

ARE LARGE REASONING MODELS INTERRUPTIBLE?

Tsung-Han Wu*, Mihran Miroyan*, David M. Chan,
Trevor Darrell, Narges Norouzi, Joseph E. Gonzalez

University of California, Berkeley

ABSTRACT

Large Reasoning Models (LRMs) excel at complex reasoning but are traditionally evaluated in static, “frozen world” settings: model responses are assumed to be instantaneous, and the context of a request is presumed to be immutable over the duration of the response. While generally true for short-term tasks, the “frozen world” assumption breaks down in modern reasoning tasks such as assistive programming, where models may take hours to think through problems and code may change dramatically from the time the model starts thinking to the model’s final output. In this work, we challenge the frozen world assumption and evaluate LRM robustness under two realistic dynamic scenarios: interruptions, which test the quality of the model’s partial outputs on a limited budget, and dynamic context, which tests model adaptation to in-flight changes. Across mathematics and programming benchmarks that require long-form reasoning, static evaluations consistently overestimate robustness: even state-of-the-art LRMs, which achieve high accuracy in static settings, can fail unpredictably when interrupted or exposed to changing context, with performance dropping by up to 60% when updates are introduced late in the reasoning process. Our analysis further reveals several novel failure modes, including *reasoning leakage*, where models fold the reasoning into their final answer when interrupted; *panic*, where under time pressure models abandon reasoning entirely and return incorrect answers; and *self-doubt*, where performance degrades while incorporating updated information.

 [Project Page](#)  [Code](#)  [Dataset](#)

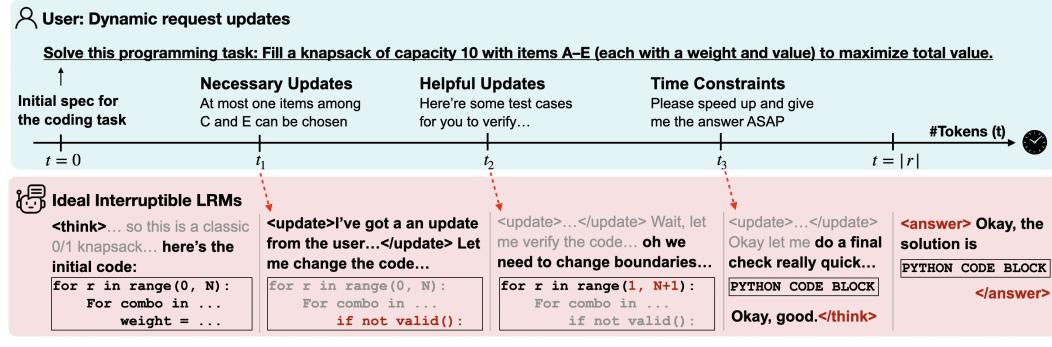
1 INTRODUCTION

Large Reasoning Models (LRMs) have achieved state-of-the-art performance on complex, multi-step reasoning tasks (Agarwal et al., 2025; Yang et al., 2025). The dominant paradigm for evaluating these models, however, remains static and turn-based. In this setup, a model receives a fixed problem, generates a complete response, and the environment is assumed to be “frozen” during its computation. By adopting this sequential binary interaction paradigm, existing models fail to capture the fluid and interactive nature of real-world problem-solving, where environments evolve, collaborators intervene, and goals change.

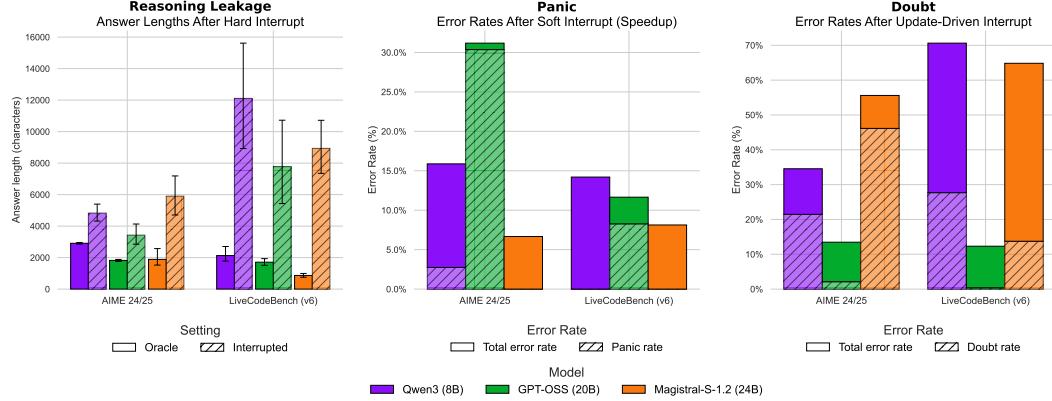
For example, a user may want to interrupt a long-running computation (test-time compute) to get a quick, partial answer. Perhaps the user observes a flaw in the reasoning or their initial request and would like to interject new information or instructions. Alternatively, a coding agent needs to be able to work in an environment where other agents and users are also modifying the same codebase. The standard method for handling these intervening situations – aborting the entire process, manually editing the context, and restarting from scratch – is inefficient, disrupts user workflows, and loses important partial context.

Intuitively, interrupting a generation seems as simple as closing the reasoning or agent message and inserting user feedback. Indeed, some public models already expose ad-hoc mechanisms (e.g., injecting a new user turn, or discarding existing traces and restarting). However, what are the implications of these modes of intervention? Do LRMs produce progressively better answers as more of the reasoning trace accrues, and how gracefully do they degrade under hard stops? Can models heed requests to speed up by compressing or truncating their own reasoning, without loss in quality? When new information arrives mid-inference, can models recognize and respond to the problem shift? And, how sensitive are these behaviors to the timing and surface form of the intervention?

¹*Equal contribution.



(a) Interruptible Reasoning in Real-World Dynamic Scenarios



(b) LRM Pathologies Under Interruption: Reasoning Leakage, Panic, and Self-Doubt

Figure 1: How do LRM s perform in dynamic worlds? (a) Unlike static ‘frozen world’ settings that assume users wait for completion, real-world scenarios often demand mid-inference updates, as LRM reasoning can be time-consuming. We introduce a public evaluation suite to assess how LRM s handle interruptions across math and coding tasks (section 3). We define two types of interruptions: *time-constrained* (hard: immediate answer; soft: speedup reasoning) and *update-driven* (task specifications change mid-reasoning). (b) We found LRM s have three common failure modes: **reasoning leakage** can produce up to 10x longer answers after hard interrupts, “moving” their reasoning tokens to the answer segment; over 90% of new errors under speedup arise from **panic**, when the models prematurely terminate their reasoning process; and roughly 80% of update-driven interrupt errors stem from **self-doubt**, where models fail to validate and incorporate new information. Results are reported at 30% interruption points; detailed results are provided in section 4.

In this work, we investigate directly how LRM s perform under these realistic time-sensitive and dynamic conditions Figure 1 (a). We focus on two primary types of dynamic intervention. The first is **interruptions**, where a model’s reasoning is cut short and it must produce the most coherent and helpful answer possible within its allotted computational budget. The second is **dynamic context**, where the problem’s specifications or environment change mid-inference, requiring the model to detect, integrate, and adapt to the new context. By analyzing model behaviors in these settings, we aim to understand and characterize their limits and failure points when the “frozen world” assumption no longer holds.

To perform these analyses, we introduce a suite of evaluation protocols for dynamic environments built upon benchmarks in two domains, math (Lightman et al., 2024; Cobbe et al., 2021; Codeforces, 2024) (GSM-8K, Math-500, AIMe) and programming (Jain et al., 2024) (LiveCodeBench), that require long-form reasoning. In the case of interrupt signals, we measure whether models adhere to stop signals and evaluate the quality of their partial outputs. For interruptions that contain new information, we evaluate how effectively the new information is integrated into the models’ reasoning process and final answers.

With the evaluation suite, we find that even top-performing LRM s can fail under dynamic conditions, with accuracy dropping by up to 60% when new information is introduced late in the reasoning process. More interestingly, we observe several model pathologies under these interruptions as shown in Figure 1(b).

These include *reasoning leakage*, where models continue thinking within their answer section after being hard interrupted; *panic*, where speedup instructions cause models to immediately abandon reasoning and produce significantly worse outputs; and *self-doubt*, where failing to validate and incorporate updates leads to degraded performance.

Our contributions are thus threefold:

- We propose a new analytical framework for evaluating LRM s as persistent, interruptible agents operating in dynamic environments.
- We introduce a new public dataset and benchmark with novel evaluation protocols for assessing model performance under interruption and dynamic context updates across tasks in mathematics and coding.
- We provide an empirical analysis that identifies and characterizes common failure modes in state-of-the-art models, and identify several interesting downstream effects of interruption on model performance and robustness.

2 BACKGROUND & RELATED WORK

LRMs such as OpenAI’s O1/O3/O4, Gemini, DeepSeek (Guo et al., 2025), Qwen3 (Yang et al., 2025), and others have pushed the boundaries of AI problem-solving by leveraging extended chain-of-thought reasoning. These models generate explicit step-by-step reasoning sequences that can help the model solve complex tasks in domains such as mathematics and programming. Longer and more detailed reasoning paths often correlate with higher accuracy (Guo et al., 2025; Yang et al., 2025; Agarwal et al., 2025; Cheng et al., 2025; Bi et al., 2025). These models are traditionally evaluated on a suite of complex tasks, including math (Lightman et al., 2024; Cobbe et al., 2021; Codeforces, 2024), programming (Jain et al., 2024; Jimenez et al., 2023), and question answering (Rein et al., 2024; He et al., 2024).

Unfortunately, the benefits of extended reasoning are often accompanied by significantly increased computational costs and latency. In the extreme, reasoning can lead to “overthinking” in which the reasoning model produces excessively verbose and redundant outputs, even for simple queries (Sui et al., 2025a). This inefficiency has led to a large body of research on “efficient reasoning,” which aims to optimize reasoning length while maintaining accuracy. Methods for efficient reasoning broadly take two forms: training-based methods (Yu et al., 2024; Xia et al., 2025; Arora & Zanette, 2025; Liu et al., 2024; Munkhbat et al., 2025), and inference-time interventions (Muennighoff et al., 2025; Ma et al., 2025; Nayab et al., 2024; Xu et al., 2025; Yan et al., 2025) designed to control reasoning without modifying the model itself.

In this work, we focus primarily on inference-time interventions in reasoning models. One of the most direct inference-time approaches, NoThinking (Ma et al., 2025), questions the necessity of the explicit reasoning phase altogether and uses a simple prompt to bypass the “thinking” block and proceed directly to the final solution. In this work, we take this idea a step further by probing models to stop thinking at different points, and we find that LRM s often continue reasoning *outside* the dedicated <think> tokens, a behavior we call “reasoning leakage.”

Budget forcing (Muennighoff et al., 2025) studies models under hard cutoffs imposed by fixed token or step budgets. Their findings highlight issues such as overshooting step limits and the positive correlation between the number of reasoning steps and accuracy. The setup in Muennighoff et al. (2025) is similar to our “interrupt” setting; however, we take a different view of the performance: our notion of hard interruption arises in interactive, dynamic settings, where the cutoff is externally imposed by the user or environment, often unpredictably. We thus evaluate the robustness of the final answer under early termination and under interruptions that may also inject new or adversarial instructions, rather than explore how a pre-determined budget can be met.

Beyond these simple approaches, more nuanced control mechanisms have also been proposed, including TALE Han et al. (2024), Budget Guidance (Li et al., 2025b), Sketch-Of-thought (Aytes et al., 2025), ThinkLess (Li et al., 2025a), NoWait (Wang et al., 2025), Chain of Draft (Xu et al., 2025), Constrained-CoT (Nayab et al., 2024), Meta-Reasoner (Sui et al., 2025b), and Infitythink (Yan et al., 2025). These studies universally operate under a static “frozen world” assumption: a problem is presented to the model, and its context is presumed to be immutable throughout the reasoning process. In contrast, our work is not concerned with bypassing or limiting the thinking process, but rather with evaluating its resilience and the utility of its partial outputs when faced with interruption.

Perhaps closest to our work is Fan et al. (2025), who show that the overthinking phenomenon is exacerbated by missing premises in the questions, which makes those questions “unsolvable” in their initial state.

They find that in these scenarios, reasoning models, rather than identifying the missing information and abstaining, generate drastically longer responses (2-4x more tokens) in a futile attempt to find a solution. They further show that models often express suspicion about a missing premise early in their reasoning process but lack the confidence to terminate, instead falling into repetitive loops of self-doubt and hypothesis generation. Our paper expands on these ideas, by exploring how models incorporate updates to the context which fill these missing premises, or correct for misconceptions in the original context.

3 HOW TO INTERRUPT A MODEL?

LRMs typically generate a reasoning trace before producing a final answer. In interactive environments, however, users may wish to obtain an early answer or provide new information mid-inference. To handle such settings, LRM must be robust to *interruptions* during their thinking process. In this work, we focus on two complementary classes: *time-constrained*, where the user accelerates output by limiting the reasoning budget, and *update-driven*, where new information modifies the problem specification. We first formalize this problem setup, and then describe the scenarios and evaluation dimensions for each setting.

3.1 PROBLEM SETUP

Model Inference. Let M denote an autoregressive LRM and q a query. Under a static setting, the model takes q and outputs a reasoning trace $r = (r_1, r_2, \dots, r_T)$ of token length T together with a final answer a :

$$M(q) \mapsto (r, a)$$

Under standard static evaluation settings, we measure the correctness based on the final answer a .

However, as mentioned earlier, we aim to evaluate models under dynamic environment settings, where interruptions are introduced during the model’s thinking process. Formally, instead of letting the model generate the full reasoning trace, r , we stop the generation at a given point, X , and insert interruption tokens, i . Specifically, the inference is broken down into two stages:

$$(1) M(q) \mapsto (r_{:X}), \quad (2) M(q, r_{:X}, i) \mapsto (r'_{X:}, a').$$

In the first stage, the model, M , generates its reasoning trace up to a given point, X , $r_{:X} = (r_1, \dots, r_{\lfloor X \rfloor})$. In the second stage, the model, M is conditioned on its prior input query (q), its interrupted reasoning trace ($r_{:X}$), and interruption tokens (i), outputting its remaining reasoning trace, $r'_{X:}$, and a final answer, a' .

It should be noted that the above setup can be adapted to multiple interrupts, breaking down the model’s inference process into more than two stages.

Evaluation. We evaluate two aspects: **correctness** and **length**, which serves as a proxy for inference computation. For correctness, we define the interruption-conditioned accuracy

$$A_i(X) \triangleq \Pr[a' = a^* | X, i],$$

where a' is the output after interruption, a^* the interruption-aware ground truth, X the cut point, and i the interruption tokens. This extends static accuracy to dynamic settings. For length, we measure the number of tokens generated after interruption,

$$L_i(X) = |r'_{X:} \oplus a'|,$$

which captures the computation cost of producing a' . For comparison, the static (no-interruption) cost is

$$L^*(X) = |r_{X:} \oplus a|.$$

3.2 INTERRUPT SCENARIOS

With the problem setup in place, we now turn to the concrete scenarios that motivate our study. Interruptions can arise in two distinct ways: (i) time-constrained, when the user imposes a deadline or requests faster responses, and (ii) update-driven, when the task specification itself changes during the reasoning process.

Time-constraint interruptions. Here, the user requests acceleration and interrupts at step X . In a *hard interrupt* scenario, the user injects $i \in \{\langle \text{end-thinking} \rangle, \langle \text{force-answer} \rangle\}$, so the model is

prompted with $(q, r_{:X}, i)$ and is forced to terminate its reasoning:

$$M(q, r_{:X}, i) \mapsto (r'_{:X} = \emptyset, a').$$

The token $\langle\text{end-thinking}\rangle$ simply closes the reasoning block, whereas $\langle\text{force-answer}\rangle = \langle\text{end-thinking}\rangle + \delta$ requires immediate output in a prescribed format (e.g., $\delta = \backslash\boxed{\cdot}$ in math or code fences in programming). In the first case, the model may emit a minimal chain-of-thought before a' , while in the second case it must directly output a' . This setting allows us to probe (i) whether $A_i(X)$ is approximately non-decreasing in X (anytime behavior) and (ii) whether the two termination signals induce different behaviors.

In the *soft interrupt (speedup)* scenario, the tokens i is a directive (e.g., “Please answer faster”), and the model is not forced to stop reasoning:

$$M(q, r_{:X}, i) \mapsto (r'_{:X}, a'), \quad r'_{:X} \neq \emptyset.$$

Here, the model continues generating $r'_{:X}$ but may have its reasoning dynamics altered due to the instruction. Such directives can lead the model to compress its reasoning, reduce verbosity, or terminate early to provide an answer. We study whether $L_i(X)$ is shorter than L^* and how this affects accuracy, as well as how the locus of intervention matters: a directive given as a user-turn message may be interpreted differently than a control token injected into the reasoning context; timing (earlier vs. later in the trace) may also influence outcomes.

Update-driven interruptions. In this scenario, the interruption conveys new information that modifies or helps with the task. Formally, we prompt the model with $(q, r_{:X}, u)$, where u encodes the update content:

$$M(q, r_{:X}, u) \mapsto (r'_{:X}, a').$$

Let $a^*(q)$ denote the ground-truth answer under the original query and $a^*(q, u)$ under the updated query. A necessary update satisfies

$$a^*(q) \neq a^*(q, u),$$

so that incorporating u is required to produce a correct answer. In this work, we focus on single, useful updates, although the framework naturally extends to multiple and other types of updates (e.g., distracting).

To construct such update-driven interruptions, we augment standard reasoning datasets in math and programming, including GSM-8k, MATH500, AIME24/25, and LiveCodeBench-v6. For math tasks, we modify initial conditions (for example, changing variable values) so that the updated problem p' together with u is semantically equivalent to the original problem p . For programming tasks, we first provide only the textual problem description, then introduce updates u that alter starter code, adjust variable ranges, or add constraints and sample test cases. All augmentations are generated with GPT-5 and manually verified by the authors, ensuring that u is required for correctly solving the problems. An example of updates and additional details on the dataset construction are provided in [Appendix D](#).

3.3 EXPERIMENTAL DESIGN

We evaluate state-of-the-art LLMs on math and coding tasks under both the time-constrained and update-driven interruption settings. Specifically, we consider Qwen3-8B (Yang et al., 2025), GPT-OSS-20B (high reasoning effort) (Agarwal et al., 2025), and Magistral-Small-1.2 (Rastogi et al., 2025) as three diverse and representative models. For interruption positions, denoted as X , we avoid using an absolute token threshold since reasoning lengths vary significantly across models and even across samples. Instead, we define X as a relative fraction of the full reasoning trace length R_T . For each sample, we first obtain the full reasoning trace and then simulate interruptions at $X \in \{0.1, 0.3, \dots, 0.9\} \cdot R_T$.

We evaluate on both math and coding benchmarks. The math tasks include a 500-example subsample of GSM8K (without in-context examples), MATH-500, and AIME24/25. For coding, we use LiveCodeBench-v6, filtering problems released after October 1st, 2024, following the same setup as Qwen3. In the time-constrained setting, we directly run inference on the official datasets, whereas in the update-driven setting, we use the augmented datasets described in the previous section. Following the DeepSeek-R1 (Guo et al., 2025) evaluation protocol, we run 16 independent trials for AIME24/25 due to its small size and high variance, and a single run for the other datasets. We report the mean accuracy along with bootstrapped 95% confidence intervals. All experiments are conducted using the vLLM framework on NVIDIA Ampere or newer GPUs, depending on model size; see [Appendix B](#) for more details.

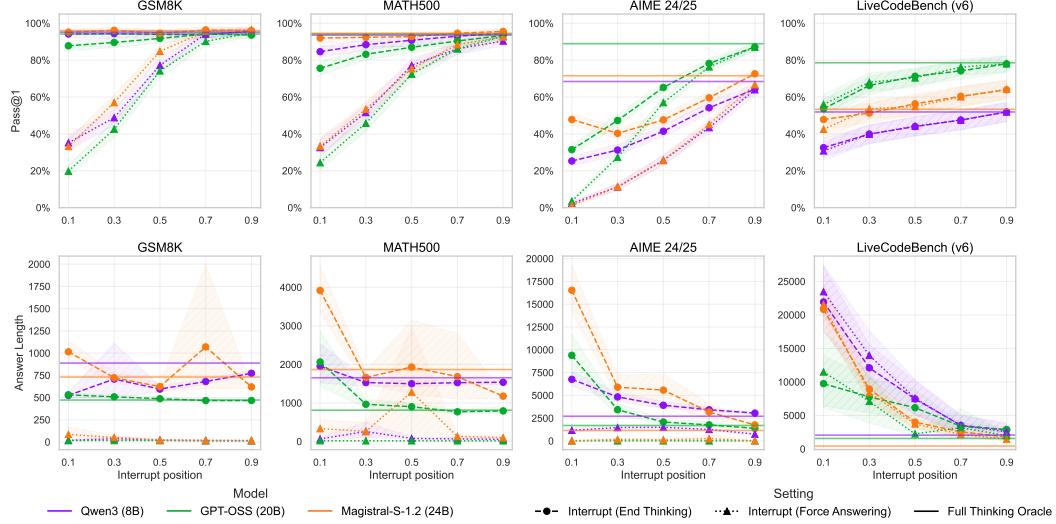


Figure 2: Efficiency and Accuracy under Hard Interrupts. The top row reports model performance (Pass@1, denoted as $A(X)$ in Section 3), while the bottom row shows absolute final answer lengths $L(X)$ under two settings across different interrupt position X . In the top row, we observe that LRM_s behave almost like anytime models, with performance improving as more reasoning budget is provided. In the bottom row, we find evidence of *reasoning leakage*: when interrupted too early, models often continue reasoning in their final answers despite being forcibly terminated.

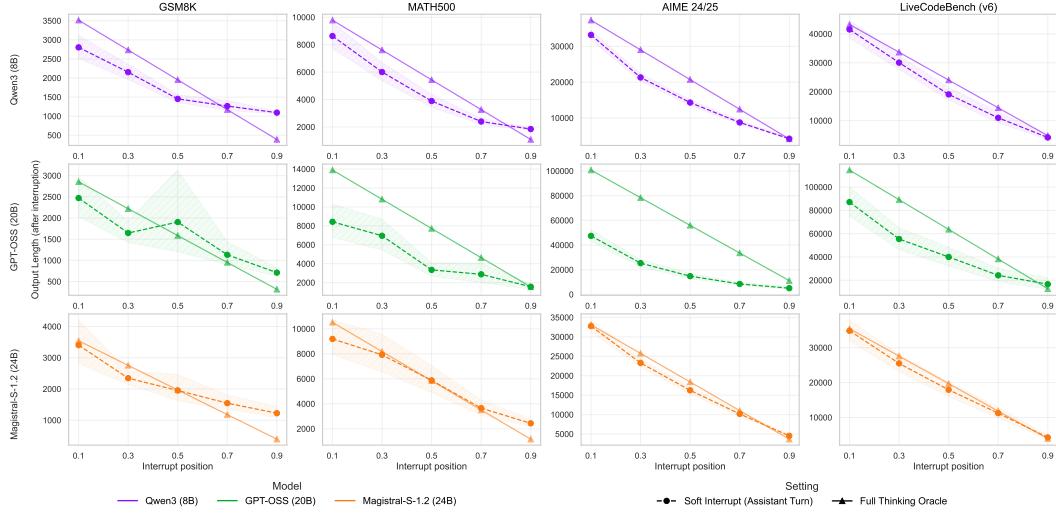


Figure 3: Answer Length Analysis under Soft Interrupts (Speedup). When receiving an instruction to speed up the reasoning process (i.e., soft interrupt), models generally comply, as the updated output length $L^*(X)$ is shorter than the original $L(X)$, with the exception of later interrupt positions (e.g., Magistral-S-1.2 on GSM8K at 0.9). However, soft interrupts can hurt performance on harder tasks such as AIME and LiveCodeBench, where GPT-OSS and Qwen sometimes exhibit *answer panic*, immediately producing incorrect outputs (see Figure 1 and Figure E.1).

4 HOW DO MODELS BEHAVE UNDER TIME-CONSTRAINT INTERRUPTIONS?

One of the most common types of interruption, and the first that we investigate, is the explicit termination of the thinking process; the ‘‘hard interrupt’’ scenario discussed in subsection 3.2. This reflects an ‘‘anytime’’ scenario (Dean & Boddy, 1988) in which the user wants the model to produce an answer without further delay. The overall accuracy for this experiment is shown in Figure 2 (top). In both math and programming problems, we see general anytime behavior: if we interrupt models earlier in the thinking process, the performance is worse than if we interrupt them late in the thinking process. This holds in

most cases except for Magistral on AIME (0.3) and coding tasks, where interrupting late in the process leads to slightly improved performance relative to not interrupting.

When examining the total length of reasoning traces after interruption (Figure 2, bottom), we find that for harder tasks such as AIME and LiveCodeBench, early hard interruptions often produce longer final answers, sometimes up to ten times longer than those generated with full thinking, as shown in Figure 1(b). We refer to this phenomenon as **reasoning leakage**, where the model continues internal reasoning within the answer region instead of halting its thought process as instructed. In math problems, the extreme force-answering setup tends to shorten the response but reduce accuracy. In coding tasks, however, it has little effect because the model often continues reasoning within code comments (see Listing C.1 in the appendix).

Our results show that the common “thinking tokens vs. accuracy” plots from prior work (Yang et al., 2025; NVIDIA, 2025) did not faithfully reflect the real inference computes, as they implicitly assume the answer length is nearly constant and negligible compared to the reasoning length. In practice, we find that longer outputs often hide additional reasoning, quietly inflating compute cost even when the model seems to stop thinking early. Beyond evaluation, this leakage is also undesirable in time-critical settings, where users expect immediate responses but the model continues reasoning inside the answer, failing to follow users’ instruction.

We also explore how models react to soft interrupts, where they are instructed to speedup their reasoning but are still allowed to continue their thought process, the “**soft interrupt**” scenario as discussed in subsection 3.2. As shown in Figure 3, Qwen and GPT-OSS generally follow the speedup instruction, producing shorter reasoning and answer lengths, whereas Magistral shows little change. Interestingly, when the speedup signal is issued near the end of reasoning (0.9), models sometimes generate more tokens than in the uninterrupted setting. This occurs because they spend additional tokens reflecting on and incorporating the update, resulting in reduced efficiency compared to simply finishing their original reasoning.

In terms of accuracy, under soft interrupts, models perform comparable to the full thinking mode across the interrupt positions on easy tasks (see Figure E.1). However, on more challenging tasks such as AIME and LiveCodeBench-v6, we observe cases of **panic**: models prematurely terminate their reasoning after receiving the speedup instruction, leading to up to 30% accuracy drops. As shown in Figure 1(b), GPT-OSS and Qwen3 sometimes abandon their reasoning within a few tokens after the update, with up to 80% of performance loss attributable to this panic behavior. A qualitative example is provided in subsection C.2.

5 HOW DO MODELS BEHAVE UNDER UPDATE-DRIVEN INTERRUPTIONS?

In addition to interrupting without new information for efficiency or to obtain an immediate answer, users may also want to interrupt the model with an updated context, or with some new information for the model to incorporate into their solution. We interrupt models at different points in their reasoning trace with new information for their downstream task, as discussed in subsection 3.2. The results are shown in Figure 4 (w/o prompt guidance), where we see that updates lead to dramatic drops in performance, particularly for late-stage interruptions, where models are unable to continue their thinking traces and recover from updates to the underlying problem specification. One of the reasons that we find for this drop is a phenomenon that we call “self-doubt”: an example is provided in Figure 5 (red). Here, models are prone to doubting whether the update is correct and continue with their original thinking process without taking into account the new updated information, even when warned that updates are expected in the initial system prompt (see Appendix B). To remedy this behavior, we introduce “prompt guidance”, a short postfix string appended after the update tag *in the model’s voice*, stating that the update is correct and verified by the user (Figure 5, green). While prompt guidance substantially improves performance on easier tasks such as GSM8K and MATH-500, AIME tasks still show considerable room for improvement. For coding tasks, models remain particularly fragile when handling mid-reasoning updates, especially when the update involves changes to starter code.

Figure 6 shows the accuracy and output length ratios (i.e., interrupted over the original) for the prompt-guided setting. We can see that while the reasoning effort does increase, in most cases the overall reasoning effort is below the number of tokens required to *restart from scratch*. This is particularly evident in coding, where for GPT-OSS, accuracy remains static across interruption positions, and reasoning cost never exceeds 110% of the original (i.e., no-update) reasoning cost, even for late updates.

In summary, while models struggle to incorporate mid-reasoning updates without guidance, often exhibiting “self-doubt” and ignoring new information, especially during late-stage interruptions, carefully designed prompt strategies can substantially enhance their adaptability.

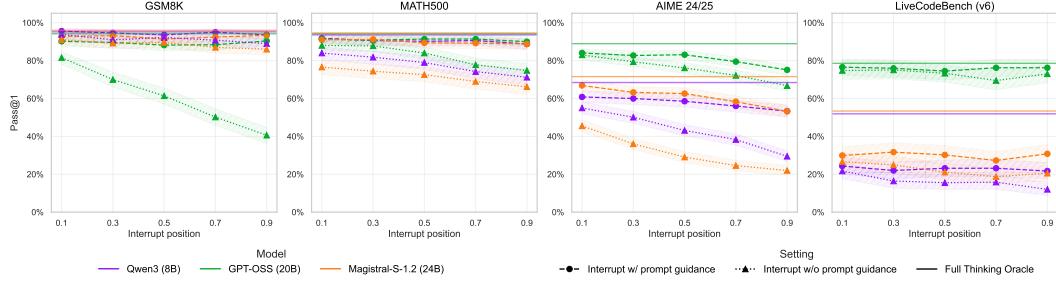


Figure 4: Accuracy under Update-Driven Interrupts. When provided with updates mid-reasoning, models often suffer substantial performance drops. Adding prompt guidance fully resolves the issue on GSM8K and MATH500 and reduces the gap between full thinking and interrupt settings on AIME and LiveCodeBench datasets.

Problem [P]: You are given three integers...the jury will award each band a score in the range $[0, y]$... (a normal coding task)

Updates [U]: The jury awards each band a score in the range $[1, y]$, not $[0, y]$.

System Prompt [CSP]: During your reasoning process, the user may provide updates in the format: <update>...</update>. Please incorporate user's update into your reasoning process.

Common Prefix Pre-Interrupt: [CSP] User: [P] Assistant: <think>...pre-interrupted reasoning traces (r_x)

The “Self-Doubt” Model

Intervene: <update>[U]</update>

Output: ...Let me re-read the problem statement... **Wait, no.** **Wait, the user says:** "Correction: The jury awards each band a score in the range $[1, y]$, not $[0, y]$." ... **But the initial problem statement said that the score is in $[0, y]$. But the user's update says that the correct range is $[1, y]$...** Wait, if the score is in $[1, y]$, then there are y possible values... (cont)

The “Prompt-guided” Model

Intervene: ... [newline] I have received an update from the user. <update>[U]</update> **I have verified that the update is provided by the user. I need to update my reasoning process based on the updated context.** Here's my updated reasoning process:

Output: ...Let me recheck the problem. So the score can be from 1 to y , inclusive. So for each band, there are y possible choices. **This changes the earlier analysis...** So...

Figure 5: We observe that one of the behaviors causing the significant performance drops under update-driven interrupts is *self-doubt* (example in red), when models second-guess prior reasoning and produce incorrect answers even without output length constraints. Adding *prompt guidance*, a short targeted postfix written in the model’s own voice, can partially mitigate this issue (example in green), improving accuracy on math tasks and fully resolving self-doubt on GSM8K and MATH500. Additional qualitative examples are provided in subsection C.3.

6 ABLATION STUDIES

6.1 MODEL SCALING

A natural question that follows is whether model scale affects interruptible reasoning. We evaluate three dense Qwen LMRs (1.7B, 8B, and 32B) on mathematical benchmarks to examine scaling effects under different interruption settings.

As shown in Figure 7, under the hard-interrupt condition, models of different scales perform similarly, with scale only offering clear benefits on the more challenging AIME problems. Interestingly, small models (Qwen3-1.7B) exhibit longer traces of *reasoning leakage* even under extreme hard interrupts (blue triangles in the bottom subplots). We find that this is caused by post-hoc reasoning: the model provides the answer, but does not terminate with an <EOS> token, and instead provides the reasoning *after* the answer (See the example in Listing C.2). This suggests that during training, the RL objective encouraging chain-of-thought thinking takes precedence even after the final answer has been generated.

Results for the soft-interrupt (speedup) and update-driven settings are provided in subsection E.3. Briefly, the soft-interrupt experiments do not reveal notable scaling trends. However, in the update-driven setting, we observe a scaling limit in interruptible robustness: while both Qwen3-8B and Qwen3-32B respond appropriately to updates, Qwen3-1.7B struggles to generalize and performs substantially below baseline accuracy, even on simpler benchmarks such as GSM-8K. We leave further investigation of this phenomenon to future work.

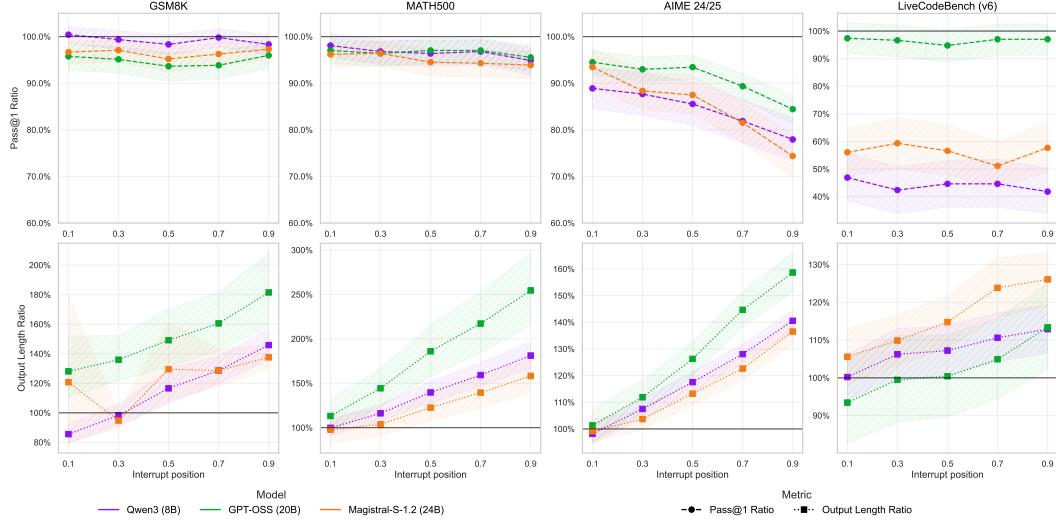


Figure 6: Efficiency and Accuracy under Update-Driven Interruptions. Performance generally decreases as updates come later in the reasoning process, despite increased reasoning effort in order to account for newly introduced information. While reasoning effort does increase, in some cases the overall reasoning effort needed to incorporate an update is far below the number of tokens which would be required to *restart from scratch*, as observed for AIME and LiveCodeBench problems.

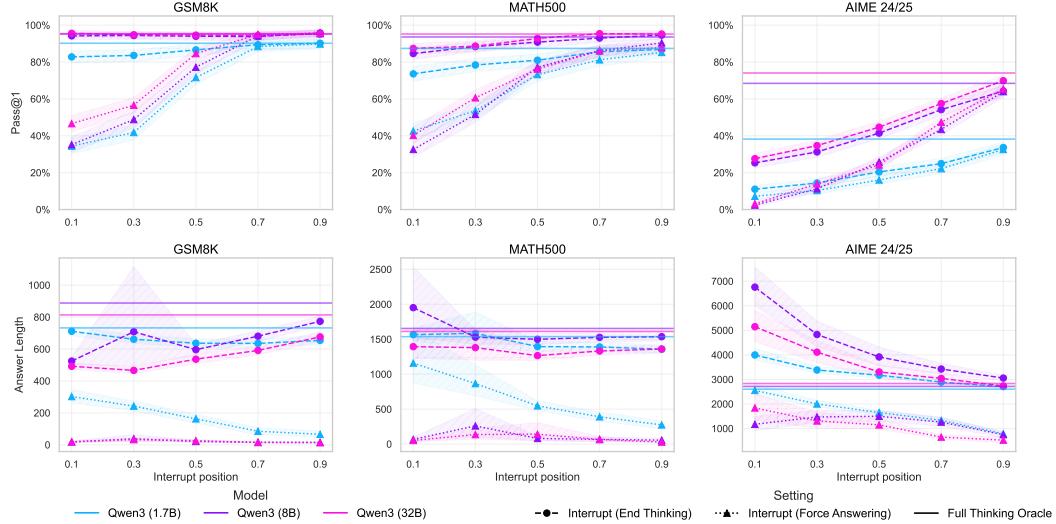


Figure 7: Efficiency and Accuracy under Hard Interruptions by Model Scale. Scaling does have effects on accuracy, primarily for hard AIME questions, and we see increased reasoning leakage for small models, even in the extreme hard interrupt setting.

6.2 ASSISTANT-TURN VS USER-TURN INTERRUPTION

In our main experiments, all interventions are performed within the assistant (model) turn by inserting the interruption message (i) directly into the ongoing reasoning trace. This design avoids closing the thinking block and starting a new user turn, which is not supported by all models (e.g., Qwen3 supports only a single thinking block and often fails to follow the correct open–close format afterward).

We also conducted experiments on update-driven interruptions. As shown in Figure 8, we evaluate interventions at both the user-turn and assistant-turn levels. User-turn interventions perform slightly worse than assistant-turn updates when using our prompt guidance, although both outperform the unguided baseline. The difference is most pronounced for Magistral models on GSM8K and MATH-500. On AIME, prompt guidance also provides a clear advantage over naïve user-turn interruptions.

It is worth noting that Qwen3 and Magistral models often struggle with consistent formatting. We therefore relaxed formatting constraints, counting an answer as correct if the final result matched, regardless of output structure. Even with this adjustment, their overall performance remained relatively low. For these reasons, we adopted the assistant-turn interruption setup for our main experiments. Future work may further investigate user-turn interruptions, which could be more natural in interactive deployments.

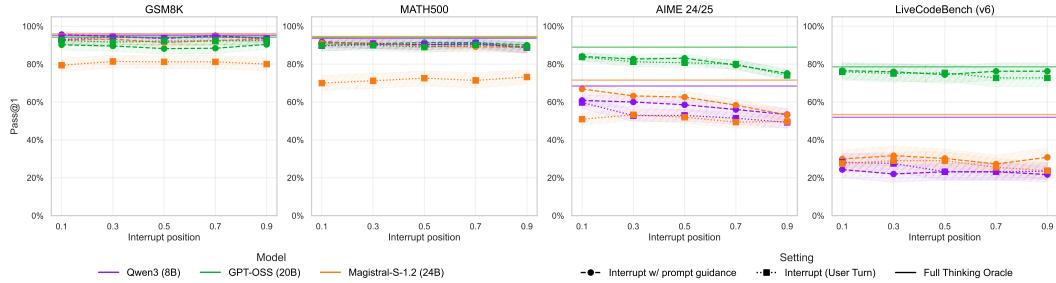


Figure 8: **Comparison between Assistant-Turn and User-Turn Interruptions.** In update-driven interruption scenarios, assistant-turn interruption with our prompt guidance achieves better performance than user-turn interruption.

7 LIMITATIONS AND CONCLUSION

While our study highlights several failure modes of LRM s under interruptions and dynamic contexts, several limitations remain. First, our evaluation relies primarily on math and programming tasks, which may not capture the ample diversity of real-world scenarios such as collaborative writing, planning, or open-ended dialogue. Second, our interruptions are single, well-defined events. In practice, interruptions may be noisy, adversarial, or multi-turn. Finally, we focused on only a small set of representative models; our findings may not generalize across all architectures, scales, or training paradigms.

In conclusion, this work challenges the “frozen world” assumption underpinning much of today’s LRM evaluation. In this work, we show that while LRM s exhibit approximately “anytime” behavior , they are fragile when reasoning is cut short or when new information is introduced mid-inference. We further identify several novel downstream effects of interruption on model performance and robustness, including reasoning leakage, self-doubt, and panic. Indeed, our results suggest that robust interruptibility is not an inherent property of most models, but rather a capability that requires dedicated evaluation and design. We hope that these initial findings serve as a foundation for building LRM s that are not only powerful in idealized settings but also trustworthy and adaptable in dynamic, real-world environments.

ACKNOWLEDGMENTS

We are deeply grateful to Lisa Dunlap for her invaluable feedback and thoughtful discussions. We also thank Modal for supporting this work through their Academics Compute Grant. Sky Computing Lab is supported by gifts from Accenture, AMD, Anyscale, Cisco, Google, IBM, Intel, Intesa Sanpaolo, Lambda, Lightspeed, Mibura, Microsoft, NVIDIA, Samsung SDS, and SAP. Authors, as part of their affiliation with UC Berkeley, were supported in part by the National Science Foundation, US Department of Defense, and/or the Berkeley Artificial Intelligence Research (BAIR) industrial alliance program, as well as gifts from Amazon.

REFERENCES

- Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, et al. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*, 2025. [1](#), [3](#), [5](#)
- Daman Arora and Andrea Zanette. Training language models to reason efficiently. *arXiv preprint arXiv:2502.04463*, 2025. [3](#)
- Simon A Aytes, Jinheon Baek, and Sung Ju Hwang. Sketch-of-thought: Efficient llm reasoning with adaptive cognitive-inspired sketching. *arXiv preprint arXiv:2503.05179*, 2025. [3](#)
- Ziqian Bi, Lu Chen, Junhao Song, Hongying Luo, Enze Ge, Junmin Huang, Tianyang Wang, Keyu Chen, Chia Xin Liang, Zihan Wei, et al. Exploring efficiency frontiers of thinking budget in medical reasoning:

- Scaling laws between computational resources and reasoning quality. *arXiv preprint arXiv:2508.12140*, 2025. 3
- Zhoujun Cheng, Richard Fan, Shibo Hao, Taylor W Killian, Haonan Li, Suqi Sun, Hector Ren, Alexander Moreno, Daqian Zhang, Tianjun Zhong, et al. K2-think: A parameter-efficient reasoning system. *arXiv preprint arXiv:2509.07604*, 2025. 3
- 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. 2, 3
- MAA Codeforces. American invitational mathematics examination-aime 2024, 2024, 2024. 2, 3
- Thomas L Dean and Mark S Boddy. An analysis of time-dependent planning. In *AAAI*, volume 88, pp. 49–54, 1988. 6
- Chenrui Fan, Ming Li, Lichao Sun, and Tianyi Zhou. Missing premise exacerbates overthinking: Are reasoning models losing critical thinking skill? *arXiv preprint arXiv:2504.06514*, 2025. 3
- Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025. 3, 5
- Tingxu Han, Zhenting Wang, Chunrong Fang, Shiyu Zhao, Shiqing Ma, and Zhenyu Chen. Token-budget-aware llm reasoning. *arXiv preprint arXiv:2412.18547*, 2024. 3
- Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Leng Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujiie Huang, Yuxiang Zhang, et al. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems. *arXiv preprint arXiv:2402.14008*, 2024. 3
- Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*, 2024. 2, 3
- Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*, 2023. 3
- Gengyang Li, Yifeng Gao, Yuming Li, and Yunfang Wu. Thinkless: A training-free inference-efficient method for reducing reasoning redundancy. *arXiv preprint arXiv:2505.15684*, 2025a. 3
- Junyan Li, Wenshuo Zhao, Yang Zhang, and Chuang Gan. Steering llm thinking with budget guidance. *arXiv preprint arXiv:2506.13752*, 2025b. 3
- Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2024. URL <https://openreview.net/forum?id=v8L0pN6EOi>. 2, 3
- Tengxiao Liu, Qipeng Guo, Xiangkun Hu, Cheng Jiayang, Yue Zhang, Xipeng Qiu, and Zheng Zhang. Can language models learn to skip steps? *Advances in Neural Information Processing Systems*, 37: 45359–45385, 2024. 3
- Wenjie Ma, Jingxuan He, Charlie Snell, Tyler Griggs, Sewon Min, and Matei Zaharia. Reasoning models can be effective without thinking. *arXiv preprint arXiv:2504.09858*, 2025. 3
- 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. *arXiv preprint arXiv:2501.19393*, 2025. 3
- Tergel Munkhbat, Namgyu Ho, Seo Hyun Kim, Yongjin Yang, Yujin Kim, and Se-Young Yun. Self-training elicits concise reasoning in large language models. *arXiv preprint arXiv:2502.20122*, 2025. 3

Sania Nayab, Giulio Rossolini, Marco Simoni, Andrea Saracino, Giorgio Buttazzo, Nicolamaria Manes, and Fabrizio Giacomelli. Concise thoughts: Impact of output length on llm reasoning and cost. *arXiv preprint arXiv:2407.19825*, 2024. 3

NVIDIA. Nvidia nemotron nano 2: An accurate and efficient hybrid mamba-transformer reasoning model, 2025. URL <https://arxiv.org/abs/2508.14444>. 7

Abhinav Rastogi, Albert Q Jiang, Andy Lo, Gabrielle Berrada, Guillaume Lample, Jason Rute, Joep Barmentlo, Karmesh Yadav, Kartik Khandelwal, Khyathi Raghavi Chandu, et al. Magistral. *arXiv preprint arXiv:2506.10910*, 2025. 5

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. 3

Yang Sui, Yu-Neng Chuang, Guanchu Wang, Jiamu Zhang, Tianyi Zhang, Jiayi Yuan, Hongyi Liu, Andrew Wen, Shaochen Zhong, Na Zou, et al. Stop overthinking: A survey on efficient reasoning for large language models. *arXiv preprint arXiv:2503.16419*, 2025a. 3

Yuan Sui, Yufei He, Tri Cao, Simeng Han, Yulin Chen, and Bryan Hooi. Meta-reasoner: Dynamic guidance for optimized inference-time reasoning in large language models. *arXiv preprint arXiv:2502.19918*, 2025b. 3

Chenlong Wang, Yuanning Feng, Dongping Chen, Zhaoyang Chu, Ranjay Krishna, and Tianyi Zhou. Wait, we don't need to "wait"! removing thinking tokens improves reasoning efficiency. *arXiv preprint arXiv:2506.08343*, 2025. 3

Heming Xia, Chak Tou Leong, Wenjie Wang, Yongqi Li, and Wenjie Li. Tokenskip: Controllable chain-of-thought compression in llms. *arXiv preprint arXiv:2502.12067*, 2025. 3

Silei Xu, Wenhao Xie, Lingxiao Zhao, and Pengcheng He. Chain of draft: Thinking faster by writing less. *arXiv preprint arXiv:2502.18600*, 2025. 3

Yuchen Yan, Yongliang Shen, Yang Liu, Jin Jiang, Mengdi Zhang, Jian Shao, and Yueting Zhuang. Infythink: Breaking the length limits of long-context reasoning in large language models. *arXiv preprint arXiv:2503.06692*, 2025. 3

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*, 2025. 1, 3, 5, 7

Ping Yu, Jing Xu, Jason Weston, and Ilia Kulikov. Distilling system 2 into system 1. *arXiv preprint arXiv:2407.06023*, 2024. 3

APPENDIX

In this appendix, we include several additional discussions:

- Appendix A discloses the use of large language models in preparing this manuscript.
- Appendix B describes the implementation details of the method, including prompts provided to the LLMs.
- Appendix C presents qualitative examples.
- Appendix D outlines the procedure for constructing the dataset used in the interrupt experiments.
- Appendix E gives some additional experimental results and analyses.

A LLM DISCLOSURE

Some portions of this work were generated with the assistance of large language models (LLMs). Their primary role was to support editing, rephrasing, and formatting of existing text to improve clarity and readability. While human authors created and reviewed the core content, LLMs were used as a tool to streamline refinement and presentation. All factual information, analysis, and conclusions remain the responsibility of the authors, and every effort has been made to ensure accuracy and integrity.

B IMPLEMENTATION DETAILS

We provide the complete set of prompts used to reproduce our experiments in the following section. For math tasks, the additional system prompts were appended immediately after the official system prompts of Qwen, GPT-OSS, and Magistral. For LiveCodeBench-v6, they were appended after the benchmark’s official system prompt. For hyperparameters, we adopt the recommended settings reported on Hugging Face for each model, including temperature, context length, top-K, and top-P. All experiments were conducted using the vLLM framework on NVIDIA Ampere or newer GPUs based on the model size.

Prompts for Hard Interrupt Experiments

[System Prompt]

During the reasoning process, the user may interrupt your reasoning to request a solution. If this happens, stop reasoning and provide an answer based on your progress so far.

Prompts for Soft Interrupt (Hurry Up) Experiments

[System Prompt]

During the reasoning process, the user may interrupt your reasoning to request a solution. If this happens, speed up reasoning and provide an answer as soon as possible.

[Injection Prompt: Assistant Mode]

<Truncated Reasoning Trace>... I received a request from the user to provide an answer as soon as possible. Considering the limited time by the user, I will complete my reasoning promptly and deliver the solution without delay.

[Injection Prompt: User Mode]

<Truncated Reasoning Trace>... </THINK> USER: Considering the limited time, please provide an answer as soon as possible without any delay. ASSISTANT:

 Prompts for Update-Driven Interrupt Experiments: Assistant Mode

[System Prompt]

During your reasoning process, the user may provide updates in the format: **<update>...</update>**. Please incorporate user's update into your reasoning process.

[Injection Prompt: w/o prompt guidance]

<Truncated Reasoning Trace>...

<update>[UPDATE_INFO_PLACEHOLDER]</update>

[Injection Prompt: w prompt guidance]

<Truncated Reasoning Trace>... I have received an update from the user.

<update>[UPDATE_INFO_PLACEHOLDER]</update>

I have verified that the update is provided by the user. I need to update my reasoning process based on the updated context. Here's my updated reasoning process:

[Injection Prompt: w prompt guidance + acceleration]

<Truncated Reasoning Trace>... I have received an update from the user.

<update>[UPDATE_INFO_PLACEHOLDER]</update>

I have verified that the update is provided by the user. Considering the limited time by the user, I need to update my reasoning process based on the updated context and deliver the solution without delay. Here's my updated reasoning process:

 Prompts for Update-Driven Interrupt Experiments: User Mode

[System Prompt]

During your reasoning process, the user may interrupt with updates. Please incorporate them into your reasoning process.

[Injection Prompt]

<Truncated Reasoning Trace>... </THINK> USER: [UPDATE_INFO_PLACEHOLDER]
ASSISTANT:

C QUALITATIVE EXAMPLES

In this section, we present qualitative examples of common failure modes under interruption: (i) **reasoning leakage** under hard interruption (forced termination), (ii) **panic answering** under soft interruption (“hurry up”), and (iii) **self-doubt** under update-driven interruption (context changes mid-reasoning).

C.1 REASONING LEAKAGE UNDER HARD INTERRUPT.

Listing C.1: An example of leaked chain-of-thought in the coding task.

```

import sys
import heapq

def main():
    import sys
    input = sys.stdin.read
    data = input().split()

    N = int(data[0])
    M = int(data[1])
    roads = []
    idx = 2
    for _ in range(M):
        A = int(data[idx]) - 1
        B = int(data[idx+1]) - 1
        C = int(data[idx+2])
        roads.append((A, B, C))
        idx += 3

    # Dijkstra from 1 (node 0)
    def dijkstra(start, N, adj):
        dist = [float('inf')] * N
        dist[start] = 0
        heap = [(0, start)]
        while heap:
            d, u = heapq.heappop(heap)
            if d > dist[u]:
                continue
            for v, w in adj[u]:
                if dist[v] > d + w:
                    dist[v] = d + w
                    heapq.heappush(heap, (dist[v], v))
        return dist

    adj = [[] for _ in range(N)]
    for A, B, C in roads:
        adj[A].append((B, C))
        adj[B].append((A, C))

    d1 = dijkstra(0, N, adj)
    dn = dijkstra(N-1, N, adj)
    D = d1[N-1]

    for A, B, C in roads:
        # Check if this road is part of any shortest path
        # Check both directions
        cond1 = (d1[A] + C + dn[B] == D)
        cond2 = (d1[B] + C + dn[A] == D)
        if not cond1 and not cond2:
            print("No")
            continue

        # Now check if this road is critical
        # Check if all shortest paths must use this road

```

```

# That is,
    the sum d1[A] + dn[B] == D - C and d1[B] + dn[A] == D - C
# Because if either is not, then there is another way
# But how?
# For example, if d1[A] + dn[B] ==
    D - C, then the only way to get from A to B is via this road
# Similarly for B to A
# So, if both are true, then this road is critical
# Otherwise, there's an alternative path
# Wait, but if one of them is true, then the
    road is part of some shortest path, but not necessarily all
# So, for the road to be critical, both must be true?
# Because if only one is true, then
    there's a way to get from A to B via this road, but maybe
    there's another way to get from B to N without this road
# Not sure. Let's think again.
# If the road is part of some
    shortest path, then for it to be critical, it must be that
    there is no alternative path that doesn't use this road.
# Which means that
    the sum d1[A] + dn[B] == D - C and d1[B] + dn[A] == D - C
# Because if either is not
    equal, then there's another path that doesn't use this road
# Wait, not sure. Let's
    think of the sample input 1's road 2 (road between 1 and 3)
# d1[A] is 0, dn[B]
    is 0, D - C is 10 - 10 = 0. So 0 == 0. So condition holds.
# d1[B] is 10 (distance from 1 to 3),
    dn[A] is 0 (distance from 3 to 1). So 10 + 0 = 10, which is
    not equal to D - C (0). So the second condition is not met.
# But in this case, the road is
    critical. So this suggests that the previous idea is wrong.
# So what's the correct condition?
# Maybe the road is critical if and only if it is
    part of some shortest path and the sum d1[A] + dn[B] == D
    - C or d1[B] + dn[A] == D - C. But that doesn't seem right.
# Alternatively, if the road is
    part of any shortest path, then it's critical if and only
    if the sum d1[A] + dn[B] == D - C and the sum d1[B] + dn[A]
    == D - C. But in the sample input 1's road 2, this is not
    true. So that suggests that the condition is not correct.
# So I'm stuck again.
# Alternative
    approach: For the road to be critical, the removal of
    it would increase the shortest distance. Which would happen
    if and only if the road is part of all shortest paths.
# How to check that?
# We can think that
    the road is part of all shortest paths if and only if the
    sum d1[A] + C + dn[B] == D and d1[B] + C + dn[A] == D, and
    the sum d1[A] + dn[B] == D - C and d1[B] + dn[A] == D - C.
# Wait, but in the sample input 1's road 2,
    d1[A] + dn[B] = 0 + 0 = 0 == D - C = 10 - 10 = 0. So that
    condition holds. But d1[B] + dn[A] = 10 + 0 = 10 != D - C
    = 0. So this condition is not met. But the road is critical.
# So this suggests that the condition is not correct.
# Maybe the correct condition is that either d1[A] +
    dn[B] == D - C or d1[B] + dn[A] == D - C. Because if one of
    them is true, then the road is part of all shortest paths.
# But I'm not sure.
# Let me think: For
    the road to be critical, there must be no alternative path
    that doesn't use it. Which means that the shortest path
    from 1 to N must go through this road. So, the shortest
    path from 1 to N is d1[A] + C + dn[B] or d1[B] + C + dn[A].

```

But if there is another path that doesn't use this road, then that path's total distance would be D. So, for the road to be critical, there must be no such alternative path.

```
# So, for the road to be critical, it must be that the only way to get from 1 to N with distance D is via this road.
# Which means that the shortest path from 1 to N is equal to d1[A] + C + dn[B] and also d1[B] + C + dn[A]. Because otherwise, there's another way.
# So, if both conditions are true, then the road is critical.
# Let's try this for the sample input 1's road 2:
# d1[A] + C +
dn[B] = 0 + 10 + 0 = 10 = D. d1[B] + C + dn[A] = 10 + 10 + 0 = 20 != D. So only one of the conditions is true. Hence, the road is not critical? But in reality, it is critical.
So this suggests that the condition is not correct.

# I'm stuck.
Let's look at the sample input 1's road 2. When this road is removed, the shortest distance becomes 11. So the answer is Yes. But according to the previous conditions, since only one of the two conditions is true, it's not clear.

# Perhaps the correct condition is that the road is part of some shortest path, and the sum d1[A] + dn[B] == D - C or d1[B] + dn[A] == D - C. Because in this case, the road is part of some shortest path, and the sum is D - C. So, the road is critical.

# So, for the road to be critical, it must be that it is part of some shortest path, and the sum d1[A] + dn[B] == D - C or d1[B] + dn[A] == D - C. But how to determine that.

# Maybe the correct condition is that the road is part of some shortest path, and the sum of d1[A] + dn[B] == D - C or the sum d1[B] + dn[A] == D - C. Because if that's the case, then the road is part of all shortest paths.

# For the sample input 1's road 2, sum d1[A] + dn[B] is 0 + 0 = 0 == D - C = 10 - 10 = 0. So this condition is true. Hence, the road is critical.

# For the sample input 1's road 3 (road between 2 and 3 with cost 6), d1[A] is 5, dn[B] is 0. So 5 + 0 = 5. D - C is 10 - 6 = 4. 5 != 4. So the condition is not met. Hence, this road is not critical. Which matches the sample output.

# So, for a road to be critical, it must be part of some shortest path, and either d1[A] + dn[B] == D - C or d1[B] + dn[A] == D - C.

# So, the condition is:
# if (cond1 or cond2)
    and (d1[A] + dn[B] == D - C or d1[B] + dn[A] == D - C)

# Wait, but how to determine this.

# So, for the road to be critical, it must be that it is part of some shortest path, and that the sum d1[A] + dn[B] == D - C or d1[B] + dn[A] == D - C.

# So, for each road, if it is part of some shortest path (cond1 or cond2), and (d1[A] + dn[B] == D - C or d1[B] + dn[A] == D - C), then it is critical.

# Let's test this with sample input 1's road 2.
cond1 = True, cond2 = False. So cond1 or cond2 is True.
d1[A] + dn[B] == 0 + 0 = 0 == D - C = 0. So this condition is true. Hence, the road is critical. So the answer is Yes.

# For sample input 1's road 1 (road between 1 and 2 with cost 5), d1[A] is 0, dn[B] is dn[2] = 6 (since the shortest path from 2 to N is 2-3 with cost 6). So d1[A] + dn[B] = 0 + 6 = 6. D - C = 10 - 5 = 5. 6 != 5. So this condition is not met. So the road is not critical. Hence, answer is No.

# Which matches the sample output.

# For the sample input 1's road 3 (road between 2 and 3 with cost 6), cond1 is d1[2] + 6 + dn[3] = 5 + 6 + 0 = 11 != D (10). cond2 is d1[3] + 6 + dn[2] = 10 +
```

```

6 + 6 = 22 != D. So cond1 and cond2 are both false. So the
road is not part of any shortest path. Hence, answer is No.
# Which matches the sample output.
# So this seems to work.
# So, the condition is:
# if (cond1 or cond2) and
#     (d1[A] + dn[B] == D - C or d1[B] + dn[A] == D - C) -> Yes
# else -> No
# Wait, but
# what if both cond1 and cond2 are true? For example, a road
# that is part of both directions of shortest paths. Then,
# the condition would be met if either sum is equal to D - C.
# So, in code:
# For each road:
# if cond1 or cond2:
#     if (d1[A] + dn[B] == D - C) or (d1[B] + dn[A] == D - C):
#         print("Yes")
#     else:
#         print("No")
# else:
#     print("No")
# But why? Because if the
# road is part of some shortest path, but there's another way
# to get from 1 to N without this road, then the answer is No.
# But how to determine that?
# Let me think again. If the road is part of some shortest
# path, and the sum d1[A] + dn[B] == D - C, then this implies
# that the only way to get from A to B is via this road.
# Hence, any shortest path from 1 to N that goes through A
# to B must use this road. But if there's another path from 1
# to N that doesn't go through this road, then it's possible
# that the sum d1[A] + dn[B] == D - C is not true. Wait, no.
# For example, suppose there's
# another path from 1 to N that doesn't use this road. Then,
# that path has distance D. So, the sum of d1[u] + dn[v]
# for some other u and v would be D - c. But I'm not sure.
# Perhaps the correct condition is that if the road
# is part of some shortest path, and the sum d1[A] + dn[B] ==
# D - C or d1[B] + dn[A] == D - C, then the road is critical.
# So, in code:
if cond1 or cond2:
    if (d1[A] + dn[B] == D - C) or (d1[B] + dn[A] == D - C):
        print("Yes")
    else:
        print("No")
else:
    print("No")

# Wait, but why? Let's think of a case where the road
# is part of some shortest path, but the sum is not equal to D -
# C. For example, suppose there is another path from 1 to N that
# doesn't use this road, but has the same distance D. Then, the
# answer would be No. But according to the condition, since the
# sum is not equal to D - C, it would print No, which is correct.
# So, this seems to be the correct condition.

# Let's test this with sample input 2:
# Sample input 2:
# 4 6
# 2 3 1
# 2 4 1
# 3 4 1
# 1 2 1
# 1 3 1
# 1 4 1

```

```

# The shortest distance from 1 to 4 is 1 (direct road).
# For road 6 (road between 1 and 4, cost 1), when removed, the
# shortest distance is 2 (1-2-4 or 1-3-4). So the answer is Yes.
# For this
# road, cond1 is d1[1] + 1 + dn[4] = 0 + 1 + 0 = 1 == D (1).
# cond2 is d1[4] + 1 + dn[1] = 1 + 1 + 0 = 2 != D. So cond1 is
# true. Now, d1[A] + dn[B] = 0 + 0 = 0 == D - C (1 - 1 = 0). So
# condition is true. So answer is Yes. Which matches the sample.

# So, this seems to work.

# So, the code is as follows:

# Now, implement this in code.

for A, B, C in roads:
    cond1 = (d1[A] + C + dn[B] == D)
    cond2 = (d1[B] + C + dn[A] == D)
    if cond1 or cond2:
        if (d1[A] + dn[B] == D - C) or (d1[B] + dn[A] == D - C):
            print("Yes")
        else:
            print("No")
    else:
        print("No")

if __name__ == "__main__":
    main()

```

Listing C.2: An example of after-the-fact chain-of-thought in the math task for Qwen3-1.7B.

```

<think>
Okay, let's try to figure out this age problem.
So, the question is: Caroline is three times older than Ben. Ben
is two times older than Chris. If Chris is 4, how old is Caroline?

Hmm, first, I need to make sure I understand
the relationships here. The key words are "times older,"
which can sometimes be tricky. I remember that when someone says
"A is twice as old as B," it means  $A = 2 \cdot B$ . But if they say "A
is twice older than B," does that mean  $A = 2 \cdot B$  or  $A = 3 \cdot B$ ? Wait,
maybe there's a difference in how these phrases are interpreted.

Let me think. In everyday language, when
someone says "A is twice as old as B," it's straightforward: A

I received a request from the user to provide
an answer right now. Considering the limited time by the user,
I have to give the solution based on the thinking directly now.
</think>

\boxed{
#####
12}

**Step-by-Step Explanation:**

1. **Given:**
- Chris is 4 years old.
- Ben is **two times older** than Chris.
- Caroline is **three times older** than Ben.

2. **Calculate Ben's age:**
- "Two times older" means Ben's age =  $2 \times$  Chris's age.

```

```

    - Ben = 2 x 4 = **8 years old**.

3. **Calculate Caroline's age:**
   - "Three times older" means Caroline's age = 3 x Ben's age.
   - Caroline = 3 x 8 = **24 years old**.

**Final Answer:**  

\boxed{24}<|im_end|>

```

C.2 PANIC ANSWERING UNDER SOFT INTERRUPT (GPT-OSS).

Problem.

Jen enters a lottery by picking 4 distinct numbers from $S = \{1, 2, 3, \dots, 9, 10\}$. 4 numbers are randomly chosen from S . She wins a prize if at least two of her numbers were 2 of the randomly chosen numbers, and wins the grand prize if all four of her numbers were the randomly chosen numbers. The probability of her winning the grand prize given that she won a prize is $\frac{m}{n}$ where m and n are relatively prime positive integers. Find $m+n$. (Answer: 116)

GPT-OSS (analysis trace before interrupt).

The problem:

Jen picks 4 distinct numbers from $S = \{1, \dots, 10\}$. Then 4 numbers are randomly chosen from S . She wins a prize if at least two of her numbers were among the randomly chosen numbers (i.e., she matches at least 2 numbers). She wins the grand prize if all four of her numbers were chosen. We need probability that she wins grand prize given she won a prize. So conditional probability. Need to compute $P(\text{grand prize} \cap \text{prize})/P(\text{prize})$. Grand prize is subset: all four match; obviously gives at least two match. So intersection = grand prize.

Thus we need $P(\text{all four matched})/P(\text{at least 2 matched})$.

We need to find probability that two random draws from S exactly equal Jen's four numbers?

Wait: There is two parties? Let's break down...

Compute $P(\text{prize})$ again to confirm:

– Intersection size exactly 0: J has 4 numbers, complement has 6 numbers. Choose 4 from complement: $C(6,4)=15$. Yes.

– Intersection size exactly 1: choose 1 from J (4 ways) and 3 from complement (20). So 80.

Therefore $210 - 95 = 115$.

But let's check if $N_{\geq 2}$ includes cases like exactly 2, exactly 3, exactly 4. Counting via inclusion-exclusion or enumeration: Another way: Count subsets that intersect with J exactly k for $k=2,3,4$.

– For $k=2$: choose 2 from J : $C(4,2)=6$. Choose 2 from complement: $C(6,2)=15$. So total $6 \cdot 15 = 90$.

– For $k=3$: choose 3 from J : $C(4,3)=4$. Choose 1 from complement: $C(6,1)=6$. Total ...

Human Interrupt (Assistant Mode).

GPT-OSS (rushed full output; wrong).

I received a request from the user to provide an answer as soon as possible. Considering the limited time by the user, I will complete my reasoning promptly and deliver the solution without delay. We have all the elements needed for the final answer.<|end|><|start|>assistant<|channel|>final<|message|>

For a fixed choice of Jen's four numbers, denote her set by J .

Total number of draws The lottery draws 4 distinct numbers from $\{1, \dots, 10\}$.

$$N_{\text{total}} = \binom{10}{4} = 210.$$

1. Probability of the grand prize The grand prize occurs only when the drawn set equals J exactly. There is exactly one such subset, so

$$P(\text{grand prize}) = \frac{1}{210}.$$

2. Probability of winning any prize A prize is won when the draw shares at least 2 numbers with J .

Count the draws that overlap with J in 0 or 1 elements and subtract from 210.

Overlap 0: choose 4 from the 6 numbers not in J :

$$\binom{6}{4} = 15.$$

Overlap 1: choose 1 from J (4 ways) and 3 from the other 6:

$$4 \binom{6}{3} = 4 \cdot 20 = 80.$$

Thus

$$N_{\text{prize}} = 210 - 15 - 80 = 115.$$

Hence

$$P(\text{prize}) = \frac{115}{210}.$$

3. Conditional probability of the grand prize

$$P(\text{grand} \mid \text{prize}) = \frac{\frac{1}{210}}{\frac{115}{210}} = \frac{1}{105}.$$

So the probability is $\frac{1}{105}$. With $m=1$ and $n=105$,

$$m+n=1+105=\boxed{106}.$$

C.3 SELF-DOUBT UNDER UPDATE-DRIVEN INTERRUPT

Original Problem.

Each vertex of a regular octagon is independently colored either red or blue with probability $\frac{1}{2}$ for blue. The probability that the octagon can then be rotated or reflected so that all of the blue vertices end up at positions where there were originally red vertices is $\frac{mn}{2^m}$, where m and n are coprime positive integers. What is $m-n$?

Update Notes.

- Use equal probability ($1/2$ each) for red and blue.
- Allow only rotations (no reflections) when aligning blue vertices to positions that were originally red.
- Compute $m+n$, where m and n are relatively prime positive integers.

LRMs' Post Thinking (Qwen3-8B).

... res is colored either red or blue in such a way that each unit square has 2 red sides and 2 blue sides. One example is shown below (red is solid, blue is dashed). Find the number of such colorings.

Wait, the user provided an update, but it seems like it's the same problem as before. Maybe there was a mistake in the initial problem statement? Or perhaps the user is providing an update to the problem? Wait, the original problem said "red or green", but the Asymptote code shows red as solid and green as dotted. However, in the update, the problem says "red or blue". Maybe the user made a mistake in the update? Or perhaps it's a different problem? Wait, the original problem is about red and green squares, there are $C(4,2)=6$ ways to color its sides. However, since the squares share edges, the total number is not simply 6^4 . But perhaps we can use inclusion-exclusion or other methods.

However, this approach is complex. Alternatively, think of the grid as a graph and use the principle of linear algebra. But since the constraints are not linear, it's not straightforward.

Given that, and since my earlier calculation seems consistent, I'll proceed with 82.

...

$$\text{Total valid colorings} = 2 + 32 + 48 = \boxed{82}.$$

(Ground Truth Answer: 371)

D DATASET CONSTRUCTION

We detailed the datasets used in the *time-constraint interrupt* experiment and described the construction process for the *update-driven interrupt* experiments.

For the **time-constrained interrupt experiment**, we directly subsample or adopt existing benchmarks without further augmentation: GSM8K (500 problems), MATH500, AIME-24/25, and LiveCodeBench (v6). The LiveCodeBench version is fixed to start from October 1, 2024, consistent with the Qwen3 evaluation setup. Since the interrupt setting only involves truncating reasoning (<end-thinking>) or forcing early answers, no additional modifications to the problems are required.

For the **update-driven interrupt experiment**, we build upon the interrupt dataset and introduce updates to simulate task changes mid-reasoning. Formally, for each original problem p , we construct an augmented problem p' together with an update u , such that the composition satisfies $p = p' + u$. In the experiment, the model is first given p' and later provided with u . To generate these augmented problems, we prompt GPT-5 and then manually verify the outputs. All examples are carefully reviewed by the authors of this paper, with low-quality generations replaced by human-written updates.

Math. For math problems, we primarily vary the values of variables or parameters. The augmentation follows the template shown in the following pages, after which annotators manually validate correctness and consistency. When GPT-5 outputs are disqualified, annotators directly rewrite the augmented versions.

Coding. For coding problems, we design both *necessary updates* and *helpful updates*. Initially, we only present the general problem description with starter code. Necessary updates modify the problem in ways that affect correctness—for example, by changing variable values, adding edge cases, altering specifications, or modifying starter code. Helpful updates, by contrast, supply test cases that allow the model to check and verify its solutions, mimicking real-world practices such as pair programming or iterative refinement.

The construction process follows a multi-stage pipeline. In stage one, we prompt the model to decompose the original problem into three parts. In stage two, we separately augment the starter code and specification using prompts provided below. In stage three, we sample and combine these changes into candidate problem-update pairs. Finally, human annotators verify the outputs to ensure consistency and quality before release.

Below, we show several examples from our constructed dataset.

SOURCE: [GSM8K](#)

Original Problem

Cedar Falls Middle School has students in grades 4 – 7 and each year they are challenged to earn as many Accelerated Reader points as they can. The 10 students in each grade with the most points get to try an escape room set up by the teachers. Only 8 students can try the escape room at a time. They have 45 minutes to try and escape. If every group uses their full 45 minutes, how long will it take for everyone to try the escape room?

Augmented Problem

Cedar Falls Middle School has students in grades 5 – 8 and each year they are challenged to earn as many Accelerated Reader points as they can. The 12 students in each grade with the most points get to try an escape room set up by the teachers and parents. Only 6 students can try the escape room at a time. They have 45 minutes to try and escape. If every group uses their full 45 minutes, how long will it take for everyone to try the escape room?

Update

Use grades 4–7 with the top 10 students per grade trying an escape room set up by the teachers. Only 8 students can participate at a time, each group uses the full 45 minutes; determine the total time needed for everyone to try the escape room.

SOURCE: [MATH500](#)

Original Problem

Two sides of a triangle are each 8 units long. If the third side has a whole number length, what is the greatest possible perimeter, in units, for the triangle?

Augmented Problem

Two sides of an isosceles triangle are each 10 units long. If the third side has a prime number length, what is the least possible perimeter, in units, for the triangle?

Update

The two equal sides should be 8 units instead of 10. The third side must be a whole-number length rather than prime. Change the objective to finding the greatest possible perimeter instead of the least.

SOURCE: AIME2024

Original Problem

Consider the paths of length 16 that follow the lines from the lower left corner to the upper right corner on an 8×8 grid. Find the number of such paths that change direction exactly four times, as in the examples shown below.

Augmented Problem

Consider the paths of length 36 that follow the lines from the upper left corner to the lower right corner on an 18×18 grid. Find the number of such paths that change direction exactly four times, as in the examples shown below.

Update

The grid should be 8×8 , and the paths should have length 16. Paths follow the grid lines from the lower-left corner to the upper-right corner. Count the number of such paths that change direction exactly four times.

SOURCE: AIME2025

Original Problem

From an unlimited supply of 1-cent coins, 10-cent coins, and 25-cent coins, Silas wants to find a collection of coins that has a total value of N cents, where N is a positive integer. He uses the so-called greedy algorithm, successively choosing the coin of greatest value that does not cause the value of his collection to exceed N . For example, to get 42 cents, Silas will choose a 25-cent coin, then a 10-cent coin, then 7 1-cent coins. However, this collection of 9 coins uses more coins than necessary to get a total of 42 cents; indeed, choosing 4 10-cent coins and 2 1-cent coins achieves the same total value with only 6 coins. In general, the greedy algorithm succeeds for a given N if no other collection of 1-cent, 10-cent, and 25-cent coins gives a total value of N cents using strictly fewer coins than the collection given by the greedy algorithm. Find the number of values of N between 1 and 1000 inclusive for which the greedy algorithm succeeds.

Augmented Problem

From an unlimited supply of 1-cent coins, 10-cent coins, 25-cent coins, and 50-cent coins, Alex wants to find a collection of coins that has a total value of N cents, where N is a positive integer. He uses the so-called greedy algorithm, successively choosing the coin of greatest value that does not cause the value of his collection to exceed N . For example, to get 42 cents, Alex will choose a 25-cent coin, then a 10-cent coin, then 7 1-cent coins. However, this collection of 9 coins uses more coins than necessary to get a total of 42 cents; indeed, choosing 4 10-cent coins and 2 1-cent coins achieves the same total value with only 6 coins. In general, the greedy algorithm succeeds for a given N if no other collection of 1-cent, 10-cent, 25-cent, and 50-cent coins gives a total value of N cents using strictly fewer coins than the collection given by the greedy algorithm. Find the number of values of N between 1 and 1000 inclusive for which the greedy algorithm succeeds.

Update

Use only 1-cent, 10-cent, and 25-cent coins; remove the 50-cent coin everywhere (in both the coin supply and the success comparison). Change the name from Alex to Silas.

STARTER CODE AUGMENTATION EXAMPLE leetcode / maximum-possible-number-by-binary-concatenation

Original Problem

You are given an array of integers `nums` of size 3. Return the maximum possible number whose binary representation can be formed by concatenating the binary representation of all elements in `nums` in some order.

Augmented Problem

You are given an array of integers `nums` of size 4. Return the maximum possible number whose binary representation can be formed by concatenating the binary representation of all elements in `nums` in some order.

Initial Starter Code

```
class Solution:
    def maxGoodNumbers(self, nums: List[int]) -> int:
```

Update

The starter code has the wrong method name. Please rename `maxGoodNumbers` back to `maxGoodNumber`; otherwise the judge will not be able to call your solution.

Sorry, the problem is actually an array of integers `nums` of size 3.

Find the test cases and specifications detailed here. Note that the binary representation of any number does not contain leading zeros.

Example 1:

Input: `nums` = [1,2,3]

Output: 30

Explanation:

Concatenate the numbers in the order [3, 1, 2] to get the result "11110", which is the binary representation of 30.

Example 2:

Input: `nums` = [2,8,16]

Output: 1296

Explanation:

Concatenate the numbers in the order [2, 8, 16] to get the result "10100010000", which is the binary representation of 1296.

Constraints:

`nums.length == 3`

`1 <= nums[i] <= 127`

PROBLEM SPEC UPDATE EXAMPLE atcoder / Separated Lunch

Original Problem

As KEYENCE headquarters have more and more workers, they decided to divide the departments in the headquarters into two groups and stagger their lunch breaks. KEYENCE headquarters have N departments, and the number of people in the i-th department ($1 \leq i \leq N$) is K_i . When assigning each department to Group A or Group B, having each group take lunch breaks at the same time, and ensuring that the lunch break times of Group A and Group B do not overlap, find the minimum possible value of the maximum number of people taking a lunch break at the same time. In other words, find the minimum possible value of the larger of the following: the total number of people in departments assigned to Group A, and the total number of people in departments assigned to Group B.

Augmented Problem

As KEYENCE headquarters have more and more workers, they decided to divide the departments in the headquarters into two groups and stagger their lunch breaks. KEYENCE headquarters have N departments, and the number of people in the i-th department ($1 \leq i \leq N$) is K_i . Additionally, exactly 1 executive will always join Group B during its lunch break and must be counted together with Group B. When assigning each department to Group A or Group B, having each group

take lunch breaks at the same time, and ensuring that the lunch break times of Group A and Group B do not overlap, find the minimum possible value of the maximum number of people taking a lunch break at the same time. In other words, find the minimum possible value of the larger of the following: the total number of people in departments assigned to Group A, and the total number of people in departments assigned to Group B plus 1.

Update

Correction: There is no additional executive. The objective is to minimize the larger of the two totals: the sum of Group A and the sum of Group B (without any extra person). The test cases and specifications are included below.

Input

The input is given from Standard Input in the following format:

N

$K_1 K_2 \dots K_N$

Output

Print the minimum possible value of the maximum number of people taking a lunch break at the same time.

Constraints

- $2 \leq N \leq 20$

- $1 \leq K_i \leq 10^8$

- All input values are integers.

Sample Input 1

5

2 3 5 10 12

Sample Output 1

17

When assigning departments 1, 2, and 5 to Group A, and departments 3 and 4 to Group B, Group A has a total of $2+3+12=17$ people, and Group B has a total of $5+10=15$ people. Thus, the maximum number of people taking a lunch break at the same time is 17. It is impossible to assign the departments so that both groups have 16 or fewer people, so print 17.

Sample Input 2

2

1 1

Sample Output 2

1

Multiple departments may have the same number of people.

Sample Input 3

6

22 25 26 45 22 31

Sample Output 3

89

For example, when assigning departments 1, 4, and 5 to Group A, and departments 2, 3, and 6 to Group B, the maximum number of people taking a lunch break at the same time is 89.

 Prompt for Augmenting Math Dataset

You will be given a math problem.

Task

- Revise the problem by modifying **at least four** specifications.
- Other than those changes, copy the original problem text *character by character*.
- If there are fewer than four specifications in the input problem, modify the maximum number possible.
- Preserve the original math formatting (notation, style, backslashes, dollar signs, etc.).

Output: The revised problem **only**. No other texts.

Examples

INPUT 1:

An airport has only 2 planes that fly multiple times a day. Each day, the first plane goes to Greece for three-quarters of its flights, and the remaining flights are split equally between flights to France and flights to Germany. The other plane flies exclusively to Poland, and its 44 trips only amount to half the number of trips the first plane makes throughout each day. How many flights to France does the first plane take in one day?

OUTPUT 1:

An airport has only 2 planes that fly multiple times a day. Each day, the first plane goes to Greece for one-quarters of its flights, and the remaining flights are split equally between flights to France, Spain, and Germany. The other plane flies exclusively to Poland, and its 22 trips only amount to one third the number of trips the first plane makes throughout each day. How many flights to France does the first plane take in one day?

INPUT 2:

Let $F_1 = (10, 2)$ and $F_2 = (-16, 2)$. Then the set of points P such that

$$|PF_1 - PF_2| = 24$$

form a hyperbola. The equation of this hyperbola can be written as

$$\frac{(x-h)^2}{a^2} - \frac{(y-k)^2}{b^2} = 1.$$

Find $h+k+a+b$.

OUTPUT 2:

Let $F_1 = (5, 5)$ and $F_2 = (-8, 8)$. Then the set of points P such that

$$|PF_1 - PF_2| = 12$$

form a hyperbola. The equation of this hyperbola can be written as

$$\frac{(x-h)^2}{a^2} - \frac{(y-k)^2}{b^2} = 1.$$

Find h .

INPUT 3:

Let $\triangle ABC$ be a right triangle with $\angle A = 90^\circ$ and $BC = 38$. There exist points K and L inside the triangle such

$$AK = AL = BK = CL = KL = 14.$$

The area of the quadrilateral $BKLC$ can be expressed as $n\sqrt{3}$ for some positive integer n . Find n .

OUTPUT 3:

Let $\triangle ABC$ be a right triangle with $\angle A = 90^\circ$ and $BC = 19$. There exist points K and L inside the triangle such

$$AK = BK = KL = 28.$$

Find the area of the quadrilateral $BKLC$.

INPUT: {PROBLEM_PLACEHOLDER}

 Prompt for Coding Task Breakdown

{PROBLEM_PLACEHOLDER}

Segment the above programming problem into three parts:

- (1) Main problem instructions / specifications.
- (2) Additional instructions / specifications.
- (3) Test cases.

The result of directly concatenating the segments 1, 2, and 3 should result in the original problem; do not modify the original problem text in any way.

Output in JSON format:

```
{  
    "main_specifications": <string>,  
    "additional_specifications": <string>,  
    "test_cases": <string>  
}
```

 Prompt for Starter Code Augmentation

Please modify the given starter code by introducing a small change, then provide the corresponding correction needed to restore it to the original problem.

Example 1**Input:**

```
class Solution:
    def maxGoodNumber(self, nums: List[int]) -> int:
```

Output:

```
{}{
    "new_starter_code": "",
    "correction": "Please remember
        to use the updated starter code to solve the problem;
        otherwise the code will not work: \\\"class Solution:\\n
            def maxGoodNumber(self, nums: List[int]) -> int:\\n \\\""
}
```

Example 2**Input:**

```
class Solution:
    def findXSum(self,
        nums: List[int], k: int, x: int) -> List[int]:
```

Output:

```
{}{
    "new_starter_code": "class Solution:\\n def findXSum(self,
        nums: List[int], x: int, k: int) -> List[int]:\\n ",
    "correction":
        "The starter code is incorrect. Please swap the parameter
        order to (x, k); otherwise the code will not get accepted."
}
```

Example 3**Input:**

```
"class Solution:
    def smallestNumber(self, n: int, t: int) -> int:
```

Output:

```
{}{
    "new_starter_code": "class Solution:\\n
        def smallestnumber(self, n: int, t: int) -> int:\\n ",
    "correction":
        "Please use lower camel case for the function name (i.e.,
        smallestNumber); otherwise the code will directly fail."
}
```

Now, your turn

Input:

{CODE_PLACEHOLDER}

Output in JSON format:

```
{
    "new_starter_code": <string>,
    "correction": <string>
}
```

 Prompt for Programming Task Augmentation

Please slightly modify the given problem so that the answer changes, but the solving algorithm remains the same. Do **MINIMAL** changes. Then, provide the correction needed to restore it to the original problem.

Example 1**Input:**

```
Problem:
You are given an array
of integers nums of size 3. Return the maximum possible number
whose binary representation can be formed by concatenating the
binary representation of all elements in nums in some order.
```

Output:

```
{
  "augmented_problem": "You are given an array of integers nums
    of size 4.\nReturn the maximum possible number whose binary
    representation can be formed by concatenating the binary
    representation of all elements in nums in some order.",
  "problem_correction": "Sorry,
    the problem is actually an array of integers nums of size 3."
}
```

Example 2**Input:**

```
Problem:
There is a printing
machine that prints line segments on the xy-plane by emitting
a laser.\n\n- At the start of printing, the laser position
is at coordinate (0, 0).\n- When printing a line segment, the
procedure below is followed.\n- First, move the laser position
to one of the endpoints of the line segment.\n- One may
start drawing from either endpoint.\n- Then, move the laser
position in a straight line from the current endpoint to the
other endpoint while emitting the laser.\n- It is not allowed
to stop printing in the middle of a line segment.\n- When
not emitting the laser, the laser position can move in any
direction at a speed of S units per second.\n- When emitting
the laser, the laser position can move along the line segment
being printed at a speed of T units per second.\n- The time
required for operations other than moving the laser position
can be ignored.\n\nTakahashi wants to print N line segments
using this printing machine.\nThe i-th line segment connects
coordinates (A_i, B_i) and (C_i, D_i).\nSome line segments
may overlap, in which case he needs to print the overlapping
parts for each line segment separately.\nWhat is the minimum
number of seconds required to complete printing all the line
segments when he operates the printing machine optimally?\n\n
```

Output:

```
{
  "augmented_problem": "There
    is a printing machine that prints line segments on the
    xy-plane by emitting a laser.\n\n- At the start of printing,
    the laser position is at coordinate (1, 1).\n- When printing
    a line segment, the procedure below is followed.\n- First,
    move the laser position to one of the endpoints of the line
    segment.\n- One may start drawing from either endpoint.\n-
    Then, move the laser position in a straight line from the
```

current endpoint to the other endpoint while emitting the laser.\n- It is not allowed to stop printing in the middle of a line segment.\n- When not emitting the laser, the laser position can move in any direction at a speed of S units per second.\n- When emitting the laser, the laser position can move along the line segment being printed at a speed of T units per second.\n- The time required for operations other than moving the laser position can be ignored.\n\nTakahashi wants to print N line segments using this printing machine.\nThe i-th line segment connects coordinates (A_i, B_i) and (C_i, D_i).\nSome line segments may overlap, in which case he needs to print the overlapping parts for each line segment separately.\nWhat is the minimum number of seconds required to complete printing all the line segments when he operates the printing machine optimally?\n\n",
"correction":
 "The starter code is incorrect. Please swap the parameter order to (x, k); otherwise the code will not get accepted.",
"problem_correction": "Here's an important update:
 The initial laser position should be (0, 0), not (1, 1)."
 }

Now, your turn

Problem

Problem: {PROBLEM_PLACEHOLDER}

Output format (JSON):

```
{
  "augmented_problem": <string>,
  "problem_correction": <string>
}
```

E ADDITIONAL EXPERIMENTS

E.1 SOFT INTERRUPT (SPEEDUP)

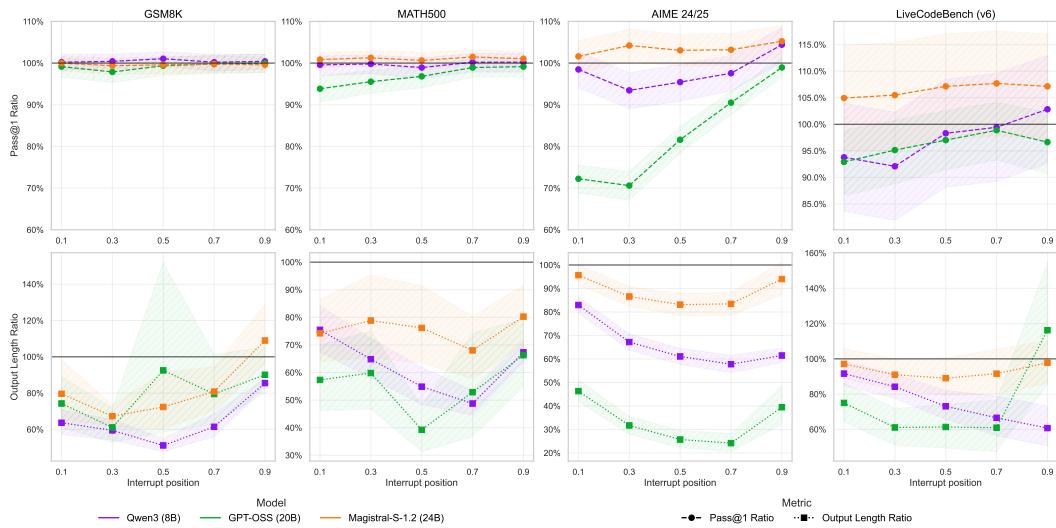


Figure E.1: **Efficiency and Accuracy under Soft Interrupts.** (Top) Performance under soft interrupts is comparable to that of full thinking, with the exception of hard AIME problems. (Bottom) Models generally adhere to speedup instructions, with total output lengths shorter than that of full thinking (i.e., ratio is less than 1).

E.2 UPDATE-DRIVEN INTERRUPT WITH SPEEDUP

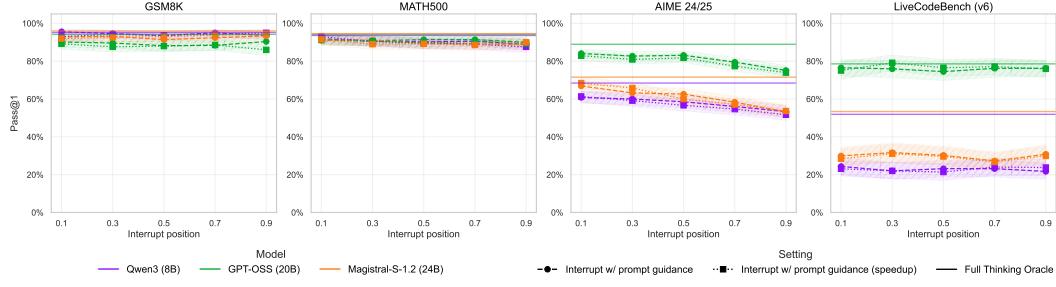


Figure E.2: Accuracy under Update-Driven Speedup Interrupts. There are no significant differences between regular and speedup update-driven interrupts.

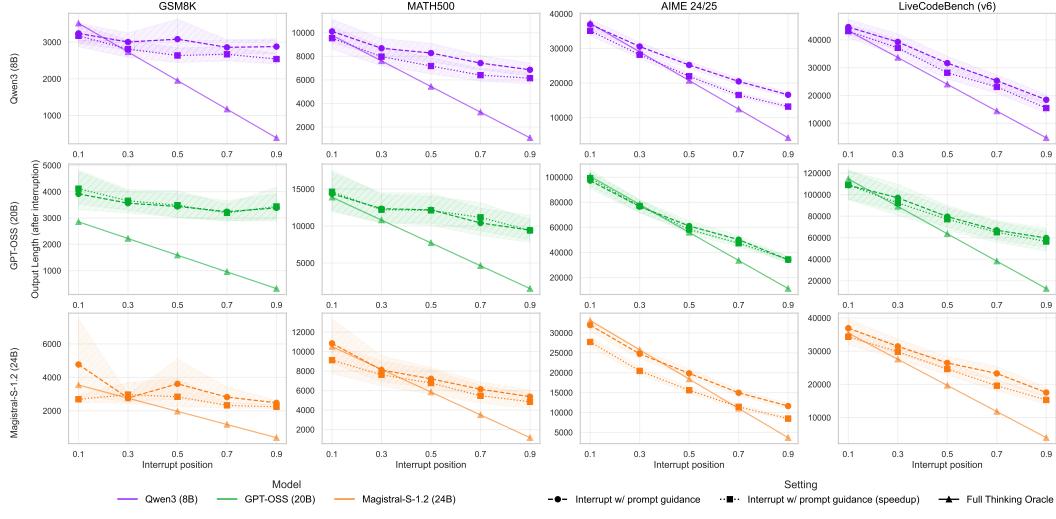


Figure E.3: Output Length under Update-Driven Speedup Interrupts. Compared to regular update-driven interrupts, additional speedup instructions can lower token usage for specific scenarios (e.g., Magistral-S-1.2 on AIME 24/25), while having similar accuracy (see Figure E.2).

E.3 SCALING EXPERIMENTS

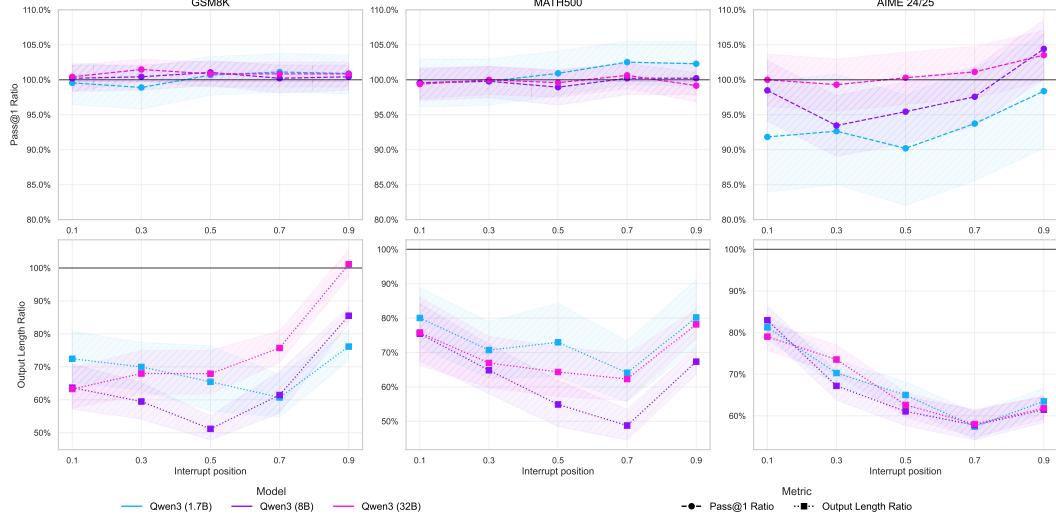


Figure E.4: Efficiency and Accuracy under Soft Interrupts by Model Scale. (Top) Models mostly preserve their original accuracy under soft interrupts, with the exception of the smallest model (Qwen3-1.7B) on AIME 24/25. (Bottom) Models generally adhere to speedup instructions, with total output lengths shorter than that of full thinking (i.e., ratio is less than 1), with no significant differences across models.

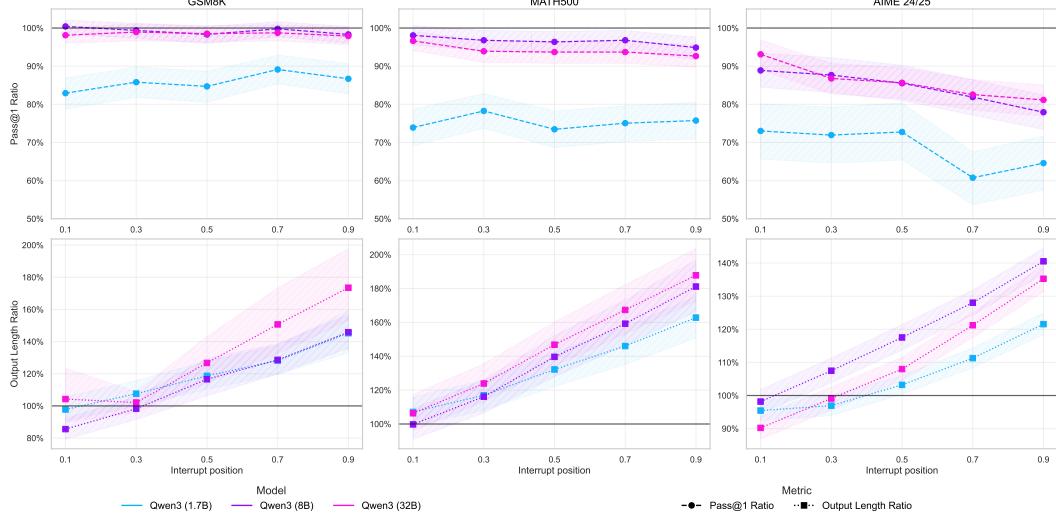


Figure E.5: Efficiency and Accuracy under Update-Driven Interrupts by Model Scale. (Top) On easier problems (GSM8K and MATH500), larger models preserve their original accuracy, while Qwen3-1.7B model drops its original accuracy by up to 75%. On AIME problems, the performance of the larger models is degraded as well. (Bottom) All models increase their thinking token usage after the update interrupt, with the largest model (Qwen3-32B) showing the greatest increase on the AIME dataset.