THINKMID: THINKING AND CORRECTING IN THE MIDDLE OF RESPONSE GENERATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Large Reasoning Models (LRMs) scale at test time by spending more computation on a problem, yet existing methods largely decide how much to think before answering, rather than when to think again while answering. We introduce *ThinkMid*, a training-free decoding framework that dynamically allocates additional reasoning during response generation. ThinkMid preserves the model's original reasoning trace, generates the response incrementally, and monitors sentence-level uncertainty using token entropy. When a sentence is sufficiently uncertain, generation pauses and the same model performs an auxiliary self-review in the existing conversational context. If an error is identified, a further reasoning turn produces a local correction; the flagged segment is replaced, and generation resumes from the revised prefix. ThinkMid requires no external verifier or additional training and is complementary to existing test-time scaling strategies. Across mathematical, scientific, and expert-level reasoning benchmarks using two LRMs with different architectures and reasoning formats, ThinkMid improves over the single-solve baseline and compute-matched test-time scaling methods while invoking additional reasoning for only a small fraction of generated sentences. These results indicate that where additional computation is inserted during generation can be as important as how much computation is used.

1 INTRODUCTION

Large Reasoning Models (LRMs), such as OpenAI o1, DeepSeek-R1, Qwen3 and gpt-oss (OpenAI, 2024; DeepSeek-AI et al., 2025; Yang et al., 2025a; OpenAI, 2025), have demonstrated strong performance on tasks ranging from competition mathematics and graduate-level science to software engineering. They first *think* through a problem in an internal chain of thought and then *respond* with a solution, and their accuracy improves with additional inference-time computation (Snell et al., 2024). Existing test-time scaling strategies allocate this computation in three ways (Figure 1(a)–(c)): *sequential* scaling extends the thinking before answering (Muennighoff et al., 2025); *parallel* scaling generates several candidate solutions and selects one by voting or with a verifier (Wang et al., 2023; Cobbe et al., 2021; Lightman et al., 2024); and *iterative* scaling revises a completed attempt through rounds of feedback (Madaan et al., 2023; Shinn et al., 2023).

Despite their differences, these strategies share two limitations. First, the extra computation is not free of risk: forcing the thinking to continue leads models to second-guess correct conclusions, drift or loop, so that accuracy is non-monotonic in thinking length (Chen et al., 2024; Wu et al., 2025; Ghosal et al., 2025; Gema et al., 2025), and every re-generated candidate or rewritten draft is a fresh opportunity to replace a right answer with a wrong one (Huang et al., 2024; Stechly et al., 2024; Kamoi et al., 2024). Second, all of them concentrate deliberation before the response and leave the response itself unchecked: whatever verification the model performs happens before a single response token exists, so an error that arises while writing, a mis-copied value, a dropped case, a fabricated intermediate result, is committed as written.

This matters because errors in generated text *snowball*. In multi-step reasoning, a mistake in an early step propagates through every step that builds on it (Dziri et al., 2023), and once a model has stated a hallucinated claim it tends to commit to it and to produce further hallucinations that keep the text consistent with the first (Zhang et al., 2024; Li et al., 2026). A single unchecked slip in the response therefore rarely stays local: it conditions everything written afterwards and compounds into a wrong

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

final answer. **The question is thus not only how much additional computation to spend, but where in generation to spend it, and how to keep an error, once found, from conditioning what follows.**

This is not how people work through a hard problem (Nelson & Narens, 1990; Thompson et al., 2011): we think first, then write, and while writing we pause where we are unsure, think again, fix what we have written, and continue. In this paper we ask whether an LRM can be given the same ability. ***Can a model decide, while writing its response, where additional thinking is worth spending, and act on that decision in place?***

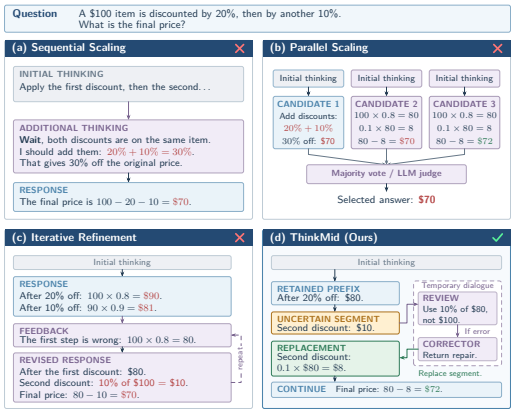


Figure 1: Comparison of test-time scaling strategies.

To this end, we propose *ThinkMid*, a training-free decoding procedure that adds dynamic, mid-response thinking to an LRM (Figure 1(d)). ThinkMid leaves the original thinking untouched, generates the response segment by segment, and separates two decisions. The first is *when* to think again: the mean token entropy of a segment, a free by-product of decoding that marks the critical junctions of a reasoning trajectory (Wang et al., 2025; Cheng et al., 2025), decides whether the model pauses. Entropy serves only as a trigger, not as an error detector: it indicates where the model hesitates, not whether it is wrong. The second is *whether* anything is wrong, and this is left to the model itself: in a second thinking turn that continues the same conversation, it reviews its response so far, and only a confirmed error leads to a correction. Because a hallucinated claim left in context begets further hallucinations, the flagged segment is then rolled back and replaced, so that decoding resumes conditioned on the correction alone; confident segments pass at no extra cost. ThinkMid needs no external verifier, reward model or training; it is orthogonal to the three existing families and composes with them, since the thinking phase and the number of solves are unchanged; and it applies to any model with a thinking phase, including Harmony-style channel formats (OpenAI, 2025).

Extensive experiments on AIME 2025, GPQA Diamond and Humanity's Last Exam with two LRMs of different architectures and thinking formats (Qwen3.6-35B-A3B and GPT-OSS-20B) show that ThinkMid improves over the single solve it extends on every benchmark and model and outperforms thinking forcing, best-of-N and Self-Refine when each baseline is allocated the tokens ThinkMid spent on the same problem: accuracy rises by 7.8% and 7.3% on average, by up to 13.4% on AIME 2025, and two- to three-fold on Humanity's Last Exam. Placing reviews by uncertainty matches periodic review at 26% fewer tokens and beats random placement by 6.6% at equal cost, and a paired analysis of individual corrections shows that they rescue failing trajectories without breaking correct ones. **Together, these results indicate that where additional computation is inserted during generation can matter as much as how much is used.** ThinkMid offers a plug-and-play way to make LRMs think dynamically at test time, composable with existing scaling strategies.

2 Related Work

Test-Time Computation Scaling and Allocation. Test-time scaling improves the reasoning of LRMs (OpenAI, 2024; Guo et al., 2025; Snell et al., 2024) along three families: *sequential* scaling lengthens the thinking (Muennighoff et al., 2025; Aggarwal & Welleck, 2025), *parallel* scaling draws many solves and selects among them (Wang et al., 2023; Lightman et al., 2024), and *iterative* scaling critiques and rewrites a finished response (Madaan et al., 2023; Shinn et al., 2023). A second line allocates compute adaptively from the model's confidence, stopping sampling, exiting thinking early or pruning traces (Aggarwal et al., 2023; Fu et al., 2024; Yang et al., 2025b; Fu et al., 2025; Ma et al., 2025). All of these stay within the think-then-answer paradigm and use confidence *subtractively*, to decide how long to think or how many solves to run. ThinkMid is orthogonal to both: it leaves the thinking phase and the number of solves unchanged and *adds* thinking inside the response where the model is uncertain.

Self-Verification and Self-Correction. Prompting LLMs to critique their own outputs (Weng et al., 2023; Dhuliawala et al., 2023; Madaan et al., 2023) often fails on reasoning without external feedback and flips correct answers as often as it repairs wrong ones (Huang et al., 2024; Stechly et al., 2024; Kamoi et al., 2024). Yet the same models carry reliable signals about their own uncertainty, in calibrated token probabilities (Kadavath et al., 2022), in the semantic entropy of their answers (Farquhar et al., 2024) and in recognising when a question needs clarification (Kuhn et al., 2023a), which suggests that the failure lies in invoking correction everywhere rather than in a lack of self-knowledge. ThinkMid makes correction *selective*, triggered by decoding-time uncertainty; keeps the verifier in the *same conversation*, with access to the model's private thinking; and replaces only the flagged sentence instead of regenerating the response.

Metacognition and Reasoning Mechanisms in LLMs. Human problem solving couples a monitoring process, which tracks how well the reasoning is going, with a control process that decides when to pause or re-check (Flavell, 1979; Nelson & Narens, 1990). LLMs show analogous monitoring, knowing to a useful degree what they know (Kadavath et al., 2022; Yin et al., 2023) and exposing uncertainty through entropies and hidden states (Kuhn et al., 2023b; Zhang et al., 2025), and they can be prompted to act on it (Kuhn et al., 2023a; Didolkar et al., 2024); RL-trained LRMs further exhibit verification and backtracking inside their thinking (Guo et al., 2025; Gandhi et al., 2025). In all of this work such behaviour is confined to the thinking phase and the response is a transcription. ThinkMid couples monitoring and control *during* the response: segment entropy monitors how sure the model is of what it has just written, and a positive signal opens a review turn in which the model thinks again. An extended discussion is given in Appendix A.

3 THINKMID

Our framework allocates additional thinking during response generation through uncertainty monitoring, temporary self-review, and in-place revision of the response. The original thinking phase is left unchanged, and the same fixed model performs all three stages.

3.1 PROBLEM SETUP AND OVERVIEW

Let f_θ be a reasoning model with fixed parameters, and let x denote the task together with its instruction context. Under the standard think-then-answer paradigm the model first generates a thinking trace h and then a response y, each token conditioned on $(x, h, y_{<t})$. Once h is closed, the response is produced in a single pass with no further opportunity to think. We study how to allocate additional inference computation *while y is being written*, so that intermediate content can be checked and revised before further steps come to depend on it.

ThinkMid becomes active after the thinking phase and keeps h fixed throughout; it requires no training, no parameter updates and no external verifier. The response is segmented incrementally as it is decoded, and every completed segment is scored by an uncertainty estimator. When a segment is generated under high uncertainty, generation pauses and a temporary turn asks the model to review its response so far (*self-verification*). Only a confirmed error opens a second temporary turn that produces corrective text (*correction*). The corrective text replaces the flagged segment and generation resumes from the revised prefix; otherwise the response is kept unchanged. This design differs from existing test-time scaling in three respects: the original thinking is untouched, so the additional thinking is inserted inside the response and ThinkMid composes with sequential, parallel and iterative scaling; computation is triggered by the model's own uncertainty rather than allocated uniformly over the trajectory; and correction is local, replacing one segment rather than regenerating the response. The decision to spend computation, made from uncertainty, is thus separated from the decision to change the response, made by the model's own review. Figure 2 illustrates one intervention cycle and Algorithm 1 summarizes the procedure.

3.2 UNCERTAINTY-GUIDED INTERVENTION

Segmentation. The unit of intervention is a segment of the response. An incremental splitter closes a segment at each sentence boundary while treating mathematical expressions, equations and other non-prose structures as indivisible, so that a segment corresponds to a complete claim rather than a typographic fragment. A segment is the smallest unit that can be verified on its own and, if wrong,

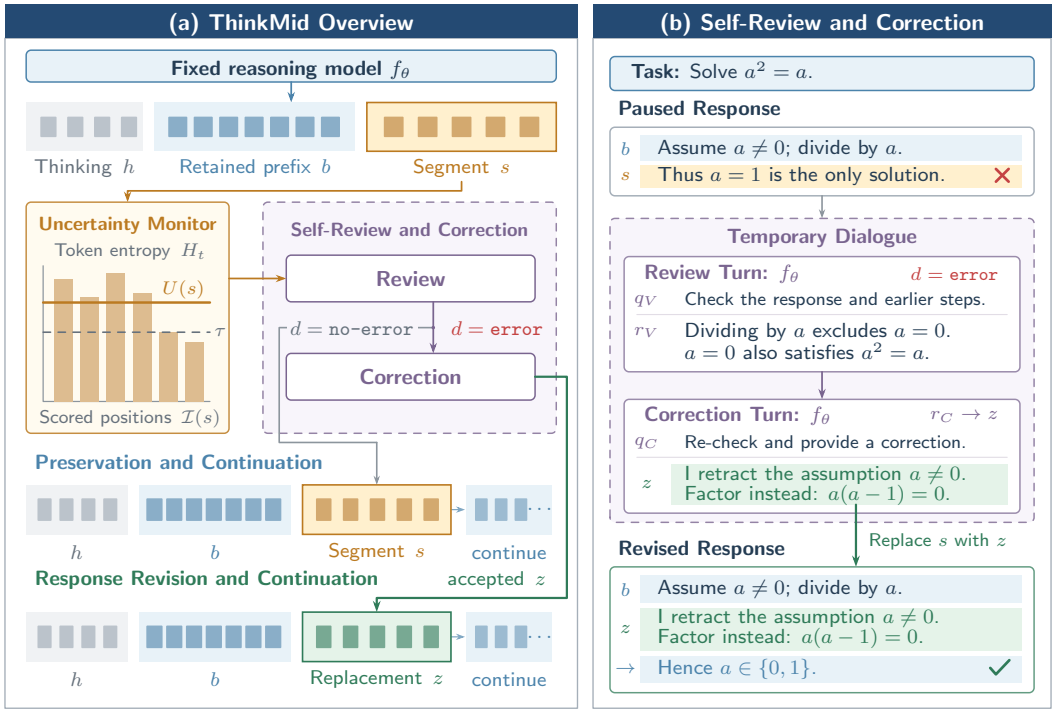


Figure 2: Overview of ThinkMid. After the thinking phase, the response is generated segment by segment while token entropy is monitored. An uncertain segment pauses generation and opens a temporary review turn; a confirmed error opens a correction turn whose output replaces the flagged segment, after which generation resumes from the revised prefix. The temporary turns are discarded; only the corrective text is transferred into the response.

replaced without disturbing the rest of the response; it is also the unit at which a human writer would naturally pause.

Uncertainty Estimator. ThinkMid is designed to operate with any estimator that assigns a scalar uncertainty to a completed segment, and treats the estimator as a black box. In this work we instantiate it with the mean predictive entropy of the tokens in the segment. For every response token we record the entropy of the model's next-token distribution: given the logits ℓ_t over the vocabulary $\mathcal{V}$,

$$p_t(v) = \mathrm{softmax}(\ell_t)_v, \qquad H_t = -\sum_{v \in \mathcal{V}} p_t(v) \log p_t(v), \tag{1}$$

computed from the full distribution before temperature scaling and truncation, so that it reflects how uncertain the model is at position t rather than how probable the sampled token happened to be. The uncertainty of a segment s is the average over its tokens,

$$U(s) = \frac{1}{|s|} \sum_{t \in s} H_t, \tag{2}$$

where $|s|$ is the number of tokens in s, and a segment triggers an intervention when its uncertainty exceeds a threshold τ:

$$\mathsf{Trigger}(s) = \mathbb{1}\big[\, U(s) \geq \tau \,\big]. \tag{3}$$

We adopt mean token entropy for three reasons. (1) It is free: the logits are already produced during decoding, so the score costs no additional forward passes, no repeated sampling and no auxiliary model, and it is available the moment a segment is completed, which is exactly when the decision to pause must be made. (2) It requires no training and no external resources, so the same instantiation applies zero-shot to new models, domains and thinking formats. (3) Averaging over the segment matches the unit of intervention: a single high-entropy token often marks a lexical fork with several

acceptable continuations, whereas a whole claim generated under elevated entropy signals a step the model is not sure about, which is what a review should target. This choice is also supported by evidence that a small minority of high-entropy tokens marks the forks at which reasoning trajectories diverge (Wang et al., 2025; Cheng et al., 2025).

We acknowledge that uncertainty estimation for LLMs is an active research area and that many alternative estimators, such as semantic entropy over sampled continuations (Kuhn et al., 2023b; Farquhar et al., 2024), probes on hidden states (Azaria & Mitchell, 2023; Zhang et al., 2025), verbalised confidence (Tian et al., 2023), agreement across samples (Wang et al., 2023) or learned process verifiers (Lightman et al., 2024), could be integrated into ThinkMid in place of Eq. 2, at different points on the trade-off between cost and detection quality. We leave this exploration to future work, as our current choice already yields consistent gains over compute-matched baselines, as shown in Sec. 4. The hyperparameters of ThinkMid are listed in Appendix C.5.

Algorithm 1 ThinkMid

Input: fixed model f_θ, task x, threshold τ.

Generate thinking h; initialize response $y \leftarrow \epsilon$ and generation state K.
while generation has not ended **do**
  Decode tokens, recording H_t, until a segment s is complete; write $y = b \oplus s$.
  if $U(s) < \tau$ **then** accept s; **continue**.
  On a copy of K, generate the review r_V with verdict d.
  $z \leftarrow \varnothing$; **if** $d = \texttt{error}$ **then** generate the corrective text z in the same temporary conversation.
  if $z \neq \varnothing$ **then**
    Restore K to the end of b; prefill z; set $y \leftarrow b \oplus z$; resume decoding after z.
  else keep y and K unchanged; accept s.
  Discard the temporary conversation.
return y.

3.3 Self-Review and Correction

Review. Let $\mathcal{D}$ denote the conversation at the pause point: the task x, the thinking h and the response generated so far. We preserve the live generation state and open a temporary continuation of $\mathcal{D}$ with a review turn. Its instruction $q_V(s)$ asks the same model, now acting as a *verifier*, to check the response it has written so far against the task, to verify a suspected error independently before asserting it, and to report a verdict $d \in \{\texttt{error}, \texttt{no-error}, \texttt{uncertain}\}$ together with a brief assessment r_V. Uncertainty is presented to the verifier as a reason to inspect, not as evidence of a mistake; a segment is changed only if the model, on reflection, confirms that it is wrong.

Correction. Only $d = \texttt{error}$ opens a correction turn. Its instruction $q_C(s)$ asks the model, now acting as a *corrector*, to re-check the diagnosis and to produce corrective text z that will replace the flagged segment, or to decline if it cannot substantiate a repair. Formally,

$$r_V \sim f_\theta(\cdot \mid \mathcal{D} \oplus q_V(s)), \tag{4}$$

$$r_C \sim f_\theta(\cdot \mid \mathcal{D} \oplus q_V(s) \oplus r_V \oplus q_C(s)), \tag{5}$$

where $\oplus$ appends a conversation turn. Both turns are ordinary assistant turns of the same model and may think under their own budgets, so the additional thinking that ThinkMid adds lives entirely in these turns. Because they continue the same conversation, they reuse its cached context, and the verifier sees the original thinking h as well as the response, a richer context than any external verifier would have.

We deliberately split verification from correction rather than asking the model to rewrite an uncertain segment directly. Rewriting every flagged segment would expose correct content to unnecessary change, and prior work shows that unconditional self-correction flips right answers to wrong ones as often as it repairs them. Asking first *whether* something is wrong, with a free `uncertain` option, and only then *what* the correction is, lets the model decline to act when it has nothing better to offer.

The corrective text z must be self-contained: it carries the corrected claims, the intermediate results needed for the response to continue, and, if the error originates earlier in the retained response, an explicit retraction of the earlier statement. This is necessary because the review and correction turns are discarded afterwards and only z is transferred into the response. The instructions are given in Appendix B.

3.4 Response Revision and Continuation

Write the live response at an intervention as $y = b \oplus s$, where b is the retained prefix and s the flagged segment, with $\oplus$ now denoting text concatenation. For corrective text z the update is

$$b \oplus s \longrightarrow b \oplus z, \qquad y_{\text{next}} \sim f_\theta(\cdot \mid x, h, b \oplus z). \tag{6}$$

The flagged segment is removed and generation resumes conditioned on the correction, so that everything downstream is consistent with it. The thinking h and the retained prefix b remain unchanged. Although the edit is local, a correction that retracts an earlier premise can redirect the rest of the solution, which is how a single intervention can change the final answer.

To realize the update, ThinkMid restores a cached generation state at the segment boundary and prefills z, rather than truncating the state per position. The mechanism therefore applies unchanged to models whose recurrent state cannot be sliced, and to any chat format that delimits a thinking phase. Uncertainty monitoring restarts on the newly generated tokens. If the verifier finds no error, or the corrector declines, the temporary turns are discarded and the live state is left intact, so an intervention that produces no correction leaves the response exactly as it would have been without ThinkMid.

ThinkMid requires only access to the next-token distribution, a multi-turn chat format and a restorable generation state, all of which standard reasoning models provide. Its cost comprises the thinking and outputs of the review and correction turns, the discarded text and the prefix processing of the instructions; we account for all of these when comparing against baselines in Sec. 4.

4 Experiments

4.1 Experimental Setup

Benchmarks. We evaluate on three benchmarks of increasing breadth: AIME 2025, 30 competition mathematics problems with integer answers (Zhang & Math-AI, 2025); GPQA Diamond, 198 graduate-level multiple-choice science questions (Rein et al., 2024); and Humanity's Last Exam (HLE), expert-level questions across academic disciplines (Center for AI Safety et al., 2026), for which we use a fixed, category-stratified sample of 100 questions from its text-only subset rather than the full set, owing to its cost. Details and grading protocols are given in Appendix C.1.

Models and Baselines. We use two reasoning models, Qwen3.6-35B-A3B (Qwen Team, 2026) and GPT-OSS-20B (OpenAI, 2025), and compare ThinkMid with five baselines on each: **Vanilla**, a single response after the initial thinking; **Thinking Forcing**, s1-style budget forcing that extends the initial thinking (Muennighoff et al., 2025); **BoN-Judge** and **BoN-Vote**, best-of-N with the candidate chosen by an LLM judge or by majority vote (Wang et al., 2023); and **Self-Refine**, iterative feedback and revision of a completed draft (Madaan et al., 2023). Together they instantiate sequential, parallel and iterative scaling on the same backbones (Appendix C.2).

Evaluation Metrics and Budget Accounting. We report **pass@1 accuracy** and mean **total token consumption (Tokens)** per evaluated attempt, averaging over $R_D = 4$ attempts per problem on AIME 2025 and one attempt on the other benchmarks. Tokens counts every prompt, thinking and response token of every inference call a method makes, including candidates, feedback rounds, retries and text later discarded by rollback, under one accounting convention for all methods. Compute matching is per attempt: every non-Vanilla baseline receives ThinkMid's realized consumption on the same problem and run as its nominal budget, $B_{x,r} = T_{\text{ThinkMid},x,r}$, while the reported Tokens are actual consumption, including any overrun of the completion policy (Appendix C.3).

Experimental settings. All models use the officially recommended decoding settings (temperature 0.6, top-$k = 20$, top-$p = 0.95$), seed 42, and a single NVIDIA A100-80GB GPU. Vanilla and ThinkMid share a maximum initial-thinking budget of $16{,}000$ tokens, and ThinkMid uses a fixed entropy threshold $\tau = 0.5$ for all models and benchmarks. Each budget-matched baseline spends its per-problem allocation according to its own mechanism: Thinking Forcing on longer initial thinking, BoN on additional candidates (so N varies across problems, with BoN-Judge reserving tokens for the judge), and Self-Refine on feedback and revision rounds; when the allocation is exhausted, the current response is allowed to finish and no further round is started. The complete budget rules and stopping policy are given in Appendix C.4.

4.2 MAIN RESULTS

Table 1: Main results across three benchmarks and two reasoning models. Pass@1 is accuracy (%; ↑); Tokens is mean actual consumption per evaluated attempt (↓). Bold and underlined values indicate the highest and second-highest available results, respectively.

Methods	AIME 2025		GPQA Diamond		HLE	
	Pass@1↑	Tokens↓	Pass@1↑	Tokens↓	Pass@1↑	Tokens↓
Qwen3.6-35B-A3B						
Vanilla	70.0	**16,919**	83.3	**12,961**	9.0	**15,121**
Thinking Forcing	<u>80.0</u>	26,406	81.8	18,791	<u>17.0</u>	27,641
BoN-Judge	71.7	26,840	<u>84.3</u>	18,534	12.0	<u>26,561</u>
BoN-Vote	73.3	26,486	83.3	<u>18,122</u>	8.0	26,776
Self-Refine	76.7	28,548	**85.4**	19,102	13.0	32,071
ThinkMid (ours)	**83.3**	<u>26,366</u>	<u>84.3</u>	18,963	**18.0**	27,401
GPT-OSS-20B						
Vanilla	63.3	**16,758**	<u>67.7</u>	**9,718**	4.0	**14,172**
Thinking Forcing	**76.7**	<u>26,825</u>	66.7	<u>18,112</u>	<u>10.0</u>	30,851
BoN-Judge	<u>70.0</u>	26,914	<u>67.7</u>	18,321	5.0	30,479
BoN-Vote	<u>70.0</u>	26,904	66.7	18,348	6.0	<u>29,479</u>
Self-Refine	66.7	27,371	66.2	18,796	6.0	31,001
ThinkMid (ours)	**76.7**	26,935	**68.2**	18,481	**12.0**	30,273

ThinkMid achieves the highest accuracy at matched compute. Table 1 reports Pass@1 and mean token consumption for the vanilla single-pass solve, the four compute-matched baselines and ThinkMid. Because every scaled method is allocated the per-problem budget that ThinkMid spent, the scaled methods consume similar token counts on each benchmark (26.4k to 28.5k on AIME with Qwen3.6), so accuracy differences reflect how the compute is used rather than how much is available. Under this condition ThinkMid is the best or tied-best method in five of the six model–benchmark cells and second in the sixth (GPQA with Qwen3.6, 84.3% against 85.4% for Self-Refine), and it is the only method that improves over the vanilla solve on every benchmark and model, by 7.8% and 7.3% on average for the two models.

Gains are largest on tasks with long derivations. Over the vanilla solve, ThinkMid gains 13.3% and 13.4% on AIME and 9.0% and 8.0% on HLE, doubling and tripling accuracy there, but only 1.0% and 0.5% on GPQA, at 1.6 to 1.9 times the vanilla token consumption. AIME and HLE solutions are long derivations in which an intermediate slip propagates to the final answer, exactly what a mid-response review can catch, whereas GPQA questions hinge on a single piece of knowledge, and no method gains more than 2.1% there.

ThinkMid outperforms all compute-matched baselines. Thinking forcing, the strongest baseline, trails ThinkMid on five of six cells and lowers GPQA accuracy below the vanilla solve on both models (1.5% and 1.0%), consistent with reports that forced continuations lead the model to overturn conclusions that were correct (Chen et al., 2024; Wu et al., 2025; Ghosal et al., 2025; Gema et al., 2025). Best-of-N admits only a few extra samples under the budget and neither selection rule turns them into reliable gains, and Self-Refine is the most expensive and least stable method, gaining 6.7% on AIME with Qwen3.6 but losing 1.5% on GPQA with GPT-OSS-20B.

The gains hold across architectures and thinking formats. Qwen3.6-35B-A3B, a hybrid mixture-of-experts model with gated-DeltaNet layers and `<think>` delimiters (Yang et al., 2025a;c), and GPT-OSS-20B, an attention-only mixture-of-experts model that reasons in Harmony channels (OpenAI, 2025), improve by similar margins and preserve the same ranking against the baselines, indicating that mid-response thinking exploits a property of reasoning models in general rather than of one architecture.

4.3 Ablation Study

Uncertainty-guided placement gives the best accuracy–cost trade-off. We ablate *where* reviews are placed on AIME 2025 with Qwen3.6-35B-A3B. All variants share every other component of ThinkMid and differ only in the positions at which a review is opened: **Entropy-Guided** after a segment whose mean entropy exceeds $\tau = 0.5$ (ThinkMid's criterion), **Random** after randomly chosen sentences, and **Fixed-Interval** at the first sentence boundary after every 200 response tokens. Entropy-Guided and Random happen to open the same number of reviews (2.2 per problem), so their comparison isolates the effect of position; Fixed-Interval opens 3.83, since under a periodic rule the count follows from the response length and the interval, and stretching the interval to match would only coarsen its grid, so we let it review more often, which favors it. Even so, entropy-guided placement matches its 83.3% with 43% fewer reviews and 25.6% fewer tokens, and with the same number of reviews it exceeds random placement by 6.6% at only 2.9% more tokens (Table 2): uncertainty selects the positions at which a review pays off, which periodic placement reaches only by reviewing far more often and random placement often misses.

Table 2: Review-placement ablation on AIME 2025 with Qwen3.6-35B-A3B.

Review Placement	Pass@1↑	Tokens↓
Random Sentences	76.7	**25,630**
Fixed Interval	**83.3**	35,447
Entropy-Guided (ours)	**83.3**	26,366

Table 3: ThinkMid accuracy at accepted replacement points.

Subset	Correct/Total	Accuracy
All Points	9/11	**81.8%**
Continuation Incorrect	5/7	**71.4%**

4.4 Hyperparameter Sensitivity Analysis

Accuracy is stable over a wide range of τ, and the default sits at the knee of the cost curve. Varying $\tau \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$ on AIME 2025 with Qwen3.6-35B-A3B (Figure 3), accuracy stays at 83.3% from $\tau = 0.1$ to 0.5 while mean token consumption halves, from 51,143 to 26,366; above 0.5, consumption falls only to 24,749 and 22,475 but accuracy drops to 81.7% and 80.0% as genuinely uncertain segments go unreviewed. The default $\tau = 0.5$ is thus the cheapest threshold attaining the highest accuracy, and the flat plateau below it means the threshold needs no fine tuning: a conservative choice costs tokens, not accuracy.

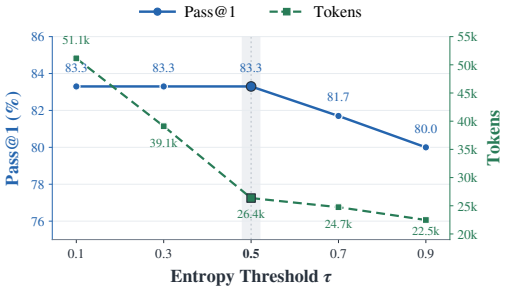


Figure 3: Entropy-threshold sensitivity on AIME 2025 with Qwen3.6-35B-A3B. Left axis: pass@1; right axis: mean token consumption. Shading marks the default $\tau = 0.5$.

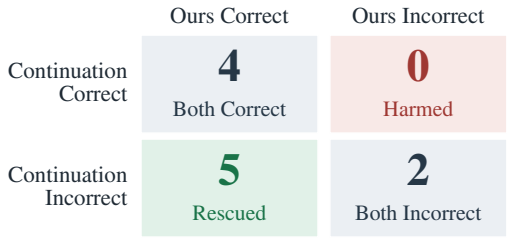


Figure 4: Paired final-answer outcomes at 11 replacement points. Ours denotes ThinkMid; plain continuation disables further review and correction.

5 Analysis and Discussion

To see whether the mechanism behaves as intended, we qualitatively analyse the corrections of one AIME 2025 run with Qwen3.6-35B-A3B, in which 8 of the 30 problems received an accepted correction, 11 in total. At each point we resume the same model from the pre-correction prefix with review and correction disabled (the *plain continuation*) and pair its final answer with ThinkMid's; the two branches share the task, the thinking and the preceding response, including earlier corrections. Individual cases below are cited as supporting evidence; the complete point-by-point details are given in Appendix D.

Corrections rescued failing continuations more often than not, and none broke a correct one. ThinkMid's final answer is correct at 9 of the 11 points, the plain continuation at 4 (Table 3, Figure 4); wherever the plain continuation is correct, so is ThinkMid. Skipping the first intervention of every intervened problem would lower the run's accuracy from 83.3% to 70.0% (25 to 21 problems).

Interventions on substantive reasoning steps were decisive. Seven of the eleven interventions, six at intermediate reasoning steps and one at a wrong answer line, fall where the plain continuation fails, and ThinkMid recovers five of them. All five recoveries occur at intermediate steps that carry reasoning or computation (I-10, I-13, II-14 and twice in II-5, whose two successive corrections together brought the derivation back on track). In the two that were not recovered (I-15's over-counted residue triples and II-10's answer of 869 against 907) the verifier identifies a genuine error, but the corrective text does not resolve it under the configured budget: reliable diagnosis and effective repair are distinct capabilities, and whether a larger correction budget closes this gap is untested.

Where the continuation was already right, the review refined rather than changed the response. At two of the four both-correct points the review was opened at the answer line (I-13 at $\boxed{204}$, II-5 at $\boxed{336}$): the verifier judged the derivation supporting the answer insufficient, and the correction kept the answer while supplementing it with a fuller derivation and justification. At the other two, entropy had landed on segments with little substantive content, a one-line statement in I-11 and a section heading in II-15, so the review cost a turn without changing the outcome. Uncertainty alone is thus not evidence of an error, and screening segments without substantive claims is a natural refinement.

The analysis covers one run, one model and accepted corrections only, and compares continued intervention with intervention-free continuation rather than isolating each correction. Within these limits, the observations are consistent with the mechanism we designed: corrections rarely hurt, are decisive where they land on a wrong reasoning step that later steps depend on, and refine rather than alter what is already right.

6 Conclusions

In conclusion, this work addresses the snowballing of errors in the responses of LRMs by introducing ThinkMid, a general inference-time framework that adds dynamic thinking *during* the response rather than only before it. ThinkMid leaves the model's original thinking untouched, monitors the uncertainty of each response segment as it is written, and, where the model hesitates, lets the same model review the segment and replace it in place once an error is confirmed, so that subsequent generation builds on verified content. The procedure requires no training or external verifier, and it is orthogonal to and composable with sequential, parallel and iterative test-time scaling. Extensive evaluations on mathematical, scientific and expert-level reasoning benchmarks with two reasoning models of different architectures demonstrate that ThinkMid consistently improves over the single solve it extends and outperforms compute-matched scaling baselines, and that uncertainty places the additional thinking where it pays off. These findings highlight that *where* inference-time computation is inserted during generation can matter as much as how much is used, and point to mid-response thinking as a practical paradigm for making reasoning models more reliable.

Limitations and Future Work. We acknowledge two limitations of our method. First, ThinkMid can only repair what the model itself can repair: in the failure cases of our analysis, the verifier correctly located an error, yet the corrector could not produce the right content within its budget, so detection outran repair. Stronger correctors, larger correction budgets, or external tools invoked at the correction turn are natural extensions. Second, mean token entropy is a proxy for hesitation rather than for correctness: it occasionally triggers on segments without substantive claims, and it cannot flag a claim the model is confidently wrong about. Because the estimator is a pluggable component of the framework, semantic entropy, hidden-state probes or learned process verifiers may be substituted to trade cost for better targeting. Nevertheless, significant value remains in the method's ability to catch and correct errors at the point where they arise, before they propagate, and in its compatibility with existing scaling strategies, whose combination with ThinkMid we leave to future work.

AI USE STATEMENT

We used generative AI tools, including ChatGPT, to assist with formalizing and presenting the proposed framework, refining review and correction prompts, discussing experimental design, and interpreting experimental observations. We also used these tools for literature identification and summarization, manuscript drafting and revision, LaTeX formatting, and the development of plotting code and scientific figures. The authors proposed the core intervention mechanism and provided the experimental results used in the analyses and visualizations.

The authors manually reviewed and revised AI-assisted text, checked references and attributed claims against the original sources, and cross-checked numerical statements, tables, and plots against the experimental records. AI-assisted plotting code was inspected and executed, and the resulting figures were checked for consistency with the underlying data. Method descriptions and pseudocode were checked against the experimental implementation. The authors retain responsibility for all research decisions and for the accuracy, originality, and integrity of the final submission, including content developed with AI assistance.

ETHICS STATEMENT

Research integrity and responsible conduct guide this work, and we commit to the principles set out in the ICLR Code of Ethics. We report experimental findings transparently and document the evaluation procedures, token accounting, and budget allocation underlying our comparisons. Our use and any redistribution of existing benchmarks and pretrained models remain subject to their respective licenses and terms of use. We acknowledge the contributions of prior research, benchmark creators, and model developers through appropriate citations. Although ThinkMid aims to improve reasoning reliability, self-review and correction can still produce errors. We therefore encourage responsible deployment and careful assessment of potential downstream risks.

REPRODUCIBILITY STATEMENT

We provide the source code for ThinkMid to support independent reproduction and verification of our results. The proposed algorithm and experimental setup are described in Sections 3 and 4.1. The appendices document implementation details, review and correction prompts, benchmark selection and grading, baseline configurations, hyperparameters, token accounting, and stopping policies.

REFERENCES

Pranjal Aggarwal and Sean Welleck. L1: Controlling how long a reasoning model thinks with reinforcement learning. *arXiv preprint arXiv:2503.04697*, 2025.

Pranjal Aggarwal et al. Let's sample step by step: Adaptive-consistency for efficient reasoning and coding with LLMs. In *EMNLP*, 2023.

Amos Azaria and Tom Mitchell. The internal state of an LLM knows when it's lying. In *Findings of EMNLP*, 2023.

Center for AI Safety, Scale AI, and HLE Contributors Consortium. A benchmark of expert-level academic questions to assess AI capabilities. *Nature*, 649:1139–1146, 2026. doi: 10.1038/s41586-025-09962-4. URL https://www.nature.com/articles/s41586-025-09962-4.

Xingyu Chen et al. Do NOT think that much for 2+3=? on the overthinking of o1-like LLMs. *arXiv preprint arXiv:2412.21187*, 2024.

Daixuan Cheng et al. Reasoning with exploration: An entropy perspective. *arXiv preprint arXiv:2506.14758*, 2025.

Karl Cobbe et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

DeepSeek-AI et al. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. arXiv preprint arXiv:2501.12948, 2025. URL https://arxiv.org/abs/2501.12948.

Shehzaad Dhuliawala et al. Chain-of-verification reduces hallucination in large language models. *arXiv preprint arXiv:2309.11495*, 2023.

Aniket Didolkar et al. Metacognitive capabilities of LLMs: An exploration in mathematical problem solving. In *NeurIPS*, 2024.

Nouha Dziri et al. Faith and fate: Limits of transformers on compositionality. In *NeurIPS*, 2023.

Sebastian Farquhar, Jannik Kossen, Lorenz Kuhn, and Yarin Gal. Detecting hallucinations in large language models using semantic entropy. *Nature*, 630:625–630, 2024.

John H. Flavell. Metacognition and cognitive monitoring: A new area of cognitive-developmental inquiry. *American Psychologist*, 34(10):906–911, 1979.

Yichao Fu et al. Efficiently serving LLM reasoning programs with Certaindex. *arXiv preprint arXiv:2412.20993*, 2024.

Yichao Fu et al. Deep think with confidence. *arXiv preprint arXiv:2508.15260*, 2025.

Kanishk Gandhi et al. Cognitive behaviors that enable self-improving reasoners, or, four habits of highly effective STaRs. *arXiv preprint arXiv:2503.01307*, 2025.

Aryo Pradipta Gema et al. Inverse scaling in test-time compute. *arXiv preprint arXiv:2507.14417*, 2025.

Soumya Suvra Ghosal et al. Does thinking more always help? understanding test-time scaling in reasoning models. *arXiv preprint arXiv:2506.04210*, 2025.

Xinyu Guan et al. rStar-Math: Small LLMs can master math reasoning with self-evolved deep thinking. *arXiv preprint arXiv:2501.04519*, 2025.

Daya Guo et al. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Jie Huang et al. Large language models cannot self-correct reasoning yet. In *ICLR*, 2024.

Saurav Kadavath et al. Language models (mostly) know what they know. *arXiv preprint arXiv:2207.05221*, 2022.

Ryo Kamoi et al. When can LLMs actually correct their own mistakes? a critical survey of self-correction of LLMs. *Transactions of the Association for Computational Linguistics*, 2024.

Muhammad Khalifa et al. GRACE: Discriminator-guided chain-of-thought reasoning. In *Findings of EMNLP*, 2023.

Lorenz Kuhn, Yarin Gal, and Sebastian Farquhar. CLAM: Selective clarification for ambiguous questions with generative language models. *arXiv preprint arXiv:2212.07769*, 2023a.

Lorenz Kuhn, Yarin Gal, and Sebastian Farquhar. Semantic uncertainty: Linguistic invariances for uncertainty estimation in natural language generation. In *ICLR*, 2023b.

Aviral Kumar et al. Training language models to self-correct via reinforcement learning. *arXiv preprint arXiv:2409.12917*, 2024.

Lin Li, Georgia Channing, Suhaas M Bhat, Gabriel Davis Jones, and Yarin Gal. Building reliable long-form generation via hallucination rejection sampling. *arXiv preprint arXiv:2606.03628*, 2026.

Yiwei Li et al. Escape sky-high cost: Early-stopping self-consistency for multi-step reasoning. In *ICLR*, 2024.

Hunter Lightman et al. Let's verify step by step. In *ICLR*, 2024.

Wenjie Ma et al. Reasoning models can be effective without thinking. *arXiv preprint arXiv:2504.09858*, 2025.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-Refine: Iterative refinement with self-feedback. arXiv preprint arXiv:2303.17651, 2023. URL https://arxiv.org/abs/2303.17651.

Nat McAleese et al. LLM critics help catch LLM bugs. *arXiv preprint arXiv:2407.00215*, 2024.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 20275–20321. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.emnlp-main.1025. URL https://aclanthology.org/2025.emnlp-main.1025/.

Thomas O. Nelson and Louis Narens. Metamemory: A theoretical framework and new findings. In *Psychology of Learning and Motivation*, volume 26, pp. 125–173. Academic Press, 1990. doi: 10.1016/S0079-7421(08)60053-5. URL https://doi.org/10.1016/S0079-7421(08)60053-5.

OpenAI. Learning to reason with LLMs. https://openai.com/index/learning-to-reason-with-llms/, 2024.

OpenAI. gpt-oss-120b & gpt-oss-20b model card. arXiv preprint arXiv:2508.10925, 2025. URL https://arxiv.org/abs/2508.10925.

Qwen Team. Qwen3.6-35B-A3B: Agentic coding power, now open to all, April 2026. URL https://qwen.ai/blog?id=qwen3.6-35b-a3b.

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. URL https://openreview.net/forum?id=Ti67584b98.

William Saunders et al. Self-critiquing models for assisting human evaluators. *arXiv preprint arXiv:2206.05802*, 2022.

Noah Shinn et al. Reflexion: Language agents with verbal reinforcement learning. In *NeurIPS*, 2023.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

Kaya Stechly, Karthik Valmeekam, and Subbarao Kambhampati. On the self-verification limitations of large language models on reasoning and planning tasks. *arXiv preprint arXiv:2402.08115*, 2024.

Mark Steyvers et al. What large language models know and what people think they know. *Nature Machine Intelligence*, 7:221–231, 2025.

Hanshi Sun et al. Fast best-of-n decoding via speculative rejection. *arXiv preprint arXiv:2410.20290*, 2024.

Valerie A. Thompson, Jamie A. Prowse Turner, and Gordon Pennycook. Intuition, reason, and metacognition. *Cognitive Psychology*, 63(3):107–140, 2011. doi: 10.1016/j.cogpsych.2011.06.001. URL https://doi.org/10.1016/j.cogpsych.2011.06.001.

Katherine Tian et al. Just ask for calibration: Strategies for eliciting calibrated confidence scores from language models fine-tuned with human feedback. In *EMNLP*, 2023.

Shenzhi Wang et al. Beyond the 80/20 rule: High-entropy minority tokens drive effective reinforcement learning for LLM reasoning. *arXiv preprint arXiv:2506.01939*, 2025.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2203.11171.

Yuqing Wang and Yun Zhao. Metacognitive prompting improves understanding in large language models. In *NAACL*, 2024.

Jason Wei et al. Chain-of-thought prompting elicits reasoning in large language models. In *NeurIPS*, 2022.

Sean Welleck, Ximing Lu, Peter West, Faeze Brahman, Tianxiao Shen, Daniel Khashabi, and Yejin Choi. Generating sequences by learning to self-correct. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=hH36JeQZDaO.

Sean Welleck et al. From decoding to meta-generation: Inference-time algorithms for large language models. *Transactions on Machine Learning Research*, 2024.

Yixuan Weng et al. Large language models are better reasoners with self-verification. In *Findings of EMNLP*, 2023.

Yangzhen Wu et al. Inference scaling laws: An empirical analysis of compute-optimal inference for problem-solving with language models. *arXiv preprint arXiv:2408.00724*, 2024.

Yuyang Wu et al. When more is less: Understanding chain-of-thought length in LLMs. *arXiv preprint arXiv:2502.07266*, 2025.

Yuxi Xie et al. Self-evaluation guided beam search for reasoning. In *NeurIPS*, 2023.

An Yang et al. Qwen3 technical report. arXiv preprint arXiv:2505.09388, 2025a. URL https://arxiv.org/abs/2505.09388.

Chenxu Yang et al. Dynamic early exit in reasoning models. *arXiv preprint arXiv:2504.15895*, 2025b.

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving Mamba2 with delta rule. In *ICLR*, 2025c.

Shunyu Yao et al. Tree of thoughts: Deliberate problem solving with large language models. In *NeurIPS*, 2023.

Zhangyue Yin et al. Do large language models know what they don't know? In *Findings of ACL*, 2023.

Anqi Zhang et al. Reasoning models know when they're right: Probing hidden states for self-verification. *arXiv preprint arXiv:2504.05419*, 2025.

Muru Zhang, Ofir Press, William Merrill, Alisa Liu, and Noah A. Smith. How language model hallucinations can snowball. In *ICML*, 2024.

Yifan Zhang and Team Math-AI. American invitational mathematics examination (AIME) 2025, 2025. URL https://huggingface.co/datasets/math-ai/aime25.

A Extended Related Work

Test-Time Computation Scaling and Allocation. Test-time scaling improves the reasoning of LRMs (OpenAI, 2024; Guo et al., 2025; Snell et al., 2024) along three families (Welleck et al., 2024): *sequential* scaling lengthens the thinking by suppressing its end token and appending cue words (Muennighoff et al., 2025) or by training length control (Aggarwal & Welleck, 2025); *parallel* scaling draws many solves and selects by majority vote (Wang et al., 2023), a trained verifier (Cobbe et al., 2021; Lightman et al., 2024) or tree search (Yao et al., 2023; Guan et al., 2025); *iterative* scaling critiques and rewrites a finished response (Madaan et al., 2023; Shinn et al., 2023). Which family wins depends on difficulty and budget (Snell et al., 2024; Wu et al., 2024), so gains must be compared at equal cost, as we do per problem. A second line *allocates* compute adaptively from the model's confidence: stopping sampling when votes converge (Aggarwal et al., 2023; Li et al., 2024), exiting or re-allocating thinking by answer certainty (Fu et al., 2024; Yang et al., 2025b), pruning low-confidence traces (Fu et al., 2025; Sun et al., 2024), or skipping thinking altogether (Ma et al., 2025). All of these stay within the think-then-answer paradigm and use confidence *subtractively*, to decide how long to think or how many solves to run. ThinkMid is orthogonal to both: it leaves the thinking phase and the number of solves unchanged and *adds* thinking inside the response where the model is uncertain.

Self-Verification and Self-Correction. LLMs have been prompted to verify or critique their own outputs (Weng et al., 2023; Dhuliawala et al., 2023; Madaan et al., 2023) and trained as critics (Saunders et al., 2022; McAleese et al., 2024), but intrinsic self-correction often fails on reasoning without external feedback, flips correct answers as often as it repairs wrong ones, and loses its edge once the baseline receives equal compute (Huang et al., 2024; Stechly et al., 2024; Kamoi et al., 2024). This pessimism has itself been qualified: the same models carry reliable signals about their own uncertainty, in the calibration of their token probabilities (Kadavath et al., 2022), in the semantic entropy of their answers, which detects confabulations without supervision (Farquhar et al., 2024), and in their ability to recognise an ambiguous question and ask for clarification instead of answering (Kuhn et al., 2023a). The failure of unconditional self-correction thus lies less in a lack of self-knowledge than in invoking correction everywhere, without a signal of where to look. Remedies train the correction ability (Welleck et al., 2023; Kumar et al., 2024) or guide decoding with step-level verifiers (Khalifa et al., 2023; Xie et al., 2023), at the price of extra training, an external verifier or many candidates per step. ThinkMid differs in three ways: correction is *selective*, triggered by decoding-time uncertainty; the verifier is the *same model in the same conversation*, appended to the live KV cache with full access to its private thinking; and correction is *local*, replacing the flagged sentence with corrective text and resuming decoding rather than regenerating the response.

Metacognition and Reasoning Mechanisms in LLMs. Human problem solving is metacognitive: a monitoring process tracks how well the current reasoning is going, and a control process decides, on that basis, when to pause, re-check or change course (Flavell, 1979; Nelson & Narens, 1990). Whether LLMs have analogous abilities is an active question. On the monitoring side, models know to a useful degree what they know (Kadavath et al., 2022; Yin et al., 2023); their uncertainty can be read from token entropies, from semantic entropy over sampled meanings (Kuhn et al., 2023b; Farquhar et al., 2024) and from hidden states (Azaria & Mitchell, 2023; Zhang et al., 2025), although their confidence is less calibrated than that of humans (Steyvers et al., 2025). On the control side, models can be prompted to act on such signals, asking for clarification when a question is ambiguous (Kuhn et al., 2023a) or reflecting on their own understanding before answering (Wang & Zhao, 2024; Didolkar et al., 2024). Within reasoning traces, a minority of high-entropy tokens marks the forks at which trajectories diverge (Wang et al., 2025; Cheng et al., 2025). Reinforcement learning with verifiable rewards has turned chain-of-thought prompting (Wei et al., 2022) into the long thinking phase of LRMs, in which verification and backtracking emerge without supervision (Guo et al., 2025; Gandhi et al., 2025). In all of this work the model's metacognitive behaviour is confined to the thinking phase and the response is a transcription. ThinkMid couples monitoring and control during the response: the mean entropy of a segment monitors how sure the model is of what it has just written, and a positive signal exercises control by opening a review turn in which the model thinks again.

B PROMPTS

Prompts are given in Figure 5.

C EXPERIMENTAL DETIALS

C.1 DETAILED DESCRIPTION OF BENCHMARKS

AIME 2025. The American Invitational Mathematics Examination is an invitational competition for the strongest participants of the AMC series; we use the 30 problems of the 2025 contests (AIME I and AIME II, 15 problems each) (Zhang & Math-AI, 2025). Every problem has a unique integer answer between 0 and 999, which removes ambiguity from grading, and solutions require multi-step derivations in algebra, combinatorics, geometry and number theory, making the benchmark a standard testbed for long-form mathematical reasoning. A response is graded correct when the last `\boxed{}` answer matches the reference integer exactly. Because the set is small, we run $R = 4$ independent attempts per problem with different seeds and report accuracy over the resulting 120 graded attempts, each weighted equally.

GPQA Diamond. GPQA is a set of graduate-level multiple-choice questions in biology, chemistry and physics written and validated by domain experts, designed to be "Google-proof": experts holding or pursuing a PhD in the field answer correctly most of the time, whereas skilled non-experts with unrestricted web access do not (Rein et al., 2024). We use the Diamond subset, the 198 questions on which both expert validators agree and most non-experts fail. Each question has four options; to remove position bias we shuffle the options deterministically per question with a hash-derived seed, so every method sees the same permutation. A response is graded correct when the extracted option letter equals the gold letter, and a response from which no letter can be extracted counts as wrong. We report accuracy over a single attempt per question.

HLE. Humanity's Last Exam is a benchmark of expert-written questions spanning more than a hundred subjects, constructed so that frontier models score far below expert level (Center for AI Safety et al., 2026). We restrict to the text-only questions, which require no image input, and evaluate a fixed, seed-stratified sample of 100 questions drawn proportionally from the benchmark's subject categories; the sample is frozen so that every method and baseline answers exactly the same questions. Questions are either multiple-choice or short-answer. Multiple-choice items are graded by the extracted option letter. Short-answer items are graded by an answer-type-aware exact-match rule, and an item that this rule marks wrong is re-checked once by an LLM grader (the same model as the solver, decoded greedily) that compares the extracted answer with the reference and may flip the verdict to correct; multiple-choice verdicts and rule-based correct verdicts are final. Every question enters the denominator, and an unparseable answer counts as wrong. We report accuracy over a single attempt per question.

C.2 MODELS AND BASELINES

Models. We use two open-weight reasoning models that expose an explicit thinking phase but differ in architecture and in how that phase is delimited. **Qwen3.6-35B-A3B** (Qwen Team, 2026) is a mixture-of-experts model with 35B total and 3B active parameters whose layers interleave gated-DeltaNet recurrence with attention (Yang et al., 2025c); it marks thinking with `<think>` and `</think>` tags and is run in bf16 with thinking enabled in its chat template. **GPT-OSS-20B** (OpenAI, 2025) is an attention-only mixture-of-experts model with 21B total and 3.6B active parameters that follows the Harmony response format, placing its reasoning in an `analysis` channel and the visible answer in a `final` channel; the channel switch plays the role of the end-of-thinking marker. It is run at `reasoning_effort=high` in its native MXFP4 precision. Both models are served with HuggingFace Transformers through our own token-level generation loop, which is required to read per-token logits, to inject or suppress tokens during decoding and to share a KV cache between the main response and the review and correction turns.

Baselines. For each backbone we compare ThinkMid with five baselines that share the model, the prompt, the chat template, the decoding parameters and the grading code. **Vanilla** generates a

System Prompt

You are an expert mathematical problem solver. Your task is to solve the following math problem. Please reason step by step. You must give a step-by-step solution in your response. Put your final answer inside \boxed{}.

User Prompt

Find the sum of all integer bases b > 9 for which 17_b is a divisor of 97_b.

Review Prompt

You are the assistant answering the original user task. Your response has been paused for self-review.

Act as a rigorous verifier of your own work.

This was the last segment accepted as correct in a previous review:

```
<previous_correct_segment>
{previous_segment_text}
</previous_correct_segment>
```

This segment now triggers review because of uncertainty:

```
<current_segment>
{current_turn_text}
</current_segment>
```

Uncertainty here may indicate an error in the solution or approach you have developed since the previous correct segment. Focus your review on the work after that segment through the current one. Do not automatically reopen previously accepted text; use the dependency rule below. If the previous segment is "(none)", no text has been accepted, so review from the beginning. Assess your response as it stood at the current segment, without relying on later text. Report your assessment without continuing the answer.

Requirements

- Re-read the original task. Independently check the claims, deductions, assumptions, and calculations in the work under review against its givens and requirements. Prior reasoning, reviews, and corrections are fallible.
- Revisit previously accepted text only to check a premise actually used by a substantive claim in the current segment. Report an earlier error only if you quote the specific faulty step, demonstrate its error, and explain how the current claim depends on it. Prior acceptance is not proof of correctness, but do not search accepted text for unrelated errors. If the current segment has no verifiable claim, calculation, or inference, do not use it as a reason to reopen accepted text or report an error there.
- Support objections with checkable evidence. Recompute relevant quantities or check the applicable conditions, counterexamples, code behavior, or boundary cases. Do not invent facts, sources, or test results. A correct answer can have invalid reasoning; establish the faulty reasoning without assuming that the answer itself must be wrong.
- Insufficient justification or another possible interpretation is not, by itself, evidence of an error. Independently supply and check the missing reasoning. Assess interpretations against the task and context; demonstrate why the interpretation actually used is invalid before rejecting it. If a material ambiguity remains unresolved, use "uncertain", not true.
- For a factual or domain-specific assertion, report an error only if you can write the decisive evidence in the "error" field using at least one of: (a) a contradiction with an explicit condition or datum in the task, identifying both the assertion and the conflicting given; (b) a reproducible calculation, with the formula, inputs, decisive steps, and result showing the discrepancy; or (c) an item-by-item structural argument, stating the applicable criteria, checking the relevant components or cases against the given facts, and showing exactly where the assertion fails. A mismatch with your remembered domain knowledge is not sufficient. Merely naming a principle, invoking a typical pattern, or listing remembered assertions as bullet points does not supply this evidence. If you cannot substantiate the objection, do not report it as an error; use "uncertain" if that material check remains unresolved.
- Declare an answer line wrong only after identifying and verifying a concrete erroneous calculation, premise, inference, or option-mapping step that leads to it. Quote the faulty step and demonstrate the error. "The answer seems unlikely" is not a reason, and an independently obtained different answer alone is not proof. If you cannot establish a specific faulty step, do not declare the answer line wrong; use "uncertain" if its correctness is unresolved.
- If a computed value matches none of the listed options, selecting the most plausible listed option by elimination is not itself an error. Do not reject the choice solely because no option exactly matches the value. Verify the calculation and elimination; an error verdict still requires a specific demonstrated mistake. This rule does not establish that the chosen option is correct; use "uncertain" if verification is inconclusive.
- Review the response as paused work, not as a completed submission. Do not report an error merely because the final answer has not yet been given, a derivation is unfinished, or a heading or open construct awaits its continuation. Missing explanation alone is not evidence of an error. Check the substantive claims and inferences already made; establish what is wrong with one of them rather than requiring future text. Preferences about style or method are not errors.
- Distinguish asserted conclusions from hypotheses or exploratory branches. Do not report an old error that has already been explicitly and correctly superseded, but check whether you have reused a superseded result.
- Explain confirmed errors with concrete evidence. Identify a specific faulty step; it need not be the earliest mistake. Respect the dependency rule for previously accepted text, and do not invent a root cause.

Decision and output

Use true if you establish at least one unresolved substantive error within the permitted review scope and with the required evidence, even if other checks remain incomplete. Use false only when there is substantive content to verify and you have sufficiently verified the requested content and its necessary dependencies within the permitted review scope without finding such an error. Use "uncertain" if the content you are asked to review contains no substantive statements, reasoning steps, or calculations to verify, or if no error was established but a material check remains inconclusive. The review trigger alone is not evidence of an error. Unfinished generation alone does not determine the verdict; assess what has already been written.

Return exactly one valid, single-line JSON object with these three keys in this order, and nothing else. Example for a confirmed error:

```
{"analysis": "<brief check>", "error": "<confirmed error and reason>",
"is_error": true}
```

Field requirements:

- "analysis": a brief account of what you checked and the decisive evidence or remaining uncertainty.
- "error": quote the erroneous statement or step and explain why it is wrong. For a factual or domain-specific objection, include the explicit contradiction, reproducible calculation, or item-by-item structural argument required above. For an error in accepted text, also show the current claim's dependency on it. When declaring an answer line wrong, include the specific verified faulty step leading to that answer, not a plausibility judgment. Include a verified correction or counterexample when available. For false or "uncertain", use the empty string.
- "is_error": the JSON boolean true, the JSON boolean false, or the JSON string "uncertain".

Escape quotes, backslashes, and line breaks correctly. Do not wrap the JSON in Markdown or include alternative verdict objects in its string fields.

Correction Prompt

You are the assistant answering the original user task. You have just reviewed your own response and reported an error.

Act as a rigorous corrector of your own work. Now verify that diagnosis and correct your original answer, not the review report.

Your replacement will replace this segment, and your response will continue immediately after the replacement:

```
<current_segment>
{segment_text}
</current_segment>
```

Earlier response text stays. Any later generated text and these temporary review/correction turns are discarded. Do not rely on discarded text or refer to these temporary turns in your correction.

Requirements

- Independently verify the reported error and the proposed correction using the original task and sound evidence. Your review and earlier reasoning may be mistaken. Do not invent facts or test results. An unfinished derivation or a final answer not yet given is not itself an error; if that is the only diagnosis, return can_correct=false.
- Identify the specific result, claim, or option you would correct. Check it independently. If recalculation or verification reproduces that same result, claim, or option, return can_correct=false. Also return can_correct=false if you cannot establish a definite correction. Do not force a different result, or rewrite a derivation merely to reach the same disputed result.
- Complete the calculations and verification needed for the proposed repair in your internal think/analysis phase before composing the replacement or output JSON. If the derivation is incomplete, work through its missing steps there and verify the result first. Do not use the replacement to finish unresolved calculations or promise future work. This does not authorize completing the visible answer. If verification remains inconclusive, return can_correct=false. Do not add an analysis output field.
- Keep the replacement at the stage reached by the judged segment. Repair work already stated, with only the supporting steps needed for that repair; do not advance to subsequent solution steps or add a concluding summary. Do not complete the problem unless this segment already gives its answer. This is a limit on solution progress, not a requirement to preserve the original length. Preserve necessary correct content.
- Write as the same assistant continuing your own solution, matching its language, voice, and format. Write the corrected work itself, not a review, a description of what should be fixed, or third-person commentary about "the model", "the assistant", or "the response". If earlier retained text is wrong, explicitly correct your earlier statement and supply the verified replacement basis here. Do not merely promise a later repair.
- State the verified, corrected result of the disputed step explicitly: give its concrete conclusion or numerical value and the supporting derivation. Merely saying that the step is "unsupported", "unjustified", or "not established" is not a correction. If you cannot establish a concrete replacement result, return can_correct=false. For an intermediate segment, this means the corrected intermediate result, not the task's final answer.
- If a repair depends on a factual or domain-specific judgment, the replacement itself must show how that judgment is reached. State the relevant task data and applicable criteria, then give the reproducible calculation or item-by-item structural checks establishing the corrected conclusion. Do not substitute a rule of thumb, remembered association, typical pattern, or plausibility judgment for this derivation. Internal verification alone is not sufficient; include the decisive evidence in the replacement. If you cannot substantiate the judgment, return can_correct=false.
- If the judged segment contains no final-answer statement, do not output, quote, or imply a final answer or select an answer option in the replacement. A heading alone is not an answer statement. Do not introduce final-answer markers or a final conclusion. Stop after the verified local repair.
- Whenever you change a final answer or selected option, the replacement itself must contain the derivation establishing the new answer. Include the decisive calculations or domain reasoning, not just a new value, option letter, or assertion that it is correct. Completing verification internally does not remove the requirement to include this derivation in the replacement.
- Before changing a multiple-choice selection, establish and show in the replacement why the new option satisfies all data and conditions given in the task and why the old option violates at least one specific given datum or condition. Check each given explicitly; being closer to a computed value or seeming more plausible is not sufficient. If you cannot establish both the new option's consistency and the old option's violation, return can_correct=false rather than change the selection.
- If the judged segment is or contains an answer line, the replacement must end with the corrected answer line in the same format. Preserve its answer marker, mathematical delimiters, option-label convention, and punctuation. Put the supporting derivation before that line, not on the answer line or inside its answer box. Write nothing after the answer line in the replacement.
- Do not replace a result or answer with an abstention such as "no answer", "cannot determine", or "unable to solve", or invent an unlisted option such as "None of the above". Do not guess. If no definite repair is established, return can_correct=false with an empty replacement instead. This does not require or permit adding an answer at an intermediate segment.
- Respect the task's actual output requirements, including code syntax. If the required derivation and correction cannot both be expressed within that format and the segment's permitted scope, return can_correct=false.

Output

Return exactly one valid, single-line JSON object with these two keys in this order, and nothing else:

```
{"can_correct": true, "replacement": "<the complete correction text>"}
```

If you cannot provide a reliable correction, return:

```
{"can_correct": false, "replacement": ""}
```

Inside "replacement", write only the text to insert, with no placeholder text, wrapper tags, input delimiters, or editorial labels.

Escape double quotes as \" and literal backslashes as \\. Encode intended line breaks as \u000a and internal tabs as \u0009. Do not wrap the JSON in Markdown. This runner may alter code fences, literal backslash-n sequences, and leading/trailing whitespace, and may skip whitespace-only changes. If those transformations prevent a reliable repair, return can_correct=false. A decline leaves the response unchanged; it does not request a retry.

Figure 5: **Prompt templates used by ThinkMid.** The system prompt and an example user query are shown above the review and correction prompts. Placeholders are filled with the corresponding response segments at inference time.

single response after the initial thinking, without additional review or refinement; it is the reference point that every scaled method extends. **Thinking Forcing** represents sequential scaling: following s1-style budget forcing (Muennighoff et al., 2025), every attempt of the model to end its thinking is suppressed and a cue word is appended, so that deliberation continues until the allotted thinking budget is reached, after which the thinking is closed and the model answers. **Best-of-N with an LLM judge (BoN-Judge)** represents parallel scaling: candidate solutions are generated independently and the same model, acting as a judge, reads all candidates and selects one. **Best-of-N with majority vote (BoN-Vote)** uses the same candidate-generation procedure but selects the most frequent final answer, following the self-consistency principle (Wang et al., 2023). **Self-Refine** represents iterative scaling: after an initial draft, the model alternates between a feedback call that critiques the current draft and a refine call that rewrites it, following Madaan et al. (2023), and the last draft is graded. These baselines contrast ThinkMid's allocation of computation during response generation with the three established alternatives: extending the initial thinking, generating and selecting among candidates, and refining completed responses. The per-problem budget rules under which each baseline runs are given in Appendix C.4.

C.3 Evaluation Metrics and Budget Accounting

We report **pass@1 accuracy** and **total token consumption (Tokens)** to evaluate answer quality and inference cost. Each evaluation run returns one final answer, including for methods that internally generate multiple candidates or revisions. For benchmark D and method M, accuracy is

$$\mathrm{Acc}_D(M) = \frac{100}{|D|R_D} \sum_{x \in D} \sum_{r=1}^{R_D} \mathbb{1}[\mathrm{Eval}_D(\hat{a}_{M,x,r}, x) = 1], \tag{7}$$

where $\hat{a}_{M,x,r}$ is the final answer and Eval_D denotes task-specific correctness evaluation. We set $R_D = 4$ for AIME 2025 and $R_D = 1$ for the other benchmarks. AIME therefore contributes 120 graded attempts over 30 distinct problems, with each attempt receiving equal weight.

Let $T_{M,x,r}$ denote the realized total token consumption of method M on problem x in run r. It includes prompt, thinking, and response tokens across all inference calls, covering initial generation, verification, correction, candidate selection, and refinement where applicable, as well as retries and generated text later discarded by rollback. We apply the same accounting convention to all methods and report

$$\mathrm{Tokens}_D(M) = \frac{1}{|D|R_D} \sum_{x \in D} \sum_{r=1}^{R_D} T_{M,x,r}. \tag{8}$$

The same quantity determines the budget for baseline comparisons. For each backbone and evaluated attempt, we use ThinkMid's realized consumption as the nominal token allocation for every non-Vanilla baseline:

$$B_{x,r} = T_{\mathrm{ThinkMid},x,r}. \tag{9}$$

Vanilla remains a single-response reference without an additional matched allocation. Budget matching therefore uses the assigned token allowance, while the Tokens metric reports actual consumption, including any completion overrun under the stopping policy described in Appendix C.4.

C.4 Decoding and Budget Settings

All models follow officially recommended decoding settings with temperature 0.6, top-$k = 20$, and top-$p = 0.95$. We set the random seed to 42 and run all experiments on a single NVIDIA A100-80GB GPU. Vanilla and ThinkMid share a maximum initial-thinking budget of $16{,}000$ tokens. Models may complete their initial thinking before reaching this limit; otherwise, we enforce the model-specific thinking boundary to begin response generation. ThinkMid allocates additional computation to review and correction during response generation, using a fixed entropy threshold of $\tau = 0.5$ across all models and benchmarks unless explicitly varied in the sensitivity analysis.

Under the per-problem budgets of Appendix C.3, *Thinking Forcing* reserves 500 tokens for the response and allocates the remainder, after prompt costs, to initial thinking. Budget forcing prevents early termination of thinking until this allowance is reached, after which the model transitions to answering. *BoN* generates candidates sequentially, each with an initial-thinking cap of $16{,}000$ tokens.

The prompt, thinking, and response tokens of every candidate count toward the per-problem budget, so the number of candidates N varies across problems. BoN-Judge additionally reserves tokens for the selection prompt, judge thinking, and selection output before allocating the remaining budget to candidate generation. *Self-Refine* allocates its budget across drafting, feedback, and revision, using the same token accounting as BoN. Both methods follow the same stopping policy: when the nominal budget is exhausted, we terminate any ongoing thinking, allow the current response to finish, and initiate no further generation rounds. This completion policy may exceed the nominal allocation; any resulting overrun is included in the reported token consumption.

C.5 Hyperparameters of ThinkMid

We describe here the hyperparameters of our method, summarized in Table 4. All of them are fixed across both backbones and all three benchmarks: none is tuned per model or per dataset, and the same values are used for every run reported in the paper except where a hyperparameter is explicitly varied in the ablation (Sec. 4.3) or the sensitivity analysis (Sec. 4.4).

Table 4: Hyperparameters of ThinkMid, identical for both models and all three benchmarks.

Quantity	Value
Initial thinking budget	16,000 tokens
Entropy threshold τ	0.5
Minimum segment length $\ell_{\min}$	3 tokens
Review turn: thinking budget / max new tokens	2,000 / 3,500
Correction turn: thinking budget / max new tokens	3,000 / 6,000
Regenerations of a malformed auxiliary output	2
Generation cap	51,200 tokens
Sampling (all turns)	temperature 0.6, top-p 0.95, top-k 20

D Intervention Analysis Details

This section documents the counterfactual analysis of Sec. 5. The run is the AIME 2025 evaluation of ThinkMid with Qwen3.6-35B-A3B at seed 42 (25 of 30 problems correct); 8 problems received at least one accepted correction, 11 in total, all within the visible response.

Protocol. At the moment each corrective text is spliced in, we record the prompt and all generated token ids up to and including the flagged sentence. The prefix is token-exact; nothing is re-tokenised from text. The same model then samples from this prefix to the end of sequence with the decoding settings of the main run (temperature 0.6, top-p 0.95, top-k 20, seed 42), the same 16,000-token initial-thinking budget and generation cap, and no review, correction or answer rescue. The continuation is graded with the same grader as the main run (last `\boxed{}` answer) and paired with ThinkMid's final answer on the same problem. For the whole-run figure quoted in Sec. refsec:analysis, problems without an intervention keep their outcome and intervened problems take the continuation from their first intervention point, which gives the accuracy the run would have had without any intervention: 21 of 30 problems (70.0%) against 25 of 30 (83.3%).

Rescued (5 points, 4 problems). In I-10, I-13 (first point), II-14 and both intermediate points of II-5, the flagged sentence states an intermediate result that later steps depend on: a count of set elements, an averaging step over segment combinations, a geometric consistency claim, and two claims about arcs and angles in a triangle's circumcircle. Continuing plainly from each of these prefixes ends in a wrong final answer, whereas the corrected run ends correctly. II-5 shows the cumulative effect of successive corrections: the first two pulled the derivation back on track and the third, at the answer line, only confirmed it.

Redundant (4 points). Two are answer-line reviews, I-13 (second point) at $\boxed{204}$ and II-5 (third point) at $\boxed{336}$, where the original answer was already correct and the corrective text adds derivation without changing it. The other two are I-11's claim that each k admits two roots, which was correct, and II-15's section heading "Condition for Two Global Minima," which carries no substantive claim; in both, the plain continuation is also correct.

Table 5: The 11 intervention points of the run. *Int.* indexes the interventions of a problem in order; *Where* is the position of the flagged sentence; *Cont. tokens* is the length of the plain continuation, where 0 means the flagged sentence was the answer line and the model stopped at once; *Plain* and *Ours* give the correctness of the plain continuation and of ThinkMid's final answer.

Task	Int.	Flagged sentence (abridged)	Cont. tokens	Plain	Ours	Outcome
I-10	1/1	By tracking the number of elements from A, B, C that each …	495	wrong	right	rescued
I-11	1/1	For each k, there are two roots.	730	right	right	redundant
I-13	1/2	By averaging over all possible combinations of segment t…	189	wrong	right	rescued
I-13	2/2	The expected number of regions is $\boxed{204}$. (answer line)	0	right	right	redundant
I-15	1/1	There are $3^3 = 27$ valid triples (u, v, w) modulo 3.	118	wrong	wrong	both wrong
II-5	1/3	The measure of arc HJ is $2 \times \angle HFJ$ or can be …	254	wrong	right	rescued
II-5	2/3	Wait, the arc between feet of altitudes H_B and H_C is …	615	wrong	right	rescued
II-5	3/3	$\boxed{336}$ (answer line)	0	right	right	redundant
II-10	1/1	$\boxed{869}$ (answer line; reference 907)	0	wrong	wrong	both wrong
II-14	1/1	This confirms the geometric consistency of the points K and …	397	wrong	right	rescued
II-15	1/1	### 2. Condition for Two Global Minima (heading)	647	right	right	redundant

Both wrong (2 points). In I-15 the flagged claim treats all 27 residue triples (u, v, w) modulo 3 as valid, an over-counted case split that the verifier flags but the corrector does not repair. In II-10 the review is opened at the answer line $\boxed{869}$, against a reference answer of 907; the verifier judges the answer wrong, but the corrective text does not arrive at 907. These are the points where the current mechanism tops out: the error is located, but the correction turn cannot supply the right content within its budget.