoc methods that show diminishing returns.



(a) Average wall-clock time for one answer.

(b) Average token usage for one answer.

Figure 4: Run time comparisons across tasks on Qwen3-14B.

Table 3: Ablation study on Once-More components using Qwen3-14B. Both verifier feedback and logit redistribution contribute substantially, with the best accuracy performance from combining both.

Method	AIME24	GPQA
Raw	26.7	48.0
w/o Redistribution	33.3	51.5
w/o Feedback	30.3	48.4
Full Once-more	**36.7**	**55.6**

Small models and the feedback quality problem. Existing self-correction methods do not work on smaller models. Self-Refine shows no improvement on the 4B model for AIME 2024 and actually degrades performance on AIME 2025 for the 8B model (16.7%→15.5%). CRITIC fares slightly better but still achieves minimal gains. This is likely due to smaller models' limited capabilities to produce meaningful feedback on long outputs. Once-More mitigates this by combining verifier feedback with logit redistribution, ensuring corrections happen even when feedback quality is low. The finer unit-level granularity also makes feedback inherently easier for verifiers to provide.

Mathematical reasoning improvement. On AIME24 and AIME25, Once-More substantially outperforms CRITIC and Self-Refine. This is because mathematical errors are unforgiving; a single misstep can invalidate an entire solution. This makes existing post-hoc methods less reliable as they must locate the faulty step in a complete solution and regenerate a full answer. In contrast, Once-More's continuous monitoring catches mistakes as they occur, enabling correction at the point of failure rather than after completion.

Efficiency analysis. Figure 4 compares wall-clock time and token consumption across methods on Qwen3 14B. Despite its regeneration cycles, Once-More demonstrates competitive wall-clock time, matching or outperforming Self-Refine and CRITIC across all tasks while achieving superior accuracy. For token usage, Once-More consistently achieves the lowest consumption among self-correction methods, using 17-21% fewer tokens than Self-Refine on AIME tasks. This efficiency advantage persists across other benchmarks. Notably, GPQA Diamond exhibits shorter generations across all methods, providing fewer opportunities for continuous intervention and partially explaining the smaller performance gains on this task. Overall, Once-More achieves superior accuracy without incurring additional computational costs compared to existing self-correction approaches.

Effects of verifier feedback and logit redistribution Table 3 evaluates the contributions of feedback and logit redistribution on Qwen3-14B. Removing redistribution but keeping feedback yields 33.3% on AIME24 (+6.6) and 51.5% on GPQA (+3.5). Removing feedback but keeping redistribution gives 30.3% (+3.6) and 48.4% (+0.4). The full system reaches 36.7% and 55.6%, which are gains of +10.0 and +7.6 over raw. The effects are roughly additive on AIME24 and clearly more than additive on GPQA, suggesting that feedback and redistribution resolve different failure modes:

Table 4: Unit length ablation study using Qwen3-14B, with mean accuracy (%) $\pm$ std. dev. over 3 runs.

	Unit length (sentences)					
Benchmark	1	2	4	32	64	128
AIME24	**36.6 $\pm$ 3.3**	34.3 $\pm$ 1.9	35.5 $\pm$ 1.9	30.0 $\pm$ 2.7	26.7 $\pm$ 2.7	26.7 $\pm$ 2.7
LiveBench	**52.3 $\pm$ 1.2**	52.1 $\pm$ 1.0	**52.3 $\pm$ 1.2**	49.8 $\pm$ 1.7	50.3 $\pm$ 1.3	45.3 $\pm$ 0.6
GPQA-D	55.6 $\pm$ 1.8	55.5 $\pm$ 1.3	**56.1 $\pm$ 1.5**	49.7 $\pm$ 1.0	49.3 $\pm$ 0.3	50.0 $\pm$ 0.7

feedback supplies semantic guidance about the error, and redistribution enforces exploration away from the previously chosen tokens so that the guidance actually changes the trajectory.

Effects of answer unit granularity Table 4 evaluates the contributions of answer unit granularity on Qwen3-14B model under default hyper-parameter setting. The results do not indicate significant performance differences for small unit lengths ($\leq$ 4 sentences). This shows that the proposed Once-More framework is robust under fine-grained segmentation. However, when the unit length increases significantly ($\geq$ 32 sentences), performance drops across all three benchmarks and converges to the performance of the raw model (without Once-More) at a unit length of 128.

Table 5: Accuracy (%) results under different Producer–Verifier configurations and raw output.

Producer:	Qwen-14B				Qwen-4B			
Verifier:	14B	8B	4B	Raw	14B	8B	4B	Raw
AIME24	36.6	36.6	30.0	26.6	33.3	23.3	16.7	13.3
LiveBench	52.5	49.5	46.5	44.0	41.0	38.5	33.0	20.3
GPQA-D	55.6	51.5	50.0	48.0	54.0	52.5	47.5	43.9

Asymmetric settings of Producer/Verifier Table 5 presents the results using both configurations of the strong-producer / weak-verifier and the weak-producer / strong-verifier. Experiments conducted under default hyperparameters. Results show that, with a strong Producer (Qwen-14B), performance degrades only mildly with weaker verifiers (4B/8B), remaining substantially better than the raw baseline. With a weak Producer (Qwen-4B), increasing verifier strength (using 14B) yields massive improvements across all benchmarks.

Comprehensive ablation study for hyper-parameters suppression α, distance decay factor τ, diffusion bandwidth σ, and perplexity threshold η can be found at Appendix. A.5.

5 Conclusion

We presented Once-More, a model-agnostic, training-free framework for continuous self-correction during generation, which monitors uncertainty at the unit level, invokes verifier feedback when needed, and enforces exploration via perplexity-guided logit redistribution. This fine-grained, intervene-as-you-go design efficiently reduces error propagation and turns incremental improvements at each unit into stronger end-to-end generations. When tested on diverse benchmarks and model sizes, the proposed Once-More consistently outperforms post-hoc refinement baselines while remaining token-efficient, demonstrating that controllable, inference-time steering can yield reliable gains without additional training. Despite these gains, we acknowledge specific limitations: the framework may struggle to trace errors rooted deep in the generation history, potentially leading to regeneration loops; confident errors (false negatives) may occasionally bypass the perplexity trigger; and the system's performance ceiling remains influenced by verifier quality. Future work will focus on addressing these challenges through dynamic rollback mechanisms and adaptive thresholding.

Reproducibility Statement

To ensure the reproducibility of our work, we provide comprehensive implementation details throughout the paper and supplementary materials. The Once-More framework's complete algorithmic description appears in Appendix A.1, with mathematical formulations for perplexity computation (Section 3.2) and logit redistribution mechanism (Section 3.3). Experimental setup details, including all hyperparameters ($K = 5$, $\alpha = 1.0$, $\beta = 1.0$), model specifications (Qwen3 4B/8B/14B), and benchmark descriptions are provided in Section 4.1. We also detail our implementation procedures for Self-Refine and CRITIC, including specific adaptations for mathematical tasks. All experiments use publicly available models and benchmarks (AIME 2024/2025, LiveBench, GPQA Diamond, SVAMP, GSM8K) with standard evaluation metrics. We report results averaged over three independent runs to account for stochastic variation. An anonymous GitHub repository has been created. It will be used to host the complete implementation of Once-More after this work is published. It will include the Producer-Verifier architecture, perplexity monitoring system, logit redistribution algorithm, and scripts to reproduce all experimental results. The repository is available at: `https://anonymous.4open.science/r/once-more-F54C/`

References

Mikhail L Arbuzov, Alexey A Shvets, and Sisong Beir. Beyond exponential decay: Rethinking error accumulation in large language models. *arXiv preprint arXiv:2505.24187*, 2025.

Kushal Arora, Layla El Asri, Hareesh Bahuleyan, and Jackie Cheung. Why exposure bias matters: An imitation learning perspective of error accumulation in language generation. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (eds.), *Findings of the Association for Computational Linguistics: ACL 2022*, pp. 700–710, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.findings-acl.58. URL `https://aclanthology.org/2022.findings-acl.58/`.

Art of Problem Solving. 2024 aime i. `https://artofproblemsolving.com/wiki/index.php/2024_AIME_I`, 2024a. Accessed: 2025.

Art of Problem Solving. 2024 aime ii. `https://artofproblemsolving.com/wiki/index.php/2024_AIME_II`, 2024b. Accessed: 2025.

Art of Problem Solving. 2025 aime i. `https://artofproblemsolving.com/wiki/index.php/2025_AIME_I`, 2025a. Accessed: 2025.

Art of Problem Solving. 2024 aime ii. `https://artofproblemsolving.com/wiki/index.php/2025_AIME_II`, 2025b. Accessed: 2025.

Justin Chih-Yao Chen, Archiki Prasad, Swarnadeep Saha, Elias Stengel-Eskin, and Mohit Bansal. Magicore: Multi-agent, iterative, coarse-to-fine refinement for reasoning. *arXiv preprint arXiv:2409.12147*, 2024.

Christopher Manning. Natural language processing with deep learning cs224n/ling284, lecture 6. `https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1214/slides/cs224n-2021-lecture06-fancy-rnn.pdf`, 2021. Accessed: 2025.

Yung-Sung Chuang, Yujia Xie, Hongyin Luo, Yoon Kim, James Glass, and Pengcheng He. Dola: Decoding by contrasting layers improves factuality in large language models. *arXiv preprint arXiv:2309.03883*, 2023.

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

Angela Fan, Mike Lewis, and Yann Dauphin. Hierarchical neural story generation. *arXiv preprint arXiv:1805.04833*, 2018.

Mohamed Amine Ferrag, Norbert Tihanyi, and Merouane Debbah. From llm reasoning to autonomous ai agents: A comprehensive review. *arXiv preprint arXiv:2504.19678*, 2025.

Yichao Fu, Xuewei Wang, Yuandong Tian, and Jiawei Zhao. Deep think with confidence, 2025. URL https://arxiv.org/abs/2508.15260.

Garry A Gabison and R Patrick Xian. Inherent and emergent liability issues in llm-based agentic systems: a principal-agent perspective. *arXiv preprint arXiv:2504.03255*, 2025.

Zhibin Gou, Zhihong Shao, Yeyun Gong, yelong shen, Yujiu Yang, Nan Duan, and Weizhu Chen. CRITIC: Large language models can self-correct with tool-interactive critiquing. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=Sx038qxjek.

Ari Holtzman, Jan Buys, Maxwell Forbes, Antoine Bosselut, David Golub, and Yejin Choi. Learning to write with cooperative discriminators. *arXiv preprint arXiv:1805.06087*, 2018.

Hugging Face. Perplexity of fixed-length models. https://huggingface.co/docs/transformers/en/perplexity, 2025. Accessed: 2025.

Fred Jelinek, Robert L Mercer, Lalit R Bahl, and James K Baker. Perplexity—a measure of the difficulty of speech recognition tasks. *The Journal of the Acoustical Society of America*, 62(S1): S63–S63, 1977.

Ryo Kamoi, Yusen Zhang, Nan Zhang, Jiawei Han, and Rui Zhang. 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, 2024.

Charles Koutcheme, Nicola Dainese, Sami Sarsa, Arto Hellas, Juho Leinonen, and Paul Denny. Open source language models can provide feedback: Evaluating llms' ability to help students using gpt-4-as-a-judge. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1*, pp. 52–58. Association for Computing Machinery, 2024.

Ben Krause, Akhilesh Deepak Gotmare, Bryan McCann, Nitish Shirish Keskar, Shafiq Joty, Richard Socher, and Nazneen Fatema Rajani. Gedi: Generative discriminator guided sequence generation. *arXiv preprint arXiv:2009.06367*, 2020.

Xiang Lisa Li, Ari Holtzman, Daniel Fried, Percy Liang, Jason Eisner, Tatsunori Hashimoto, Luke Zettlemoyer, and Mike Lewis. Contrastive decoding: Open-ended text generation as optimization. *arXiv preprint arXiv:2210.15097*, 2022.

Dancheng Liu, Amir Nassereldine, Ziming Yang, Chenhui Xu, Yuting Hu, Jiajie Li, Utkarsh Kumar, Changjae Lee, Ruiyang Qin, Yiyu Shi, et al. Large language models have intrinsic self-correction ability. *arXiv preprint arXiv:2406.15673*, 2024.

Zixuan Liu, Guangkai Jiang, and Siavash Khajavi. Llava-sco: Teach vision language models to self-correct. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pp. 3415–3424, June 2025.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36:46534–46594, 2023.

Sean O'Brien and Mike Lewis. Contrastive decoding improves reasoning in large language models. *arXiv preprint arXiv:2309.09117*, 2023.

Liangming Pan, Michael Saxon, Wenda Xu, Deepak Nathani, Xinyi Wang, and William Yang Wang. Automatically correcting large language models: Surveying the landscape of diverse automated correction strategies. *Transactions of the Association for Computational Linguistics*, 12:484–506, 2024.

Melissa Z Pan, Mert Cemri, Lakshya A Agrawal, Shuyi Yang, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Kannan Ramchandran, Dan Klein, Joseph E. Gonzalez, Matei Zaharia, and Ion Stoica. Why do multiagent systems fail? In *ICLR 2025 Workshop on Building Trust in Language Models and Applications*, 2025. URL https://openreview.net/forum?id=wM521FqPvI.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. Are NLP models really able to solve simple math word problems? In Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tur, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou (eds.), *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 2080–2094, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.168. URL https://aclanthology.org/2021.naacl-main.168/.

Debjit Paul, Mete Ismayilzada, Maxime Peyrard, Beatriz Borges, Antoine Bosselut, Robert West, and Boi Faltings. Refiner: Reasoning feedback on intermediate representations. *arXiv preprint arXiv:2304.01904*, 2023.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. Gpqa: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*, 2024.

Florian Schmidt. Generalization in generation: A closer look at exposure bias. *arXiv preprint arXiv:1910.00292*, 2019.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *Advances in Neural Information Processing Systems*, 36:38154–38180, 2023.

Zhuocheng Shen. Llm with tools: A survey. *arXiv preprint arXiv:2409.18807*, 2024.

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

Jiankai Sun, Chuanyang Zheng, Enze Xie, Zhengying Liu, Ruihang Chu, Jianing Qiu, Jiaqi Xu, Mingyu Ding, Hongyang Li, Mengzhe Geng, et al. A survey of reasoning with foundation models: Concepts, methodologies, and outlook. *ACM Computing Surveys*, 57(11):1–43, 2025.

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

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

Renxi Wang, Haonan Li, Xudong Han, Yixuan Zhang, and Timothy Baldwin. Learning from failure: Integrating negative examples when fine-tuning large language models as agents. *arXiv preprint arXiv:2402.11651*, 2024a.

Shenzhi Wang, Le Yu, Chang Gao, Chujie Zheng, Shixuan Liu, Rui Lu, Kai Dang, Xiong-Hui Chen, Jianxin Yang, Zhenru Zhang, Yuqiong Liu, An Yang, Andrew Zhao, Yang Yue, Shiji Song, Bowen Yu, Gao Huang, and Junyang Lin. Beyond the 80/20 rule: High-entropy minority tokens drive effective reinforcement learning for LLM reasoning. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025a. URL https://openreview.net/forum?id=yfcpdY4gMP.

Yifei Wang, Yuyang Wu, Zeming Wei, Stefanie Jegelka, and Yisen Wang. A theoretical understanding of self-correction through in-context alignment. *Advances in Neural Information Processing Systems*, 37:89869–89912, 2024b.

Yizhou Wang, Lingzhi Zhang, Yue Bai, Mang Tik Chiu, Zhengmian Hu, Mingyuan Zhang, Qihua Dong, Yu Yin, Sohrab Amirghodsi, and Yun Fu. Cautious next token prediction. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 25685–25697, Vienna, Austria, July 2025b. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.1318. URL https://aclanthology.org/2025.findings-acl.1318/.

Colin White, Samuel Dooley, Manley Roberts, Arka Pal, Ben Feuer, Siddhartha Jain, Ravid Shwartz-Ziv, Neel Jain, Khalid Saifullah, Sreemanti Dey, et al. Livebench: A challenging, contamination-limited llm benchmark. *arXiv preprint arXiv:2406.19314*, 2024.

Wenda Xu, Guanglei Zhu, Xuandong Zhao, Liangming Pan, Lei Li, and William Wang. Pride and prejudice: Llm amplifies self-bias in self-refinement. In *ACL (1)*, pp. 15474–15492, 2024. URL https://doi.org/10.18653/v1/2024.acl-long.826.

Yuchen Yan, Jin Jiang, Yang Liu, Yixin Cao, Xin Xu, Mengdi Zhang, Xunliang Cai, and Jian Shao. S3c-math: spontaneous step-level self-correction makes large language models better mathematical reasoners. In *Proceedings of the Thirty-Ninth AAAI Conference on Artificial Intelligence and Thirty-Seventh Conference on Innovative Applications of Artificial Intelligence and Fifteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'25/IAAI'25/EAAI'25. AAAI Press, 2025. ISBN 978-1-57735-897-8. doi: 10.1609/aaai.v39i24.34749. URL https://doi.org/10.1609/aaai.v39i24.34749.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025a.

Zhen Yang, Mingyang Zhang, Feng Chen, Ganggui Ding, Liang Hou, Xin Tao, Pengfei Wan, and Ying-Cong Chen. Less is more: Improving llm reasoning with minimal test-time intervention, 2025b. URL https://arxiv.org/abs/2510.13940.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

Ruochen Zhao, Xingxuan Li, Shafiq Joty, Chengwei Qin, and Lidong Bing. Verify-and-edit: A knowledge-enhanced chain-of-thought framework. *arXiv preprint arXiv:2305.03268*, 2023.

Tianyu Zheng, Ge Zhang, Tianhao Shen, Xueling Liu, Bill Yuchen Lin, Jie Fu, Wenhu Chen, and Xiang Yue. Opencodeinterpreter: Integrating code generation with execution and refinement. *arXiv preprint arXiv:2402.14658*, 2024.

A APPENDIX

A.1 ONCE-MORE ALGORITHM

Algorithm 1 Once-More Framework

Require: Producer LLM, Tokenizer, Verifier set $\mathcal{R}$, Goal x, role prompt, max tokens $T_{\max}$

Init:
Build chat prompt from (role prompt, x);
Encode to token id: step_ids.
Set past $\leftarrow \emptyset$, Output $\leftarrow \emptyset$, tokens $\leftarrow 0$. consecutive fails $\leftarrow 0$
Initialize ckpt_stack $\leftarrow [(0, \mathit{step\_ids})]$, supp_stack $\leftarrow$ [*None*], suppression $\leftarrow$ *None*.
for $i = 1$ to $T_{\max}$ **do**
  Propose: Generate sentence s and update $(\mathit{past}, \mathit{step\_ids}, \mathit{suppression})$.
  Monitor:
  Compute reliability $\rightarrow (ok, \lambda)$.
  Update supp_stack with $(\lambda, t_{1:L}, \ell_{1:L})$.
  Update tokens $\leftarrow \mathit{tokens} + L$.
  if not ok and $i > 1$ **then**
    Verify: Verifier judge $(\mathit{accept}, F) \leftarrow$ Judge($\mathcal{R}, x, \mathit{Output}, s$).
    if not accept **and** consecutive fails < 4 **then**
      Rollback & Guide:
      Restore last checkpoint $(k^\star, \mathit{ids}^\star)$;
      set suppression from supp_stack;
      trim past to $k^\star$;
      reset step_ids $\leftarrow \mathit{ids}^\star$;
      pop stacks;
      append feedback F to step_ids;
      consecutive fails += 1
      continue.
    end if
  end if
  Commit:
  Append s to Output;
  push new checkpoints to ckpt_stack and supp_stack;
  reset suppression $\leftarrow$ *None*.
  reset consecutive $\leftarrow 0$
  if last token of step_ids is eos **or** tokens $> T_{\max}$ **then**
    break
  end if
end for
return Output

A.2 QUANTILE CALIBRATION FOR THE RELIABILITY THRESHOLD

We set the verification trigger by calibrating a *unit perplexity* threshold from a short, held-out run.

Inputs and unit count. For each calibration prompt $i = 1, \ldots, D$, run the Producer once *without* verification and compute a unit-level perplexity P_{unit} for every emitted unit (as in Section 3.2). If the i-th output contains U_i units, it contributes U_i scalar values. The total number of calibration units is

$$M \;=\; \sum_{i=1}^{D} U_i \,.$$

Collect all unit perplexities as $\{P^{(m)}\}_{m=1}^{M}$.

Empirical quantile. Form the empirical distribution by *sorting the values themselves* in ascending order (this sorting step ignores the original time order), yielding the order statistics

$$P_{(1)} \;\le\; P_{(2)} \;\le\; \cdots \;\le\; P_{(M)} \,.$$

Fix a target verification rate $\eta \in (0, 1)$ so that roughly the largest η fraction of units will be verified at run time. Set $q = 1 - \eta$ and compute

$$h \;=\; 1 + (M-1)\,q, \qquad k = \lfloor h \rfloor, \qquad \gamma = h - k,$$

then define the calibrated threshold

$$P_{\text{th}} \;=\; \begin{cases} P_{(1)}, & h \le 1, \\ P_{(M)}, & h \ge M, \\ (1-\gamma)\,P_{(k)} \;+\; \gamma\,P_{(k+1)}, & \text{otherwise.} \end{cases}$$

At inference time, verify a provisional unit Y whenever $P_{\text{unit}}(Y) > P_{\text{th}}$.

A.3 MATHEMATICAL PROPERTIES OF THE FRAMEWORK

We now establish formal properties of the Once-More framework's perplexity computation and probability adjustment mechanism.

A.3.1 PROPERTIES OF PERPLEXITY-BASED MONITORING

Property 1 (Perplexity Lower Bound). For any position j and $K \ge 1$, the token-level perplexity satisfies $\text{PPL}_j^{(K)} \ge 1$.

Proof. By definition, $\text{PPL}_j^{(K)} = \exp(\text{ANLL}_j^{(K)})$ where

$$\text{ANLL}_j^{(K)} = \frac{1}{K}\sum_{i=1}^{K}(-\log q_j(v_{j,i})) \tag{10}$$

Since $q_j(v_{j,i}) \in (0, 1]$ for valid probabilities, we have $-\log q_j(v_{j,i}) \ge 0$. The lower bound of 1 is achieved when $K = 1$ and q_j places all mass on a single token (i.e., $q_j(v_{j,1}) = 1$), giving $-\log(1) = 0$ and $\exp(0) = 1$.

Property 2 (Sensitivity Control via K). Larger values of K increase the sensitivity of perplexity to distribution spread, while smaller values focus on top candidates.

Justification. When K is small, $\text{PPL}_j^{(K)}$ reflects only the most probable tokens. If the top token has a high probability, perplexity remains low even if tail probabilities are spread. As K increases, more of the distribution is captured, increasing sensitivity to uncertainty in lower-ranked alternatives. This creates a trade-off: larger K detects subtle uncertainties but increases variance; smaller K provides conservative, stable estimates.

A.3.2 PROPERTIES OF GUIDED REGENERATION

Property 3 (Mass Balance). Given suppression factors satisfying $0 \leq \alpha_j r_j(\text{target}_j) < 1$ and redistribution coefficient κ_j as defined in Section 3.3, the probability mass removed from the target token exactly equals the mass added to alternatives before normalization.

Proof. See Appendix A.4 for the complete derivation of κ_j, which explicitly enforces this constraint.

Property 4 (Valid Probability Distribution). The adjusted distribution $\tilde{q}_j(\cdot)$ is a valid probability distribution (non-negative, sums to 1).

Proof. By construction:

1. **Non-negativity:** For the target token, $s_j(\text{target}_j) = 1 - \alpha_j r_j(\text{target}_j) > 0$ by the constraint $\alpha_j r_j(\text{target}_j) < 1$. For alternatives, $s_j(v) = 1 + \kappa_j r_j(v)^\beta > 0$ since $\kappa_j > 0$ (as long as alternatives exist) and $r_j(v) \geq 0$. Since $q_j^{(2)}(v) \geq 0$ for all v, we have $q_j^{(2)}(v) s_j(v) \geq 0$.
2. **Normalization:** The denominator $\sum_{u \in \mathcal{V}} q_j^{(2)}(u) s_j(u) > 0$ by non-negativity and the fact that $q_j^{(2)}$ is a valid distribution. The normalization explicitly ensures $\sum_{v \in \mathcal{V}} \tilde{q}_j(v) = 1$.

Property 5 (Monotone Suppression). For fixed $q_j^{(2)}$ and importance ratios r_j, the adjusted probability $\tilde{q}_j(\text{target}_j)$ strictly decreases with α_j within the admissible range.

Proof. From the definition:

$$\tilde{q}_j(\text{target}_j) = \frac{q_j^{(2)}(\text{target}_j)(1 - \alpha_j r_j(\text{target}_j))}{\sum_u q_j^{(2)}(u) s_j(u)} \tag{11}$$

Taking the derivative with respect to α_j (applying the quotient rule and using the fact that κ_j depends on α_j), one can show that:

$$\frac{\partial \tilde{q}_j(\text{target}_j)}{\partial \alpha_j} < 0 \tag{12}$$

The key insight is that increasing α_j directly reduces the target token's weight while increasing alternatives' weights, creating a double effect.

Property 6 (Controlled Exploration via β). The tempering exponent β controls the distribution of redistributed mass:

- $\beta = 0$: Mass spreads uniformly across all alternatives
- $\beta = 1$: Mass allocates proportionally to importance ratios $r_j(v)$
- $\beta > 1$: Mass concentrates on tokens with the largest importance ratios

Justification. The boost factor for alternatives is $1 + \kappa_j r_j(v)^\beta$. When $\beta = 0$, this becomes $1 + \kappa_j$ (constant across all alternatives), distributing mass uniformly. As β increases, $r_j(v)^\beta$ amplifies differences: tokens with $r_j(v) > 1$ (model now favours more) receive exponentially more boost, while tokens with $r_j(v) < 1$ receive less. This creates an exploration-exploitation trade-off controlled by β.

Property 7 (Stability of Adjustment). When $q_j^{(2)}$ remains close to $q_j^{(1)}$ and $\beta \leq 1$, the KL divergence between the adjusted and original distributions scales linearly with maximum suppression:

$$\mathrm{KL}\big(\tilde{q}_j \,\|\, q_j^{(2)}\big) = O(\alpha_{\max}) \tag{13}$$

Proof sketch. The KL divergence is:

$$\mathrm{KL}\big(\tilde{q}_j \,\|\, q_j^{(2)}\big) = \sum_v \tilde{q}_j(v) \log \frac{\tilde{q}_j(v)}{q_j^{(2)}(v)} \tag{14}$$

When α_j is small, $s_j(v) \approx 1$ for all v, making $\tilde{q}_j(v) \approx q_j^{(2)}(v)/Z$ where $Z \approx 1$. A Taylor expansion around $\alpha_j = 0$ shows that the leading term is linear in α_j. The condition $\beta \leq 1$ ensures that the redistribution doesn't create sharp peaks that would increase divergence. For detailed calculation, note that:

$$\log \frac{\tilde{q}_j(v)}{q_j^{(2)}(v)} = \log s_j(v) - \log Z \approx s_j(v) - 1 - (Z - 1) \tag{15}$$

where both $s_j(v) - 1$ and $Z - 1$ are $O(\alpha_j)$.

Property 8 (Alignment and Diffusion Regularization). The combination of distance-decayed alignment and Gaussian diffusion prevents both over-localization and excessive spread of suppression signals.

Justification. The distance decay $\exp(-(|i-j|/\tau)^\gamma)$ reduces suppression for tokens matched far from their original position, preventing spurious long-range alignments from causing inappropriate suppression. The Gaussian diffusion with bandwidth σ then smooths the resulting signal:

$$\bar{u}_j = \frac{1}{Z_j} \sum_{i=1}^{n} \exp\left(-\frac{(j-i)^2}{2\sigma^2}\right) \hat{u}_i^{\rightarrow} \tag{16}$$

As $\sigma \to 0$, this reduces to position-wise suppression $\bar{u}_j \to \hat{u}_j^{\rightarrow}$ (localized). As σ increases, suppression spreads to neighboring positions (regularized). The final blend $u_j^\star = 0.5\hat{u}_j^{\rightarrow} + 0.5\bar{u}_j$ balances both effects, preventing over-fitting to specific positions while maintaining locality.

A.4 Derivation of the Redistribution Constant

We now derive the formula for κ_j that ensures probability mass conservation during the adjustment process.

Setup. At position j, we adjust the original distribution $q_j^{(2)}(v)$ by applying scaling factors $s_j(v)$:

$$s_j(v) = \begin{cases} 1 - \alpha_j r_j(v), & v = \text{target}_j \\ 1 + \kappa_j r_j(v)^\beta, & v \neq \text{target}_j \end{cases} \tag{17}$$

where:

- $\alpha_j \in (0, 1)$ is the suppression strength
- $r_j(v) = q_j^{(2)}(v) / \max(\varepsilon, \bar{q}_j^{(1)}(v))$ is the importance ratio
- $\beta \geq 0$ controls redistribution sharpness
- κ_j is the redistribution coefficient to be determined

Mass Conservation Constraint. For a valid probability distribution, the total probability mass before and after adjustment must be equal. Define:

Mass removed from target:

$$\begin{aligned} \Delta^- &= q_j^{(2)}(\text{target}_j) - q_j^{(2)}(\text{target}_j) \cdot s_j(\text{target}_j) && (18) \\ &= q_j^{(2)}(\text{target}_j) - q_j^{(2)}(\text{target}_j) \cdot (1 - \alpha_j r_j(\text{target}_j)) && (19) \\ &= \alpha_j r_j(\text{target}_j) \cdot q_j^{(2)}(\text{target}_j) && (20) \end{aligned}$$

Mass added to alternatives:

$$\Delta^+ = \sum_{u \neq \text{target}_j} \left[q_j^{(2)}(u) \cdot s_j(u) - q_j^{(2)}(u) \right] \tag{21}$$

$$= \sum_{u \neq \text{target}_j} \left[q_j^{(2)}(u) \cdot (1 + \kappa_j r_j(u)^\beta) - q_j^{(2)}(u) \right] \tag{22}$$

$$= \sum_{u \neq \text{target}_j} q_j^{(2)}(u) \cdot \kappa_j r_j(u)^\beta \tag{23}$$

$$= \kappa_j \sum_{u \neq \text{target}_j} q_j^{(2)}(u) \cdot r_j(u)^\beta \tag{24}$$

Solving for κ_j. Setting $\Delta^- = \Delta^+$ for mass conservation:

$$\alpha_j r_j(\text{target}_j) \cdot q_j^{(2)}(\text{target}_j) = \kappa_j \sum_{u \neq \text{target}_j} q_j^{(2)}(u) \cdot r_j(u)^\beta \tag{25}$$

Solving for κ_j:

$$\boxed{\kappa_j = \frac{\alpha_j r_j(\text{target}_j) \cdot q_j^{(2)}(\text{target}_j)}{\sum_{u \neq \text{target}_j} q_j^{(2)}(u) \cdot r_j(u)^\beta}} \tag{26}$$

Normalization. After applying the scaling factors, we normalize to obtain the final distribution:

$$\tilde{q}_j(v) = \frac{q_j^{(2)}(v) \cdot s_j(v)}{\sum_{u \in \mathcal{V}} q_j^{(2)}(u) \cdot s_j(u)} \tag{27}$$

By construction (mass conservation), the numerator of the normalization constant equals:

$$\sum_u q_j^{(2)}(u) \cdot s_j(u) = q_j^{(2)}(\text{target}_j) \cdot s_j(\text{target}_j) + \sum_{u \neq \text{target}_j} q_j^{(2)}(u) \cdot s_j(u) \tag{28}$$

$$= \left[q_j^{(2)}(\text{target}_j) - \Delta^- \right] + \left[\sum_{u \neq \text{target}_j} q_j^{(2)}(u) + \Delta^+ \right] \tag{29}$$

$$= \sum_u q_j^{(2)}(u) + (\Delta^+ - \Delta^-) \tag{30}$$

$$= 1 + 0 = 1 \tag{31}$$

Thus, the distribution is properly normalized even before the explicit normalization step, confirming that our choice of κ_j correctly preserves mass.

Numerical Safeguards. In practice, we implement several safeguards:

- Use $\varepsilon = 10^{-8}$ in importance ratios to avoid division by zero
- Clip $\alpha_j r_j(\text{target}_j)$ to ensure it remains strictly less than 1
- If the denominator in κ_j is extremely small (no viable alternatives), either reduce β toward 0 to spread mass broadly or cap κ_j and trigger a fresh generation attempt
- Verify that $\sum_u q_j^{(2)}(u) s_j(u) > 0$ before normalization

A.5 Ablation Study

A.5.1 Analysis of Suppression Strength α

The ablation study in α (suppression strength) is carried out using the Qwen-14B model on AIME24, Livebench, and GPQA-Diamond. Results are reported in Table 6 from a single run. All other parameters are chosen as the default value.

Table 6: Ablation study on suppression strength α using Qwen3 14B. Results from a single run.

	$\alpha = 0.1$	$\alpha = 0.5$	$\alpha = 1$	$\alpha = 1.5$	$\alpha = 2$
AIME24	30.0	33.3	36.6	36.6	36.6
LiveBench	49.5	52.5	54.5	54.5	54.0
GPQA-D	50.5	51.5	52.0	52.5	51.0

For all benchmarks, performance is relatively stable for suppression strengths $\alpha \geq 1$. While a small suppression strength ($\alpha = 0.1$) dilutes the effect of our proposed Once-More Framework and achieves the worst performance in the ablation study.

A.5.2 Analysis of Distance Decay Factor τ

The ablation study on τ (distance decay factor) is conducted by Qwen-14B model on AIME24, Livebench and GPQA-Diamond. The granularity of τ is set to 0.01 (extremely small τ representing decay to 0, suppression only works on the target token in this specific position), 0.5, 1, 1.5 and 100 (extremely large τ representing no decay, the target token will be equally suppressed regardless of its position). Results are reported in Table 7 from a single run. All other parameters are chosen as the default value.

Table 7: Ablation on distance decay factor τ. Accuracy from single run.

Benchmark	$\tau = 0.01$	$\tau = 0.5$	$\tau = 1$	$\tau = 1.5$	$\tau = 100$
AIME24	23.3	36.6	36.6	36.6	30.3
LiveBench	43.5	52.5	52.5	52.0	50.5
GPQA-D	45.5	54.5	55.0	55.0	52.5

The ablation results show a very interesting phenomenon. Without distance decay (an extremely small $\tau = 0.01$), model performance decreases significantly, becoming even worse than the raw model baseline. Under this setting, the Once-More framework only suppresses the target token if it is in the exact same position as in the previously rejected sequence. The model sometimes finds a trick to regenerate. Assuming this is the previously rejected unit:

```
This sequence contains an error and was therefore rejected
```

Producer first generates a few other words and then repeats exactly the same sequence as the previously rejected one, just like:

```
Got it! This sequence contains an error and therefore got
↪ rejected
```

Since the position is mismatched by these beginning words, the later regenerated sequence is not suppressed at all. However, this will be subsequently rejected by the reviewer, causing the model to get stuck in a loop until the maximum token limit is reached.

Table 8: Ablation on diffusion bandwidth σ. Accuracy from a single run.

Benchmark	$\sigma = 0.1$	$\sigma = 1$	$\sigma = 5$	$\sigma = 10$
AIME24	33.3	36.3	26.6	30.0
LiveBench	52.5	55.0	48.5	47.5
GPQA-D	50.5	52.5	49.0	49.5

A.5.3 Analysis of Diffusion Bandwidth σ

Table 8 reports the effect of diffusion bandwidth σ which controls how suppression spreads across neighboring tokens. The results show that a large $\sigma(\geq 5)$ spreads the suppression too broadly. It dilutes the suppression effect on the target token and behaves similarly to using a very small suppression strength (α). Conversely, a very small σ restricted suppression only to the current token, making the adjustment too localized and also degrading accuracy.

A.5.4 Analysis of Perplexity Threshold η

Since the parameter K (top-K for perplexity estimation) directly defines unit perplexity and consequently impacts the threshold ratio η, we conducted a joint ablation study of these two parameters. Table 9 and Table 10 show that, for small K (1 or 5), changing η primarily impacts runtime rather than accuracy. A looser threshold (η = 50%) increases verification calls and computation time, while a stricter threshold (η = 12.5%) improves efficiency. Large K values generally degrade accuracy.

Table 9: AIME24 results: accuracy and wall-clock time across perplexity threshold η and top-K settings.

	$\eta = 12.5\%$		$\eta = 25\%$		$\eta = 50\%$	
	Acc.	Time (s)	Acc.	Time (s)	Acc.	Time (s)
$K = 1$	36.6	1693	36.6	3579	36.6	6448
$K = 5$	36.6	2448	33.3	4133	36.6	6103
$K = 10$	33.3	2397	33.3	3142	36.6	5954
$K = 15$	30.0	2929	33.3	2851	33.3	7088

Table 10: GPQA results: accuracy and wall-clock time across perplexity threshold η and top-K settings.

	$\eta = 12.5\%$		$\eta = 25\%$		$\eta = 50\%$	
	Acc.	Time (s)	Acc.	Time (s)	Acc.	Time (s)
$K = 1$	51.0	1987	54.5	3640	61.6	4953
$K = 5$	47.5	2448	55.5	3848	55.0	4424
$K = 10$	49.0	2397	49.5	2098	53.5	4244
$K = 15$	48.4	2473	49.5	2486	55.0	3412

A.6 Additional Observations

A.6.1 Intermediate Reasoning Behaviour

Here is one example that the proposed Once-More framework exhibits intermediate reasoning behaviour. The following text is the partial output from AIME24 problem 83:

```
We are given a 2x3 grid where each cell contains a digit from 0 to
↪ 9. We need to fill the grid such that:
1. The **sum of the two numbers formed by reading left to right**
↪ (i.e., the top row and bottom row) is **999**.

[Perplexity acceptable: False]
[Judge] Output accepted.

2. The **sum of the three numbers formed by reading top to bottom**
↪ (i.e., the first column, second column, and third column) is **99**.

[Perplexity acceptable: True]
```

```
Let’s denote the grid as follows:
$$
\begin{array}{|c|c|c|}
\hline
a & b & c \\
\hline
d & e & f \\
\hline
\end{array}
$$
### Step 1: Understand the horizontal sums
We are told that:
$$
abc + def = 999
$$
Here, $abc$ and $def$ are 3-digit numbers formed by the digits in
↪ the top row and bottom row, respectively.

[Perplexity acceptable: False]
[Judge] Output accepted.

past_cache length: 400
That is:
- $abc = 100a + 10b + c$
- $def = 100d + 10e + f$
So:
$$
(100a + 10b + c) + (100d + 10e + f) = 999 \tag{1}
$$
### Step 2: Understand the vertical sums
We are told that:
$$
a + d + e = 99 \quad \text{(first column)} \\
b + e + f = 99 \quad \text{(second column)} \\
c + f = 99 \quad \text{(third column)}
$$

[Perplexity acceptable: False]
[Judge] Output rejected.
[Rolling back to the corresponding checkpoint]
[Feedback message to the model]:
 (The vertical sums must involve multi-digit numbers, not single
↪ digits. Re-express the vertical sums as $ad + de + ef = 99$, etc.,
↪ or clarify the problem statement.)

Clarifying: the problem says "the sum of the three numbers formed by
↪ reading top to bottom is 99". That means:
- The first column forms a number: $a \times 100 + d \times 10 +
↪ ...$, but that would be too large.

[Perplexity acceptable: False]
[Judge] Output rejected.
[Rolling back to the corresponding checkpoint]
[Feedback message to the model]:
 (Clarify how the top-to-bottom numbers are formed (e.g.,
↪ single-digit numbers or multi-digit). The current interpretation is
↪ unclear.)
```

```
Let's assume that the top-to-bottom numbers are formed by reading each of the three columns as a single-digit value (i.e., just the digit in each column). Then the sum of these three digits is 99, which is impossible since the maximum sum of three digits is $9+9+9=27$. So this interpretation is invalid.

[Perplexity acceptable: False]
[Judge] Output accepted.

Let's try a different interpretation: The **top-to-bottom** numbers are formed by concatenating the two digits in each column to form a two-digit number.

[Perplexity acceptable: False]
[Judge] Output accepted.

For example, if the first column has digits $a$ and $b$, then the number is $10a + b$. Then we can add up the three such numbers for the three columns, and that total should be 99.

So now we have:

- The top row is a 3-digit number: $100a + 10b + c$
- The bottom row is a 3-digit number: $100d + 10e + f$
- The sum of these two numbers is **999**:
  $(100a + 10b + c) + (100d + 10e + f) = 999$ | (Equation 1)

- The sum of the three column-wise two-digit numbers is **99**:
  $(10a + d) + (10b + e) + (10c + f) = 99$ | (Equation 2)

We are to find the number of such digit arrangements that satisfy both equations.
```

This partial output shows how Once-More conducted one successful self-correction.

A.7 Practical Implementation Notes

Length changes and alignment. If the revised unit $Y' = [y'_1, \ldots, y'_m]$ has $m \neq n$:

- If $m < n$, positions $j > m$ do not apply; early stopping may be the correction.
- If $m > n$, positions $j > n$ are generated without suppression (they did not occur previously).
- For $j \leq \min(m, n)$ we apply the update above. Optionally, one can align by minimal edit distance when the new attempt diverges early; our experiments did not require this.

Chunking strategy and context management

- **Choosing the unit.** For prose, we default to sentence boundaries with a max token cap, splitting very long sentences into clauses; for code, we use syntax-aware spans (function, block, or diff hunk). Smaller units give finer correction. Larger units preserve coherence.
- **Boundary overlap.** We allow a short overlap window when proposing the next unit and let the Verifier inspect a sliding window that straddles the boundary to avoid seam artifacts.
- **Context growth.** For long outputs, we keep a rolling buffer of recent accepted text, optionally with compressed summaries of older content, to stay within context limits.
- **Multiple verifiers.** Factuality, format, and safety checks can run in parallel with conjunctive acceptance, or in stages (cheap checks first). Tool use follows standard agent patterns.

A.8 Example prompts

Below are the prompts we used in Once-More for the *Producer*, *Verifier A* (Formal & Local), *Verifier B* (Global/Sanity), and an optional *Adjudicator*.

Shared verifier `SYSTEM` (use for both verifiers).

```
You are a verifier. You judge exactly one CURRENT_SPAN in a partial
↪ solution.
Do not solve the whole task. Be precise and conservative.

Inputs:
- TASK: the problem/question
- ACCEPTED_CONTEXT: the already-accepted prior steps/state
- CURRENT_SPAN: the producer's proposed new step(s) to add now
```

GPQA-style graduate Q&A (science)

Producer (`SYSTEM+USER`)

```
[SYSTEM]
You are a graduate-level problem solver. Solve the problem step by
↪ step, separated by a period. Your answer should be chosen from
↪ options A, B, C, D and end with:
Final answer: <A/B/C/D>

[USER]
QUESTION:
{{GPQA_QUESTION_STEM_AND_OPTIONS}}
```

Verifier A (Formal & Local) (`USER` only; use shared SYSTEM above)

```
[USER]
TASK: {{GPQA_QUESTION_STEM_AND_OPTIONS}}
ACCEPTED_CONTEXT: {{PRIOR_NOTES_OR_NONE}}
CURRENT_SPAN: {{PRODUCER_PARAGRAPH}}

You are a very strict verifier with Rubric:
R1 algebraic legality (no invalid cancellations, correct radical/log
↪ rules),
R2 arithmetic accuracy,
R3 domain/branch/constraints respected and stated,
R4 check carefully for the hold of equations/inequalities,
R5 check the scientific correctness of each claim.
R6 Flag missing premises, leaps, or contradictions.
R7 Counterfactual statement Checking.

Does the input unit in the right track to achieve the goal, given
↪ the context? Verify the input unit with your role and task. Only
↪ reject the answer according to your role.
The input may not necessarily solve the goal directly as there are
↪ more details in the upcoming text. If yes, answer Yes. If no, answer
↪ No and provide repair hints beginning with 'Suggestion' to improve
↪ in no more than 20 words.
```

Verifier B (Global/Sanity) (`USER` only; use shared SYSTEM above; optional)

```
[USER]
TASK: {{GPQA_QUESTION_STEM_AND_OPTIONS}}
```

```
ACCEPTED_CONTEXT: {{PRIOR_NOTES_OR_NONE}}
CURRENT_SPAN: {{PRODUCER_PARAGRAPH}}

Perform:
S1 quick alternative micro-derivation or spot-check,
S2 sanity bounds / physical plausibility,
S3 attempt 1-2 counterexamples,
S4 flag hidden assumptions.

Allowed error_tags: ["counterexample","bounds","assumption","consistency"].
```

AIME-style olympiad mathematics

Producer (`SYSTEM+USER`)

```
[SYSTEM]
You are an AIME problem solver. Produce one clean step-wise derivation and end with:
Final answer: \boxed{<integer 0-999>}

[USER]
PROBLEM:
{{AIME_PROBLEM_TEXT}}
```

Verifier A (Formal & Local) (`USER` only; use shared SYSTEM above)

```
[USER]
TASK: {{AIME_PROBLEM_TEXT}}
ACCEPTED_CONTEXT: {{PRIOR_ACCEPTED_STEPS_OR_NONE}}
CURRENT_SPAN: {{PRODUCER_STEPS}}

Rubric:
R1 algebraic legality (no invalid cancellations, correct radical/log rules),
R2 arithmetic accuracy,
R3 domain/branch/constraints respected and stated,
R4 local entailment from prior state,
R5 notation/variable hygiene,
R6 if final answer present: integer 0-999 and derived quantity matches.

Does the input unit in the right track to achieve the goal given the context? Verify the input unit with your role and task. Only reject the answer according to your role.
The input may not necessarily solve the goal directly as there are more details in the upcoming text. If yes, answer Yes. If no, answer No and provide repair hints beginning with 'Suggestion' to improve in no more than 20 words.
```

Verifier B (Global/Sanity) (`USER` only; use shared SYSTEM above; optional)

```
[USER]
TASK: {{AIME_PROBLEM_TEXT}}
ACCEPTED_CONTEXT: {{PRIOR_ACCEPTED_STEPS_OR_NONE}}
CURRENT_SPAN: {{PRODUCER_STEPS}}

Perform:
S1 quick recompute of the same claim via a different micro-tactic,
S2 sanity bounds/mod checks (e.g., parity, sign, rough magnitude),
```

```
S3 try 1-2 simple counterexamples consistent with constraints,
S4 flag hidden assumptions (e.g., x != 0).

Allowed error_tags: ["counterexample","bounds","assumption","consistency"].
```

LIVENBENCH (REASONING)

Producer (SYSTEM+USER)

```
[SYSTEM]
You are an problem solver. Solve the problem step by step seperated by period. This problem is guaranteed to have a solution. Should not exit without finding the exact solution.
The final answer is wrapped as:\n<solution>...</solution>

[USER]
PROBLEM:
{{LIVEBENCH_PROBLEM_TEXT}}
```

Verifier A (Formal & Local) (USER only; use shared SYSTEM above)

```
[USER]
TASK: {{LIVEBENCH_PROBLEM_TEXT}}
ACCEPTED_CONTEXT: {{PRIOR_ACCEPTED_STEPS_OR_NONE}}
CURRENT_SPAN: {{PRODUCER_STEPS}}

Rubric (apply all):
R1 factual correctness of claims vs. standard definitions/literature,
R2 local logical entailment (no leaps),
R3 unit/notation hygiene,
R4 current answer matches reasoning,
R5 the current answer is not necessarily complete.


Does the input unit in the right track to achieve the goal given the context? Verify the input unit with your role and task. Only reject the answer according to your role.
The input may not necessarily solve the goal directly as there are more details in the upcoming text.
If yes, answer Yes. If no, answer No and provide repair hints beginning with 'Suggestion' that can be concatenated to the context to improve future answer.
The suggestion should no more than 10 words.
```

SVAMP

Producer (SYSTEM+USER)

```
[SYSTEM]
You are an problem solver. Solve the problem step by step seperated by period. This problem is guaranteed to have a solution. Should not exit without finding the exact solution.
The final answer should be in the format: Final answer: \boxed{<integer 0-999>}

[USER]
PROBLEM:
{{SVAMP_PROBLEM_TEXT}}
```

Verifier A (Formal & Local) (`USER` only; use shared SYSTEM above)

```
[USER]
TASK: {{SVAMP_PROBLEM_TEXT}}
ACCEPTED_CONTEXT: {{PRIOR_ACCEPTED_STEPS_OR_NONE}}
CURRENT_SPAN: {{PRODUCER_STEPS}}

R1 algebraic legality (no invalid cancellations, correct radical/log
↪ rules),
R2 arithmetic accuracy,
R3 check carefully for current formulating the real life problem
↪ into math symbols, be very sensitive to negative numbers
R4 check carefully for formulating the real life problem into math
↪ symbols,
R5 check carefully for any unresonable number during the
↪ calculation, which will not happen in real life situation
R6 Check carefully about whether the current answer is overthinking
↪ or too complex to be true in real life situation.
R7 Check carefully about whether the current context model the goal
↪ correctly


Verify the input unit with your role and task. Only reject the
↪ answer according to rubric. Do not reject for other reasons.
The input may not necessarily solve the goal directly as there are
↪ more details in the upcoming text. If yes, answer Yes. If no, answer
↪ No and provide repair hints beginning with 'Suggestion' to improve
↪ in no more than 20 words.
```

GSM8K

Producer (`SYSTEM+USER`)

```
[SYSTEM]
You are an problem solver. Solve the problem step by step seperated
↪ by period. This problem is guaranteed to have a solution. Should not
↪ exit without finding the exact solution.
Final answer: \boxed{<integer 0-999>}

[USER]
PROBLEM:
{{GSM8K_PROBLEM_TEXT}}
```

Verifier A (Formal & Local) (`USER` only; use shared SYSTEM above)

```
[USER]
TASK: {{GSM8K_PROBLEM_TEXT}}
ACCEPTED_CONTEXT: {{PRIOR_ACCEPTED_STEPS_OR_NONE}}
CURRENT_SPAN: {{PRODUCER_STEPS}}

R1 algebraic legality (no invalid cancellations, correct radical/log
↪ rules),
R2 arithmetic accuracy,
R3 check carefully for current formulating the real life problem
↪ into math symbols, be very sensitive to negative numbers
R4 check carefully for formulating the real life problem into math
↪ symbols,
R5 check carefully for any unresonable number during the
↪ calculation, which will not happen in real life situation
R6 Check carefully about whether the current answer is overthinking
↪ or too complex to be true in real life situation.
```

```
R7 Check carefully about whether the current context model the goal
↪ correctly


Verify the input unit with your role and task. Only reject the
↪ answer according to rubric. Do not reject for other reasons.
The input may not necessarily solve the goal directly as there are
↪ more details in the upcoming text. If yes, answer Yes. If no, answer
↪ No and provide repair hints beginning with 'Suggestion' to improve
↪ in no more than 20 words.
```

A.9 USE OF LARGE LANGUAGE MODEL (LLM)

We used an LLM strictly for editorial assistance at the final drafting stage. Specifically, the LLM was used only to check for grammatical errors, fix typographical mistakes, and enhance sentence transitions for better readability. All technical content, including ideas, algorithms, proofs/derivations, experiments, analyses, tables/figures, and conclusions, was authored by the listed authors. The LLM did not generate, rewrite, or materially alter any scientific claims or results.

All LLM-suggested edits were manually reviewed and accepted or rejected by the authors. We retained authorship control at all times and ensured that no technical meaning was changed. The LLM is not an author and bears no responsibility for the paper's content. The authors assume full accountability for all claims and results.