WHEN TASKS COMPETE FOR TOKENS: IMPLICIT BUDGETING IN LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Modern language models are increasingly asked to handle several tasks within a single prompt. Doing so requires solving each task correctly while integrating the resulting answers into a coherent response. In this paper, we identify a fundamental limitation in this process, where models make substantially more errors when tasks are composed, even when they can solve every constituent task reliably in isolation. This degradation cannot be fully explained by the accumulation of independent errors. Across both open source and frontier models such as Claude Opus 5, our findings instead point to an implicit thinking budget underlying this effect. In particular, under composition, models use substantially fewer tokens per task than in isolation, even when the full sequence remains well within the context window. Tasks that require longer solution traces in isolation are consequently more *fragile* under composition, i.e., models make more errors on prompts containing these tasks. This fragility can also be exploited for efficiency. By appropriately grouping fragile tasks, we obtain $\sim$40% token savings across a range of math and linguistic tasks while also preserving accuracy.

1 INTRODUCTION

A single prompt often asks a language model to do several things at once, such as answering a handful of questions, solving a set of math problems, or following a list of instructions before writing one response. Benchmarks, however, still test models one task at a time (Rein et al., 2023; Hendrycks et al., 2021; Gema et al., 2025). As scores on these benchmarks saturate, they may overstate how robust the underlying capabilities are. We ask *whether a model that reliably solves a task on its own can still solve it when other tasks share the prompt*.

Prior work has found that models degrade under composition, usually by combining familiar reasoning steps (Press et al., 2023; Dziri et al., 2023), or instructions (Harada et al., 2025; Vasileva, 2026) into more complex or unseen tasks. Because composition in these settings also changes the structure and token demands of the task, it remains unclear 1) whether simply presenting multiple tasks together is sufficient to cause degradation, and 2) what mechanism drives that degradation. We therefore systematically isolate compositional effects in a minimal setting where for each model, we first identify tasks it solves correctly in isolation, establishing that the requisite capability is present. We then leave those tasks unchanged and simply concatenate them within a single prompt. This design separates degradation under increased task load from baseline difficulty on the constituent tasks.

Across frontier proprietary models such as Claude Opus 5, as well as state-of-the-art open models, we find substantial performance degradation as more previously solved tasks are concatenated within the same prompt. The drop in accuracy is substantially larger than what the constituent tasks' isolated error rates would predict, and is consistent across both mathematical and linguistic domains. We characterize this degradation along two dimensions. 1) Failures concentrate on a subset of *fragile tasks* that, for a given model, remains stable across sampling noise and composition sizes. These tasks also require longer reasoning traces in isolation. 2) As more tasks are presented together, models systematically reduce the reasoning tokens devoted to each task, even though no explicit per-task token constraint is imposed. This reduction follows a clear scaling pattern, with reasoning per task decreasing approximately as a *power law* with composition size. Together, these findings reveal an implicit thinking budget under composition, under which fragile tasks disproportionately lose accuracy as per-task reasoning decreases.

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.

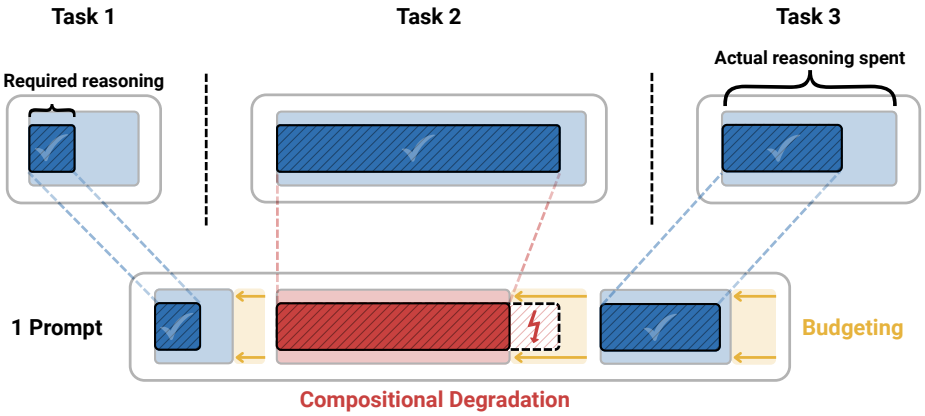


Figure 1: Composing multiple tasks in a single prompt forces the model to allocate a budget to each of them. Tasks that require larger thinking budget are more likely to be failed since their implicitly allocated thinking budget in the composition is more likely to be insufficient.

This mechanism also creates an opportunity for more efficient inference than solving every task separately. Non-fragile tasks can be grouped into larger batches, where the implicit thinking budget reduces per-task reasoning and reduces token usage without hurting accuracy. Fragile tasks, in contrast, are placed in smaller batches so that each receives more reasoning and remains accurate. This fragility-aware batching reduces token usage by 40% while preserving accuracy, and requires no modification to the model. Because it is entirely black-box, it also applies directly to proprietary models, reducing inference cost by 20% on GPQA-Diamond with Claude Opus 5 without degrading performance.

To summarize, the key contributions of our work are:

- We show that simply concatenating tasks that are already solved correctly in isolation causes substantial accuracy degradation across both frontier and open models. (Section 3).
- We identify an implicit per-task thinking budget that shrinks as a power law in the number of composed tasks, and show that this uniform reduction concentrates degradation on tasks with long isolated reasoning, a set that is stable within each model (Section 3.1).
- We leverage this structure for efficient inference by grouping tasks according to their measured fragility, reducing generated tokens by approximately 40% while preserving accuracy, on both state-of-the-art open source models and proprietary frontier models (Section 4).

2 Preliminaries

Our central goal is to investigate whether a capability demonstrated in isolation persists when several such tasks are presented together. To answer this, we use a controlled setting in which the constituent tasks are unchanged and only the number of tasks in the prompt varies. We then measure how each task's accuracy changes relative to its performance in isolation.

Compositions. We compose independent tasks drawn from the same dataset $\mathcal{D}$. A composition of size k is a tuple $C = (t_1, \ldots, t_k)$ with $t_i \in \mathcal{D}$, rendered into a single prompt by a fixed per-dataset template that asks the model to answer all k tasks in order. We split the model's single response into k answers and grade each for correctness (further details in A.2.6).

Accuracy drop. For a model m, the accuracy drop of task t_i at composition size k is $\Delta_i^m(k) = \mathrm{acc}_i^m(k) - \mathrm{acc}_i^m(1)$, where negative values indicate degradation. Unless stated otherwise, accuracy is estimated from 5 independent trials.

Datasets. We use eight datasets whose tasks vary widely in the reasoning they require. Two are constructed as controls: 1,000 indefinite integrals with randomized coefficients across diverse func-

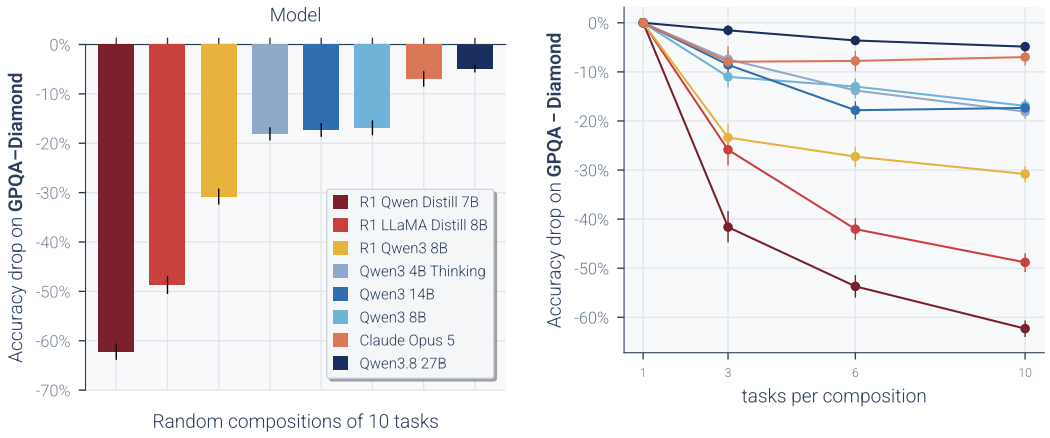


Figure 2: Accuracy degradation on confidently solved GPQA-Diamond tasks. **Left:** accuracy drop per model at $k = 10$. **Right:** accuracy drop as a function of composition size k. All tasks were solved in three out of three isolated attempts, yet accuracy drops once they are composed.

tion classes, verified symbolically, and OneWordQA, a set of factual questions with single-word or numeric answers that require minimal reasoning. The remaining six are established benchmarks: GPQA-Diamond (Rein et al., 2023), GSM8K (Cobbe et al., 2021), MATH-500 (Hendrycks et al., 2021; Lightman et al., 2023), MMLU-Redux (Gema et al., 2025), StrategyQA (Geva et al., 2021) and AIME (further details in A.2.6).

Models. We evaluate four Qwen3 models (4B Thinking, 8B, 14B, 27B) (Yang et al., 2025a; Qwen Team, 2026), post-trained with reinforcement learning, three DeepSeek-R1-distilled models (Qwen 7B, Llama 8B, Qwen3 8B) (Guo et al., 2025), obtained by supervised fine-tuning on R1 reasoning traces, and the frontier model Claude Opus 5 with medium reasoning effort. The first two groups let us compare two post-training recipes at similar scale. Open-weight models are sampled at temperature 0.6, and no generation in our experiments reaches the context limit (further details in A.2.2).

With this setup in place, we next turn to understanding the mechanism underlying compositional degradation.

3 Composing Tasks Causes Performance Degradation

Prior work has repeatedly reported that accuracy falls when several tasks share a single prompt. Studies that combine benchmark tasks (Pan et al., 2025; Cheng et al., 2023; Lin et al., 2024) or instructions (Vasileva, 2026; Harada et al., 2025) observe lower accuracy across entire datasets. However, these evaluations do not isolate whether a model fails because it cannot solve a task reliably or purely because the task is composed with others, which leaves the underlying mechanism unclear.

Building on these observations, we ask whether degradation can arise from composition alone. Answering this requires separating a model's ability to solve a task from its ability to solve the same task in composition. For that, we restrict each model M to a subset $\mathcal{D}_M \subseteq \mathcal{D}$ of *confidently solved* tasks: a task enters $\mathcal{D}_M$ only if M solves it in isolation in three out of three independent attempts. We then build compositions C of size $k \in \{3, 6, 10, 20\}$ exclusively from $\mathcal{D}_M$, sampling task categories uniformly within each dataset, and evaluate the accuracy of each task within them.

As Figure 2 shows, simply concatenating tasks that are solvable in isolation causes substantial accuracy degradation across all models on GPQA-Diamond (Rein et al., 2023). This observation also holds for all other datasets we tested, except OneWordQA, whose tasks require almost no reasoning (see A.1.1). Compositional degradation is therefore driven by effects inherent to composition and cannot be predicted from the solvability of tasks in isolation alone.

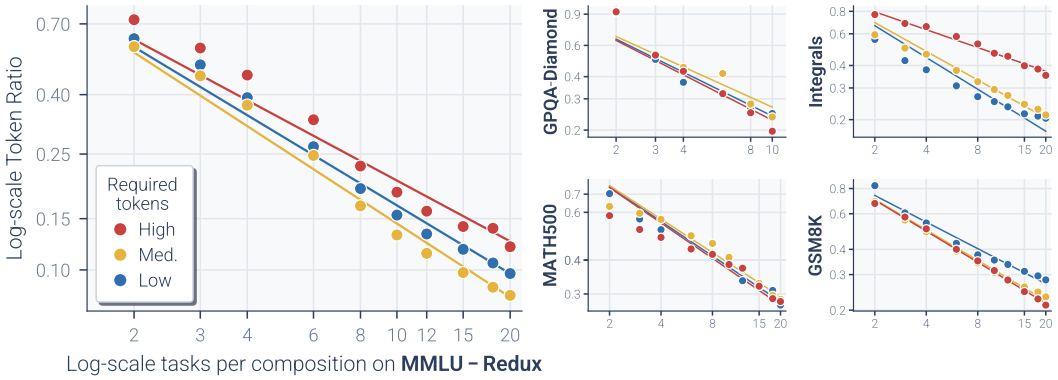


Figure 3: Token ratio $r_i(k) = T_i(k)/T_i(1)$ as a function of composition size k, where $T_i(k)$ is the number of thinking tokens spent on task t_i within a composition of size k and $T_i(1)$ the number spent in isolation. Per-task token counts in compositions are obtained by locating task tags in the reasoning trace. Tasks are binned by the quantile of their isolated trace length into Low ($\leq 20\%$), Medium (20–80%), and High ($> 80\%$). On log-log axes, the ratio decreases linearly with k, indicating that LLMs shrink the relative token usage of each task under composition by k^m, largely independent of $T_i(1)$. Results are from Qwen3 4B Thinking.

The extent of this degradation varies strongly across models. The R1-distilled models degrade the most, while the RL-trained Qwen3 models degrade moderately less. Frontier models are not immune either: despite their strong results on leading benchmarks, even Claude Opus 5 degrades on the widely used GPQA-Diamond benchmark under composition.

This has direct consequences for how frontier models can be used. Applications are increasingly moving towards long-horizon tasks, which are inherently compositions of many simpler subtasks. Solving such long tasks with high accuracy requires a fundamental understanding of what drives compositional degradation.

Contrary to the prevailing hypothesis that compositional degradation arises from a failure to generalize task-solving ability, as observed when tasks are dependent (Press et al., 2023; Dziri et al., 2023), we propose that it arises from a misallocation of the implicit thinking budget.

3.1 Degradation as a misallocation of implicit thinking budget

Reasoning models solve tasks by thinking step by step, a process shown to be crucial for task success (Wei et al., 2023). We therefore ask whether the accuracy degradation under composition can be explained by changes in thinking behaviour. A central property of this behaviour is the number of thinking tokens the model spends on a task, i.e., how much effort it invests in it. This effort can strongly affect whether a task is solved (Snell et al., 2024), and prior work has shown that the number of reasoning tokens per task generally decreases under composition (Srivastava et al., 2026; Deng et al., 2026). What remains unclear is how each constituent task is affected, that is, how its thinking length under composition compares to its thinking length in isolation.

Thinking traces shrink under composition. We close this gap by measuring the thinking tokens spent on each task within a composition of k tasks, denoted $T_i(k)$. To this end, we prefix each task with a unique tag (Q01, Q02, ...) in the prompt that does not appear in the task description itself. These tags let us attribute each segment of the reasoning trace to the task it addresses (details in A.2.1).

We quantify the reduction for each task by the token ratio $r_i(k) = T_i(k)/T_i(1)$ between its thinking tokens under composition and in isolation. Figure 3 shows how this token ratio evolves with composition size. The thinking tokens of almost all tasks decrease monotonically as compositions grow.

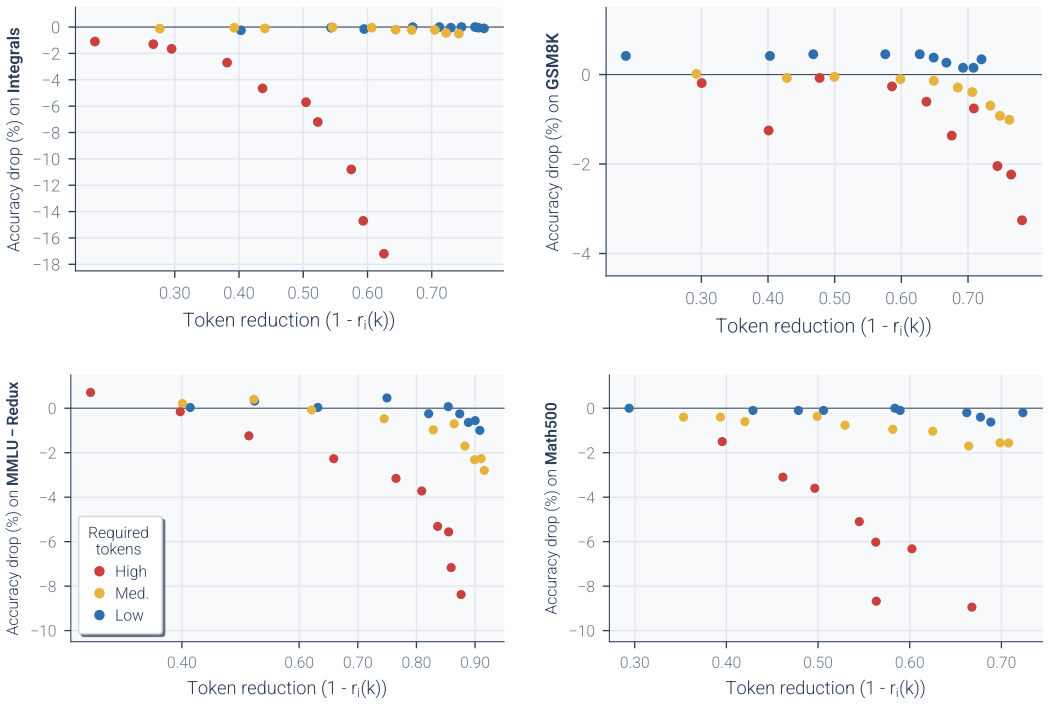


Figure 4: Accuracy drop as a function of the relative thinking token reduction for Qwen3 4B Thinking. Tasks are binned by the quantile of their isolated trace length into Low ($\leq$ 20%), Medium (20–80%), and High ($>$ 80%). As the reduction grows, accuracy drops, and tasks with high reasoning requirements degrade substantially more than low-effort tasks. **Top left:** Indefinite Integrals. **Top right:** GSM8K. **Bottom left:** MMLU-Redux. **Bottom right:** Math-500.

Long-reasoning tasks degrade more. This reduction has important implications, since prior work has established that the number of thinking tokens spent on a task can influence its accuracy. In particular, interrupting the thinking process externally at earlier stages causes a larger degradation (Muennighoff et al., 2025). Composing tasks has an analogous effect on thinking length, as it implicitly constrains the per-task thinking tokens but without any external enforcement. This naturally raises the question of whether the implicit token reduction under composition is related to the accuracy degradation.

Figure 4 shows how the reduction affects tasks with different isolated thinking lengths. Tasks with longer isolated traces degrade substantially more when thinking tokens are reduced, whereas tasks with shorter traces are largely unaffected. We refer to tasks that break under composition in this way as *fragile*. As a result, degradation under reduced thinking is highly heterogeneous across tasks.

Thinking length follows a power law. One possible source of the uneven degradation in Figure 4 could be that long-reasoning tasks lose a larger share of their thinking tokens. The token ratio approximately follows a *power law in* k,

$$\log r_i(k) \approx m \log k \quad \Longleftrightarrow \quad r_i(k) = \frac{T_i(k)}{T_i(1)} \approx k^m. \tag{1}$$

Since $r_i(1) = 1$ by definition, we fit a line through the origin on log-log axes for each model and dataset. The fitted slopes range from $m = -0.79$ to $m = -0.04$, with a median R^2 of 0.90.

Under composition, the reasoning trace of each task thus shrinks by approximately the same factor k^m, without systematic dependence on its isolated length $T_i(1)$, so long-reasoning tasks do not systematically lose a larger share of their reasoning than short ones. The model preserves the relative allocation across tasks, assigning more reasoning to tasks that required more in isolation, but applies roughly the same proportional reduction to all of them.

Since every task loses roughly the same fraction of its reasoning, the uneven degradation must come from how much of that reasoning each task can afford to lose. Short isolated traces contain redundancy in reasoning beyond what the task requires, whereas long traces lie closer to the minimum reasoning needed to solve the task reliably. Hence, the same relative reduction pushes long-reasoning tasks below their minimal requirement while leaving short ones above it, concentrating degradation on tasks with long isolated traces. Since k^m decreases with k, the implicit thinking budget shrinks as more tasks are composed, and degradation grows accordingly.

This reveals a systematic failure of implicit thinking budget estimation under composition. The implicit thinking budget that models estimate *does not generalize well to composed tasks*.

3.2 Is the set of fragile tasks stable?

Consequently, it remains to establish whether the implicit thinking budget estimation fails consistently on the same tasks across models and composition sizes. This tells us how the proposed misallocation changes as the conditions, i.e., model and composition size, vary.

Since a failure in implicit thinking budget estimation manifests as accuracy degradation, we can test this consistency directly on the accuracy drops. For two conditions A and B, we obtain the accuracy drops across tasks, Δ_A and Δ_B, and measure their Pearson correlation, applying Spearman disattenuation to reduce the impact of noisy accuracy estimates. When aggregating over multiple condition pairs, we compute correlations within strata of isolated accuracy, pool them via a Fisher-z transformation, and report the back-transformed mean (details in A.2.3).

At a fixed composition size, the mean correlation across models is only 0.27 ± 0.04. Different models thus largely disagree on which tasks degrade, even when evaluated under identical conditions. Within a model, by contrast, the correlation across composition sizes reaches 0.88 ± 0.02. Therefore, when the same model faces compositions of different sizes and structures, the same set of fragile tasks tends to break.

Fragility is therefore a stable property of a task for a given model. Measuring a task's accuracy drop at some composition sizes lets us anticipate how it will degrade in new compositions presented to the same model, which makes fragility predictable.

4 Exploiting budget misallocation for efficient and accurate inference?

Our results so far indicate that LLMs fail on compositions not because they cannot transfer their problem-solving skills to the constituent tasks, but because their estimation of the implicit thinking budget does not generalize to composition. We therefore test whether we can control this estimation so that each task receives a budget sufficient to solve it reliably.

Our observations suggest that the composition size k offers a practical way to influence implicit budgeting, since the token reduction ratio follows a power law in k. Assigning robust tasks to larger compositions and fragile tasks to smaller ones should therefore minimize token usage while keeping the accuracy drop small.

4.1 Fragility-aware batching

To test this, we propose an intervention that estimates the fragility of each task from empirical accuracy measurements and composes tasks accordingly.

Formally, we seek to assign each task a composition size k_i that maximizes the expected token savings while bounding the expected accuracy degradation by B. We express this trade-off as a Lagrangian relaxation,

$$k_i(\lambda) = \arg\max_{k \in \mathcal{K}} \Big[\mathbb{E}[\Delta_i(k)] + \lambda\big(1 - \mathrm{cost}_k\big) \Big], \tag{2}$$

where $\mathcal{K}$ is the set of available composition sizes, $\mathbb{E}[\Delta_i(k)]$ is the expected accuracy drop of task t_i at composition size k, and $\mathrm{cost}_k \in (0, 1]$ is the model-specific token cost of solving a task within a

Table 1: Task accuracy and output tokens per task under different batching policies. For each policy, the left column reports accuracy and the right column tokens, with the change to solving in isolation ($k = 1$) in parentheses, given as an absolute difference in percentage points for accuracy and as a relative change in % for tokens. The Average row gives the mean change over the seven models shown. Standard fixed-size random batching causes substantial accuracy degradation, which fragility-aware batching largely mitigates. With a model-specific linear probe, part of these token savings carries over to unseen tasks while still avoiding compositional degradation.

	Isolated, k=1		Random batching, k=6		Fragility-aware batching		Held-out (linear probe)	
	Acc.	Tok.	Acc.	Tok.	Acc.	Tok.	Acc.	Tok.
GPQA-Diamond								
Qwen3 4B T.	66.7%	7663	56.4% (−10.3)	2470 (−67.8%)	66.8% (+0.2)	5200 (−32.1%)	67.3% (+0.7)	7534 (−1.7%)
Qwen3 8B	58.7%	5973	53.4% (−5.3)	1950 (−67.3%)	60.0% (+1.3)	3520 (−41.1%)	56.9% (−1.8)	5106 (−14.5%)
Qwen3 14B	61.5%	4597	51.8% (−9.7)	1500 (−67.4%)	57.9% (−3.7)	2111 (−54.1%)	61.3% (−0.2)	3665 (−20.3%)
Qwen3.8 27B	83.2%	4303	79.2% (−4.0)	2740 (−36.3%)	84.0% (+0.8)	4102 (−4.7%)	85.3% (+2.1)	4324 (+0.5%)
R1 Qwen 7B	49.1%	6686	23.6% (−25.5)	1305 (−80.5%)	49.6% (+0.5)	4037 (−39.6%)	46.7% (−2.4)	5953 (−11.0%)
R1 Llama 8B	47.7%	7424	32.1% (−15.6)	1771 (−76.2%)	49.6% (+1.9)	4526 (−39.0%)	48.6% (+0.9)	5816 (−21.7%)
R1 Qwen3 8B	60.8%	9084	48.1% (−12.6)	2398 (−73.6%)	61.9% (+1.2)	6595 (−27.4%)	59.8% (−1.0)	8877 (−2.3%)
Average	61.1%	6533	−11.9	−67.0%	+0.3	−34.0%	−0.2	−10.1%
MMLU-Redux								
Qwen3 4B T.	84.7%	1944	82.9% (−1.8)	527 (−72.9%)	85.3% (+0.6)	527 (−72.9%)	84.9% (+0.2)	1663 (−14.5%)
Qwen3 8B	87.2%	921	84.0% (−3.3)	306 (−66.7%)	86.1% (−1.1)	285 (−69.0%)	86.9% (−0.3)	922 (+0.2%)
Qwen3 14B	89.0%	718	85.8% (−3.3)	212 (−70.5%)	88.2% (−0.9)	319 (−55.6%)	88.3% (−0.7)	678 (−5.5%)
Qwen3.8 27B	92.4%	488	91.9% (−0.5)	262 (−46.4%)	91.8% (−0.6)	196 (−59.8%)	92.1% (−0.3)	352 (−27.9%)
R1 Qwen 7B	70.1%	1078	60.6% (−9.5)	397 (−63.1%)	69.7% (−0.3)	601 (−44.3%)	70.4% (+0.3)	1127 (+4.5%)
R1 Llama 8B	75.5%	1458	58.6% (−16.9)	431 (−70.4%)	75.9% (+0.3)	896 (−38.5%)	75.3% (−0.2)	1430 (−1.9%)
R1 Qwen3 8B	81.6%	1710	64.5% (−17.1)	279 (−83.7%)	82.1% (+0.5)	1273 (−25.6%)	82.2% (+0.6)	1714 (+0.2%)
Average	82.9%	1188	−7.5	−67.7%	−0.2	−52.2%	−0.1	−6.4%
MATH-500								
Qwen3 4B T.	97.6%	4064	96.4% (−1.1)	2148 (−47.1%)	97.7% (+0.1)	3716 (−8.6%)	96.8% (−0.8)	3605 (−11.3%)
Qwen3 8B	95.9%	3449	95.6% (−0.3)	1751 (−49.2%)	95.1% (−0.8)	2173 (−37.0%)	95.9% (+0.1)	2775 (−19.6%)
Qwen3 14B	96.9%	2604	95.9% (−1.0)	1375 (−47.2%)	96.6% (−0.2)	2454 (−5.8%)	96.6% (−0.2)	2444 (−6.2%)
Qwen3.8 27B	98.1%	1014	96.7% (−1.5)	748 (−26.2%)	97.8% (−0.4)	943 (−7.1%)	97.2% (−0.9)	994 (−2.0%)
R1 Qwen 7B	93.0%	2383	69.8% (−23.2)	1658 (−30.4%)	93.2% (+0.2)	2205 (−7.5%)	93.4% (+0.4)	2381 (−0.1%)
R1 Llama 8B	88.7%	3019	79.4% (−9.3)	1569 (−48.0%)	88.5% (−0.2)	2948 (−2.3%)	88.8% (+0.2)	3333 (+10.4%)
R1 Qwen3 8B	96.8%	6767	93.0% (−3.8)	3705 (−45.2%)	96.6% (−0.2)	6608 (−2.4%)	96.7% (−0.2)	6657 (−1.6%)
Average	95.3%	3329	−5.7	−41.9%	−0.2	−10.1%	−0.2	−4.3%

composition of size k, relative to solving it in isolation. The parameter $\lambda \geq 0$ acts as an exchange rate between saved tokens and accuracy.

We then choose λ such that the mean expected accuracy drop over the dataset, $\bar{\Delta}(\lambda) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{E}\big[\Delta_i(k_i(\lambda))\big]$, conservatively stays within the budget B,

$$\lambda^* = \max\Big\{\lambda \geq 0 \;:\; \bar{\Delta}(\lambda) - \mathrm{std}\big(\bar{\Delta}(\lambda)\big) \geq -B\Big\}. \tag{3}$$

We find λ^* by a grid search on held-out splits, using repeated 5-fold cross-fitting (Chernozhukov et al., 2024) to reduce the effect of noise in the empirical accuracy estimates. Given the assignments $k_i(\lambda^*)$, we greedily group tasks into as few compositions as possible such that every task t_i lands in a composition C with $|C| \leq k_i(\lambda^*)$ (details in A.2.4). Throughout our experiments, we use the composition sizes $\mathcal{K} = \{3, 6, 10, 20\}$ and an accuracy drop budget of $B = 0.5\%$.

Table 1 compares the accuracy degradation and token savings of different batching strategies relative to solving tasks in isolation. The common practice of batching tasks randomly into fixed-size compositions (Cheng et al., 2023), here of six tasks, saves the most tokens on average but degrades accuracy strongly, particularly on GPQA-Diamond, whose tasks require many reasoning steps. Fragility-aware batching almost entirely avoids this degradation while still achieving substantial token savings. Averaged over all datasets and open-source models, our intervention achieves token savings of 41.7% while keeping accuracy constant ($+0.5pp$).

These results show that the thinking traces of certain tasks contain many redundant tokens. By controlling the composition size accordingly, this redundancy can be exploited to save tokens without incurring compositional degradation.

4.2 PREDICTING FRAGILITY ON HELD-OUT TASKS

Our results show that the set of fragile tasks is stable for a given model, which hints at the existence of steady properties that define fragility. If such properties exist, fragility should be predictable for tasks the model has never seen under composition. Hence, we extend the intervention to unseen tasks by predicting their fragility with a simple linear probe.

We embed the task description of each task with Qwen3 Embedding 8B (Zhang et al., 2025) and train a model-specific linear probe on these embeddings. To generate labels, we evaluate each task in the training split with 10 trials at every composition size $k \in \{1, 3, 6, 10, 20\}$, placing it in a different composition each time. We cast prediction as ordinal regression over binary outcomes, where for each composition size k the probe outputs the probability that the task performs at least as well in composition as in isolation. To obtain predictions on unseen tasks, we use 5-fold cross-fitting, so that the held-out folds together cover the entire dataset. Finally, we feed the predicted accuracy drops into the same batching algorithm as in Section 4.1.

Table 1 reports the results of the linear probe. Fragility-aware batching on a held-out set guided by its predictions largely avoids compositional degradation while still yielding slightly smaller token savings. Across all datasets and models, we observed an average token saving of 29.7% while the accuracy stayed constant ($+0.5pp$). Fragility is therefore partly predictable from task properties encoded in the embeddings. This suggests that scaling the predictor and strengthening its generalization could make it possible to save tokens on unseen tasks without degrading accuracy, simply by batching them appropriately.

4.3 CAN WE FIX DEGRADATION IN FRONTIER MODELS?

We further test whether fragility-aware batching carries over to flagship frontier LLMs. Since the intervention requires neither fine-tuning nor access to internal states and works purely by restructuring compositions, we can apply it directly to Claude Opus 5 on GPQA-Diamond. In isolation, Claude Opus 5 reaches an accuracy of 89.4%. Composing tasks with $k = 3$ lowers accuracy by -1.2 pp across the full dataset while reducing output tokens from 708 to 658 per task, a 7% decrease. This drop differs from Figure 2, where compositions are sampled with balanced task categories, but still shows a small and noticeable degradation, concentrated on tasks from the Biology category. Since the loss is confined to a subset of tasks, it is exactly the kind of degradation that fragility-aware batching is designed to avoid. For that, we estimate the fragility of each task from empirical observations and compose the tasks accordingly. This recovers most of the lost accuracy, reaching 89.2% (-0.2 pp), while reducing output tokens to 564 per task, a 20% decrease. Our main findings on small and medium open models thus transfer to large frontier LLMs, motivating further practical methods to save compute without sacrificing performance.

5 RELATED WORK

Previous work has examined both how task composition affects accuracy and reasoning length and how batching can improve efficiency or reliability. Several studies ask whether models can solve tasks as reliably together as they do separately (Pan et al., 2025; Wang et al., 2025; Chen et al., 2026). Related studies examine whether failures to meet multiple response requirements can be explained by independent errors (Harada et al., 2025; Vasileva, 2026). These studies document accuracy losses under composition, although their causes are not yet fully understood. At the same time, batching methods have shown promise for improving efficiency and reliability (Xu et al., 2026b; Ji et al., 2025), with accuracy gains reported in some settings (Srivastava et al., 2026; Yang et al., 2026b; Lin et al., 2024). Batching has also been observed to reduce the number of reasoning tokens used per task (Srivastava et al., 2026; Deng et al., 2026). The effects of added context also vary across questions. For factual questions, models tend to resist misleading information more successfully when they can also answer related questions correctly, suggesting that accuracy in isolation does not fully capture robustness (Xu et al., 2026a). The number of steps needed to solve a task also matters. Tasks requiring more steps show larger accuracy drops when irrelevant information is added (Yang et al., 2025b). More broadly, researchers distinguish task difficulty from sensitivity to prompt changes (Romanou et al., 2026) and examine how that sensitivity varies with model size, training and architecture (Qin et al., 2026). Overall accuracy can remain largely unchanged even

when irrelevant context changes which questions a model answers correctly (Zhang et al., 2026). Estimates of this sensitivity also depend on how answers are graded. Rigid matching rules can treat differently worded correct answers as errors, exaggerating the apparent sensitivity (Hua et al., 2025). Alongside these studies, work on reasoning allocation has shown that adding other tasks or irrelevant text can shorten reasoning traces and reduce self-checking (Rodionov et al., 2026). Researchers have also trained models to solve groups of tasks more efficiently under a shared token limit (Yang et al., 2026a). However, studies of budget allocation find that models often devote more reasoning to earlier questions than to those worth more points (Fan et al., 2026). In a single-task setting, model confidence has also been used to guide a search for shorter reasoning paths (Huang et al., 2026).

These findings can be difficult to reconcile within a common explanation, given the differences in experimental settings, interventions and datasets. Our work offers a unifying perspective on several of these observations. Budget misallocation is consistent with the greater sensitivity of deeper reasoning tasks to irrelevant context (Yang et al., 2025b), self-compression (Deng et al., 2026), composition errors that cannot be explained by independent error rates alone (Harada et al., 2025), and the token savings achieved through batching (Srivastava et al., 2026). We identify reasoning trace length in isolation as a key variable that helps connect these findings and informs simple, effective interventions. Using this information, we are able to show how to mitigate degradation under composition, even in frontier models.

6 Discussion and Conclusions

In this work, we show that language models fail on tasks they solve reliably in isolation once those tasks are combined in a single prompt. The resulting accuracy drop is larger than expected from isolated error rates alone. As more tasks are combined, models use fewer reasoning tokens per task, following an approximate power-law decay with composition size, and the largest failures occur on tasks that require longer reasoning in isolation. This reveals an implicit per-task budget under composition. We further show that this structure can be exploited for efficiency by batching tasks fragility-aware. This strategy reduces token usage by about 40% while preserving accuracy and requires no changes to the model. Because it treats the model as a black box, it applies directly to proprietary models as well, lowering inference cost on GPQA-Diamond by 20% for Claude Opus 5 without impacting performance.

We deliberately compose independent tasks of the same type, isolating the budget effect in its simplest form. An important question is how this effect extends to richer contexts. Prior work finds that reasoning traces also shorten when the same task is embedded in irrelevant context, conversation history, or a larger task (Rodionov et al., 2026), suggesting that implicit thinking budgeting may extend beyond same-type composition. This is particularly relevant to agentic workflows, where tool outputs, interaction histories, and intermediate results continually accumulate. Dependencies between subtasks add another dimension. An early error may propagate forward, while shared intermediate reasoning may also make later steps easier. Understanding how these effects interact remains an important direction for future work.

More broadly, our results suggest that the core difficulty under composition lies not in whether models know how to solve the constituent tasks, but in how much reasoning they allocate to each one. As composition size grows, per-task reasoning decreases in a predictable, model- and dataset-specific way, with the largest accuracy losses on fragile tasks. This points to budget allocation itself as an important capability that must generalize beyond single-task inference.

Limitations. While we show that composition systematically reduces per-task reasoning and disproportionately harms fragile tasks, we do not yet know what drives this reduction. Our study is limited to independent, same-dataset compositions, and detailed reasoning-length analysis is restricted to open-weight models. Extending these results to mixed-domain, dependent, and proprietary-model settings remains important future work.

AI USE STATEMENT

We used generative AI tools to support parts of this work. Claude Code (Claude Opus 5) assisted in implementing our experimental setups as code, which the authors then manually tested and audited. The conception of experiments and the interpretation of results were carried out by the authors. The OneWordQA dataset was generated with Claude Opus 5, and because its questions and answers are simple, the authors verified their correctness manually. AI tools were also used to polish the language of the manuscript. All AI-assisted content was reviewed by the authors, who take full responsibility for the final work.

REFERENCES

Jingxuan Chen, Mohammad Taher Pilehvar, and Jose Camacho-Collados. Understanding llm performance degradation in multi-instance processing: The roles of instance count and context length, 2026. URL https://arxiv.org/abs/2603.22608.

Zhoujun Cheng, Jungo Kasai, and Tao Yu. Batch prompting: Efficient inference with large language model apis, 2023. URL https://arxiv.org/abs/2301.08721.

Victor Chernozhukov, Denis Chetverikov, Mert Demirer, Esther Duflo, Christian Hansen, Whitney Newey, and James Robins. Double/debiased machine learning for treatment and causal parameters, 2024. URL https://arxiv.org/abs/1608.00060.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

Jie Deng, Shining Liang, Jun Li, Hongzhi Li, and Yutao Xie. ConPress: Learning Efficient Reasoning from Multi-Question Contextual Pressure. In *International Conference on Machine Learning*, 2026. URL https://arxiv.org/abs/2602.01472.

Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jiang, Bill Yuchen Lin, Peter West, Chandra Bhagavatula, Ronan Le Bras, Jena D. Hwang, Soumya Sanyal, Sean Welleck, Xiang Ren, Allyson Ettinger, Zaid Harchaoui, and Yejin Choi. Faith and fate: Limits of transformers on compositionality, 2023. URL https://arxiv.org/abs/2305.18654.

Chenrui Fan, Yize Cheng, Ming Li, Yongyuan Liang, Tianyi Zhou, and Soheil Feizi. Thinking Hard, Not Smart: Reasoning Models Fail to Ration Test-Time Compute Across Questions, 2026. URL https://arxiv.org/abs/2608.07968.

Aryo Pradipta Gema, Joshua Ong Jun Leang, Giwon Hong, Alessio Devoto, Alberto Carlo Maria Mancino, Rohit Saxena, Xuanli He, Yu Zhao, Xiaotang Du, Mohammad Reza Ghasemi Madani, Claire Barale, Robert McHardy, Joshua Harris, Jean Kaddour, Emile van Krieken, and Pasquale Minervini. Are we done with mmlu?, 2025. URL https://arxiv.org/abs/2406.04127.

Mor Geva, Daniel Khashabi, Elad Segal, Tushar Khot, Dan Roth, and Jonathan Berant. Did aristotle use a laptop? a question answering benchmark with implicit reasoning strategies, 2021. URL https://arxiv.org/abs/2101.02235.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Honghui Ding, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jingchang Chen, Jingyang Yuan, Jinhao Tu, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaichao You, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingxu Zhou, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan

Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638, 2025. ISSN 1476-4687. doi: 10.1038/s41586-025-09422-z. URL http://dx.doi.org/10.1038/s41586-025-09422-z.

Keno Harada, Yudai Yamazaki, Masachika Taniguchi, Edison Marrese-Taylor, Takeshi Kojima, Yusuke Iwasawa, and Yutaka Matsuo. When Instructions Multiply: Measuring and Estimating LLM Capabilities of Multiple Instructions Following. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 16506–16526, 2025. doi: 10.18653/v1/2025.findings-emnlp.896. URL https://aclanthology.org/2025.findings-emnlp.896/.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset, 2021. URL https://arxiv.org/abs/2103.03874.

Andong Hua, Kenan Tang, Chenhe Gu, Jindong Gu, Eric Wong, and Yao Qin. Flaw or Artifact? Rethinking Prompt Sensitivity in Evaluating LLMs. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 19889–19899, 2025. doi: 10.18653/v1/2025.emnlp-main.1006. URL https://aclanthology.org/2025.emnlp-main.1006/.

Zixuan Huang, Xin Xia, Yuxi Ren, Jianbin Zheng, Xuanda Wang, Zhixia Zhang, Hongyan Xie, Songshi Liang, Zehao Chen, Xuefeng Xiao, Fuzhen Zhuang, Jianxin Li, Deqing Wang, and Yikun Ban. Does Your Reasoning Model Implicitly Know When to Stop Thinking? In *International Conference on Machine Learning*, 2026. URL https://arxiv.org/abs/2602.08354.

Zhaoxuan Ji, Xinlu Wang, Zhaojing Luo, Zhongle Xie, and Meihui Zhang. Optimized Batch Prompting for Cost-effective LLMs. *Proceedings of the VLDB Endowment*, 18(7):2172–2184, 2025. doi: 10.14778/3734839.3734853. URL https://www.vldb.org/pvldb/vol18/p2172-zhang.pdf.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step, 2023. URL https://arxiv.org/abs/2305.20050.

Jianzhe Lin, Maurice Diesendruck, Liang Du, and Robin Abraham. Batchprompt: Accomplish more with less, 2024. URL https://arxiv.org/abs/2309.00384.

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, 2025. URL https://arxiv.org/abs/2501.19393.

Zhuoshi Pan, Qizhi Pei, Yu Li, Qiyao Sun, Zinan Tang, H. Vicky Zhao, Conghui He, and Lijun Wu. Rest: Stress testing large reasoning models by asking multiple problems at once, 2025. URL https://arxiv.org/abs/2507.10541.

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A. Smith, and Mike Lewis. Measuring and Narrowing the Compositionality Gap in Language Models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 5687–5711, 2023. doi: 10.18653/v1/2023.findings-emnlp.378. URL https://aclanthology.org/2023.findings-emnlp.378/.

Ruiyang Qin, Qingzhuo Wang, Tian Wang, Zhihua Wei, and Wen Shen. Evaluating and Explaining Prompt Sensitivity of LLMs Using Interactions. In *International Conference on Machine Learning*, 2026. URL https://arxiv.org/abs/2608.18539.

Qwen Team. Qwen3.8-Max: A new bar for coding and cowork, August 2026. URL https://qwen.ai/blog?id=qwen3.8.

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 qa benchmark, 2023. URL https://arxiv.org/abs/2311.12022.

Gleb Rodionov, Roman Garipov, and George Yakushev. Reasoning Shift: How Context Silently Shortens LLM Reasoning, 2026. URL https://arxiv.org/abs/2604.01161.

Angelika Romanou, Mark Ibrahim, Candace Ross, Chantal Shaib, Kerem Oktar, Samuel J. Bell, Anaelia Ovalle, Jesse Dodge, Antoine Bosselut, Koustuv Sinha, and Adina Williams. Brittlebench: Quantifying LLM Robustness via Prompt Sensitivity, 2026. URL https://arxiv.org/abs/2603.13285.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters, 2024. URL https://arxiv.org/abs/2408.03314.

Saurabh Srivastava, Janit Bidhan, Hao Yan, Abhishek Dey, Tanu Kansal, Paras Kath, Sina Mansouri, Mohit Marvania, Vamsi Shankar Simhadri, and Gaurav Singh. Reasoning Under Constraint: How Batch Prompting Suppresses Overthinking in Reasoning Models. ICLR Workshop on Logical Reasoning of Large Language Models, 2026. URL https://iclr.cc/virtual/2026/poster/10017459. Non-archival workshop presentation; preprint first posted in 2025.

Mariya I. Vasileva. Large language models can follow instructions, but not many at once: Phase transitions in compositional constraint satisfaction, 2026. URL https://arxiv.org/abs/2608.12426.

Zhengxiang Wang, Jordan Kodner, and Owen Rambow. Evaluating LLMs with Multiple Problems at once. In *Proceedings of the Fourth Workshop on Generation, Evaluation and Metrics (GEM2)*, pp. 178–199, 2025. URL https://aclanthology.org/2025.gem-1.14/.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023. URL https://arxiv.org/abs/2201.11903.

Haoming Xu, Ningyuan Zhao, Yunzhi Yao, Weihong Xu, Hongru Wang, Xinle Deng, Shumin Deng, Jeff Z. Pan, Huajun Chen, and Ningyu Zhang. Illusions of Confidence? Diagnosing LLM Truthfulness via Neighborhood Consistency. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 4432–4457. Association for Computational Linguistics, 2026a. doi: 10.18653/v1/2026.acl-long.203. URL https://aclanthology.org/2026.acl-long.203/.

Haotian Xu, Kangfei Zhao, and Jiadong Xie. Towards Cost-effective LLMs Routing with Batch Prompting, 2026b. URL https://arxiv.org/abs/2605.28268.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025a. URL https://arxiv.org/abs/2505.09388.

Bangji Yang, Hongbo Ma, Jiajun Fan, and Ge Liu. Batched Contextual Reinforcement: A Task-Scaling Law for Efficient Reasoning, 2026a. URL https://arxiv.org/abs/2604.02322v1. arXiv version 1.

Minglai Yang, Ethan Huang, Liang Zhang, Mihai Surdeanu, William Yang Wang, and Liangming Pan. How Is LLM Reasoning Distracted by Irrelevant Context? An Analysis Using a Controlled Benchmark. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 13329–13347, 2025b. doi: 10.18653/v1/2025.emnlp-main.674. URL https://aclanthology.org/2025.emnlp-main.674/.

Xuan Yang, Furong Jia, Roy Xie, Xiong Xi, Hengwei Bian, Jian Li, and Monica Agrawal. Batch-of-Thought: Cross-Instance Learning for Enhanced LLM Reasoning, 2026b. URL https://arxiv.org/abs/2601.02950.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. Qwen3 embedding: Advancing text embedding and reranking through foundation models, 2025. URL https://arxiv.org/abs/2506.05176.

Yanzhe Zhang, Sanmi Koyejo, and Diyi Yang. The Illusion of Robustness: Aggregate Accuracy Hides Prediction Flips under Task-Irrelevant Context, 2026. URL https://arxiv.org/abs/2607.12963.

A Appendix

A.1 Further results

A.1.1 Compositional Degradation is observable over a wide range of datasets and models

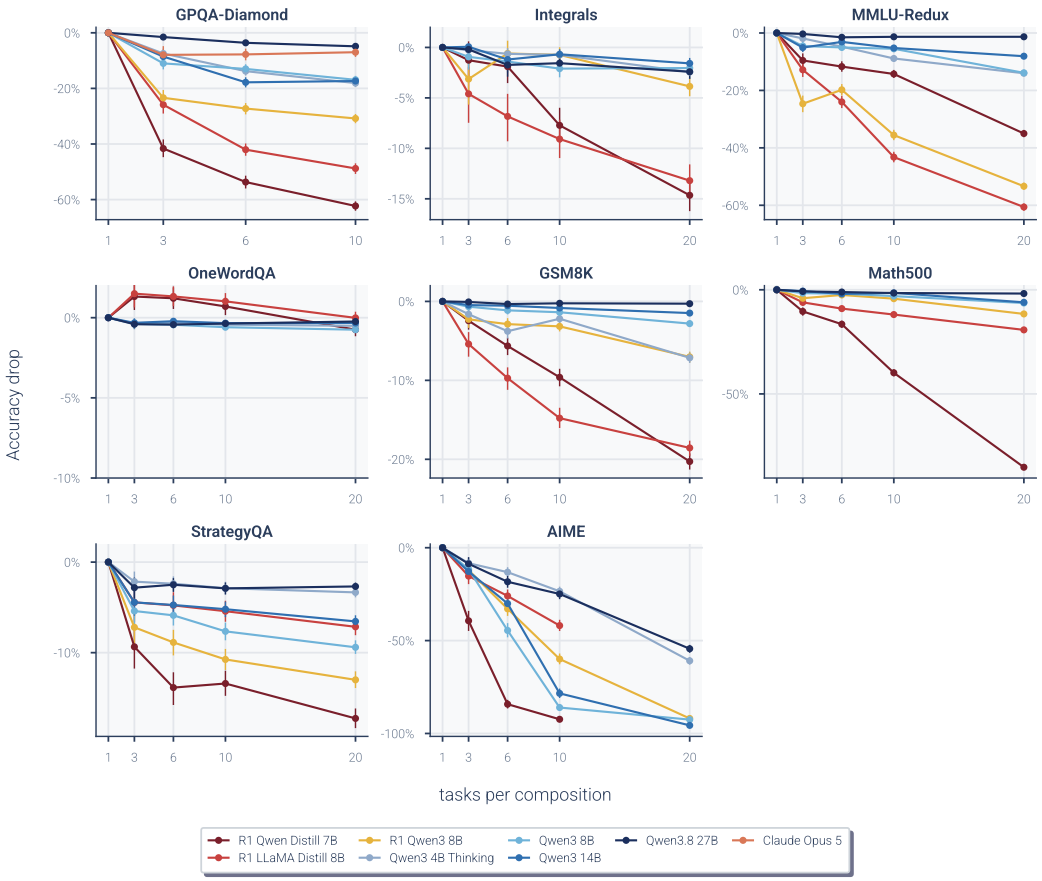


Figure 5: Accuracy drop as a function of composition size k for all datasets and models. All tasks were solved in three out of three isolated attempts, yet accuracy drops once they are composed, on every dataset except OneWordQA, whose tasks require almost no reasoning. The R1-distilled models degrade the most, and the extent of degradation varies strongly between datasets. If a data point is missing, this means that the confidently solved set did not contain enough samples to reach the desired composition size at least once.

A.1.2 COMPOSITION SIZE INFLUENCES TOKEN RATIO

Figures 6–9 show the token ratio $r_i(k) = T_i(k)/T_i(1)$ as a function of composition size k for four models. Tasks are binned by the quantile of their isolated trace length into Low ($\leq 20\%$), Medium (20–80%), and High ($> 80\%$). On log-log axes, the ratio decreases approximately linearly with k for nearly all models and datasets, indicating that LLMs shrink the relative token usage of each task under composition by k^m. This reduction shows no systematic dependence on $T_i(1)$, since the ordering of the length bins varies across models and datasets and no bin is consistently reduced more than the others. We fix each power-law fit to pass through $(1, 1)$, since a composition of a single task has a token ratio of 1 by construction. We restrict this analysis to a subset of four models, chosen to span different training recipes and model sizes, as running it on all models would be prohibitively expensive.

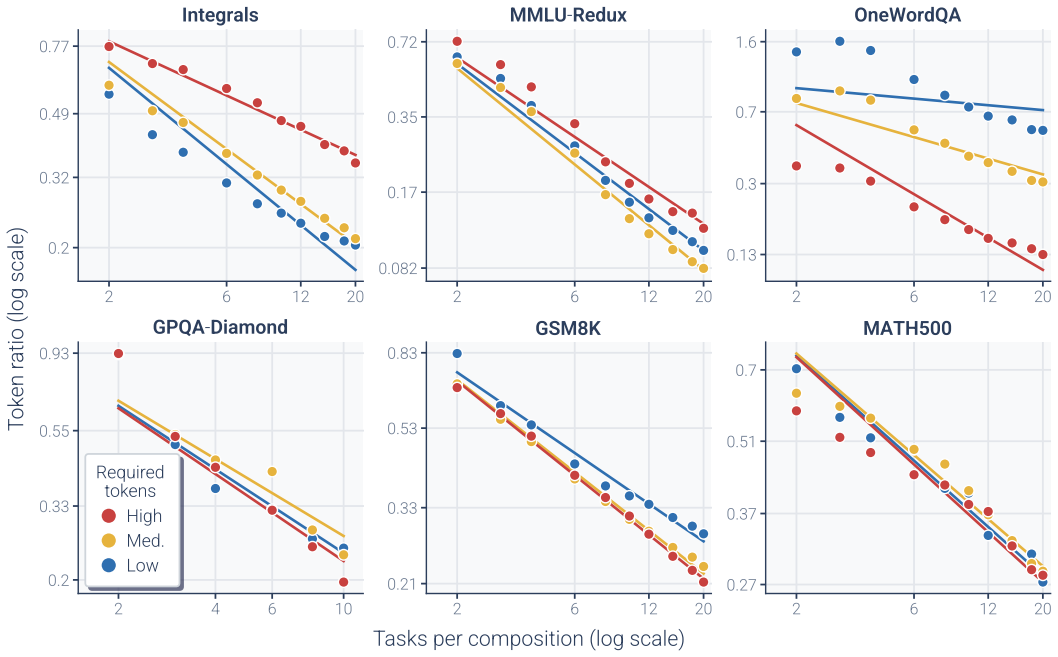


Figure 6: Token ratio on Qwen3 4B Thinking. Power-law fit: $R^2 = 0.91$.

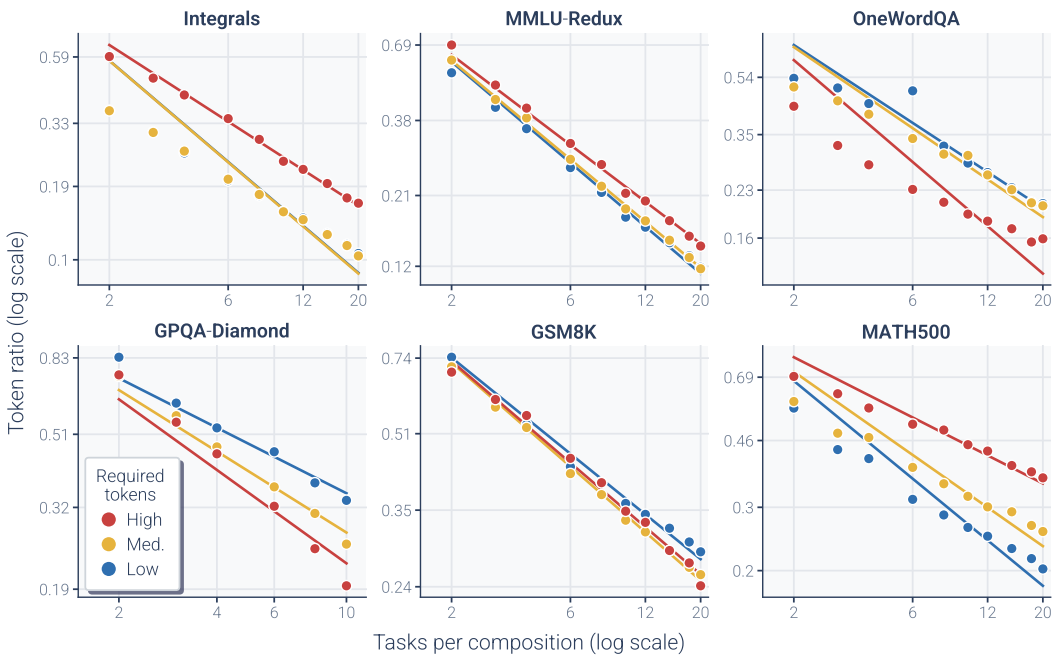


Figure 7: Token ratio on Qwen3 8B. Power-law fit: $R^2 = 0.94$.

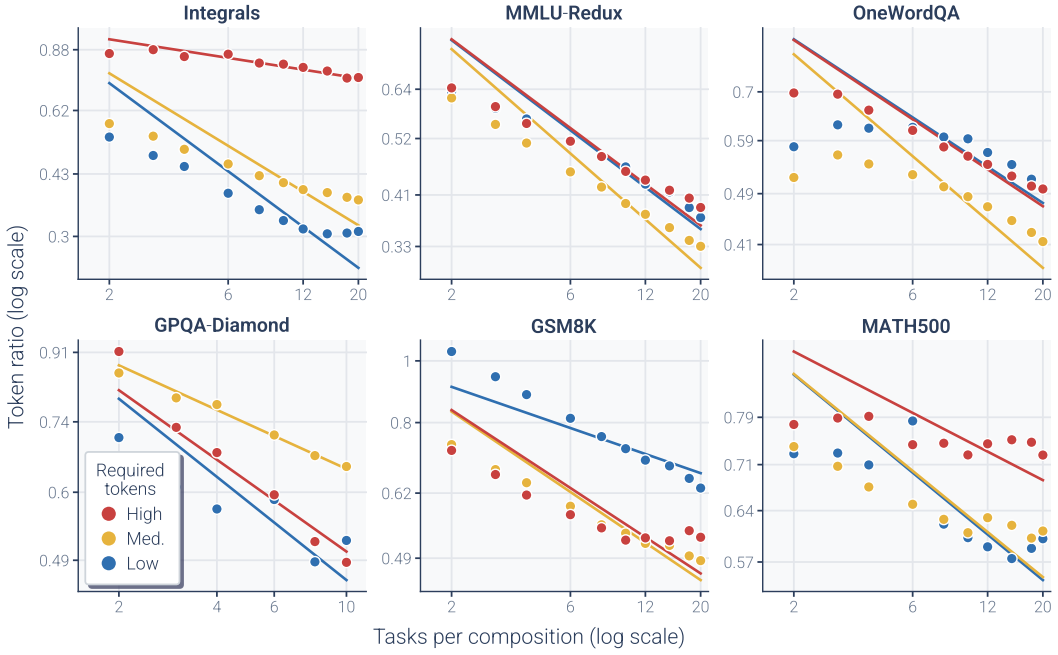


Figure 8: Token ratio on Qwen3.8 27B. Power-law fit: $R^2 = 0.72$.

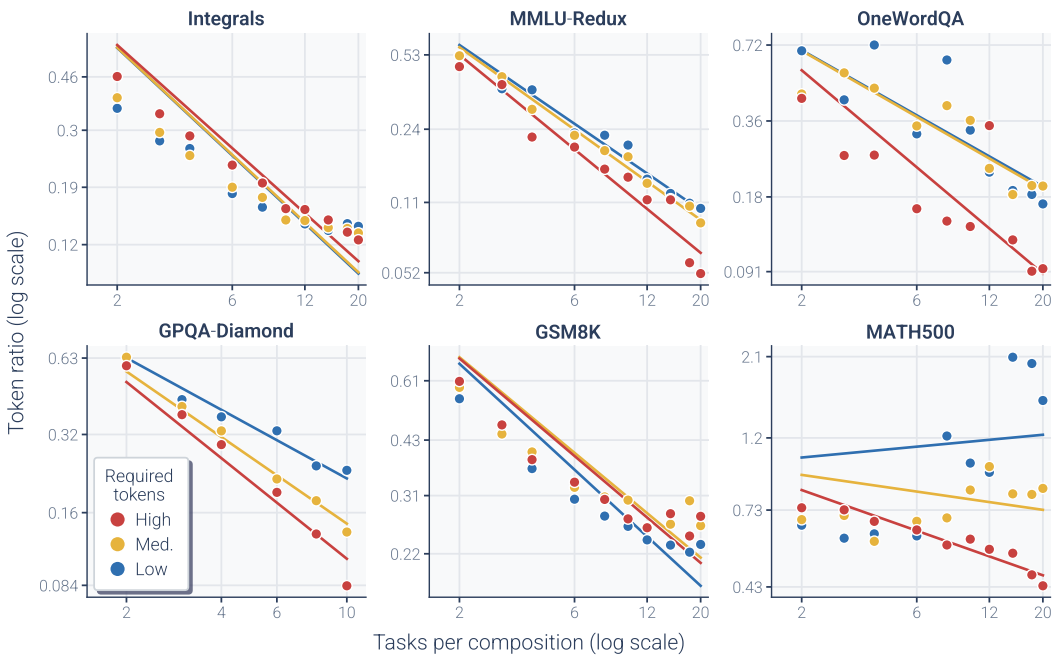


Figure 9: Token ratio on R1 Distill Qwen 7B. Power-law fit: $R^2 = 0.62$.

A.1.3 Token ratio influences accuracy degradation

Figures 10–13 extend the analysis of Figure 4 to further models. In each plot, tasks are binned by the quantile of their isolated thinking length $T_i(1)$ into Low ($\leq 20\%$), Medium (20–80%), and High ($> 80\%$), and the accuracy drop is shown as a function of the relative thinking token reduction. The pattern holds across all models. Accuracy drops as the reduction grows, and tasks with long isolated thinking traces degrade substantially more than tasks with short ones. We restrict this analysis to a subset of four models, chosen to span different training recipes and model sizes, as running it on all models would be prohibitively expensive.

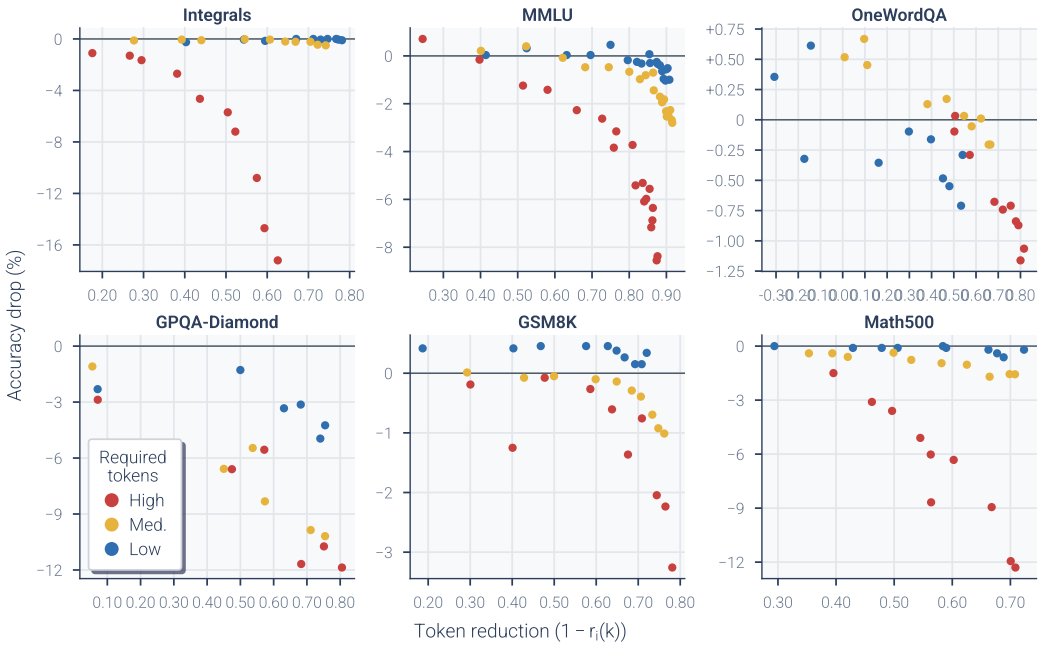


Figure 10: Accuracy drop versus relative thinking token reduction for Qwen3 4B Thinking.

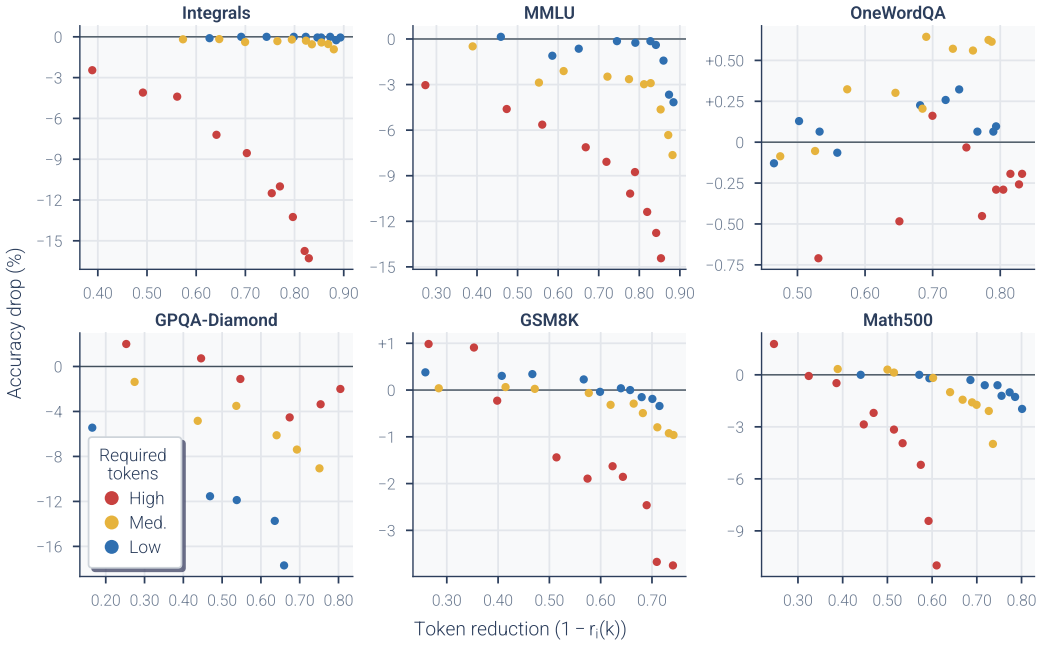


Figure 11: Accuracy drop versus relative thinking token reduction for Qwen3 8B.

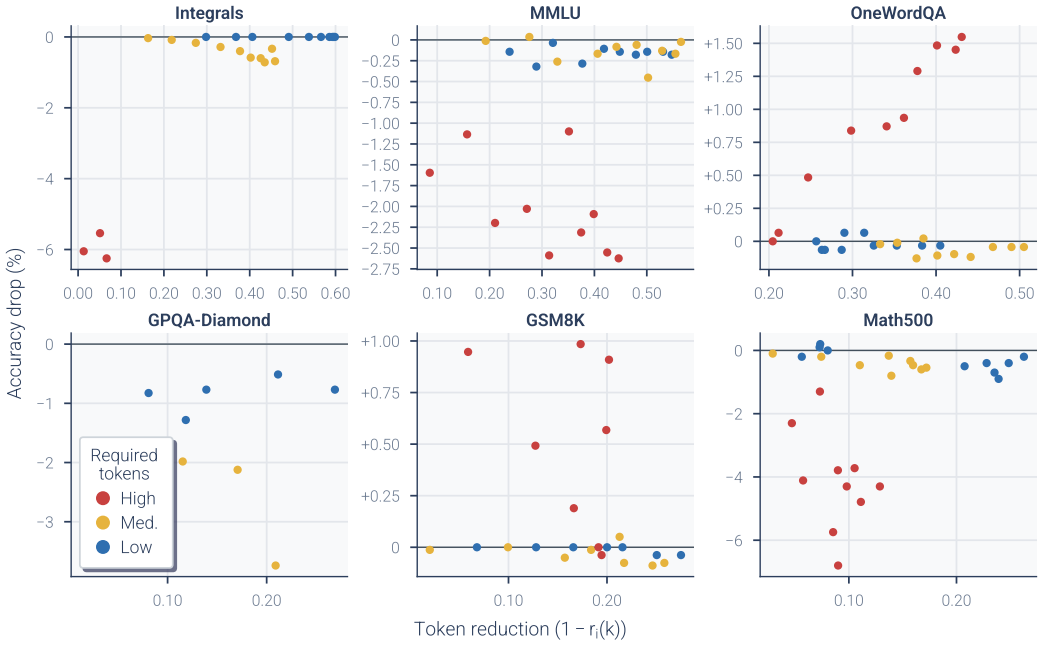


Figure 12: Accuracy drop versus relative thinking token reduction for Qwen3.8 27B.

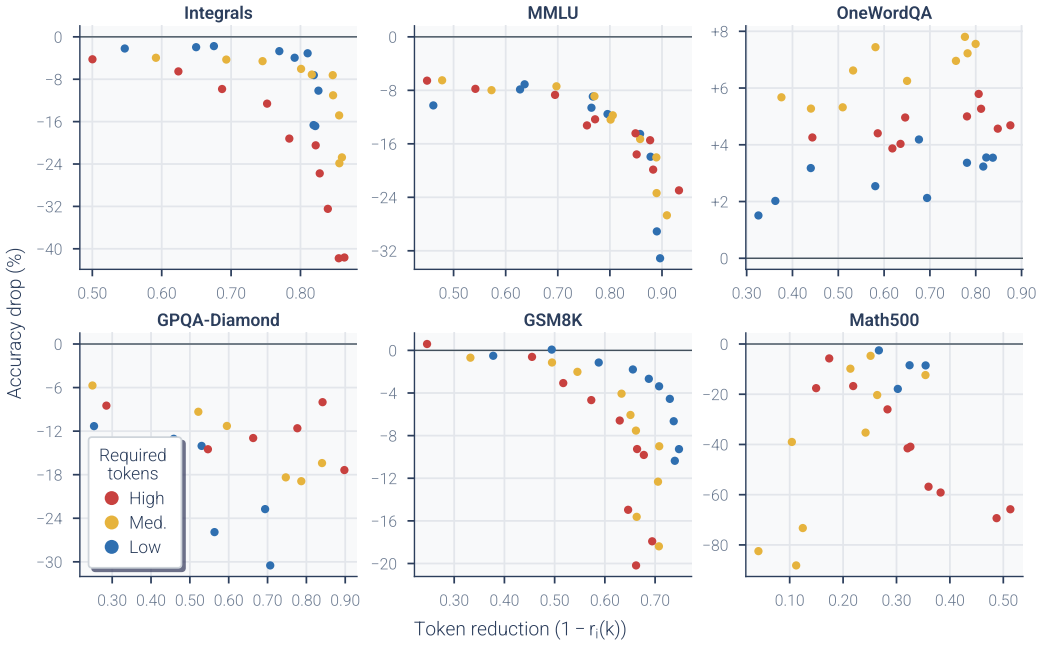


Figure 13: Accuracy drop versus relative thinking token reduction for R1 Qwen Distill 7B.

A.1.4 CORRELATING ACCURACY DROP ACROSS MODELS AND COMPOSITION SIZES

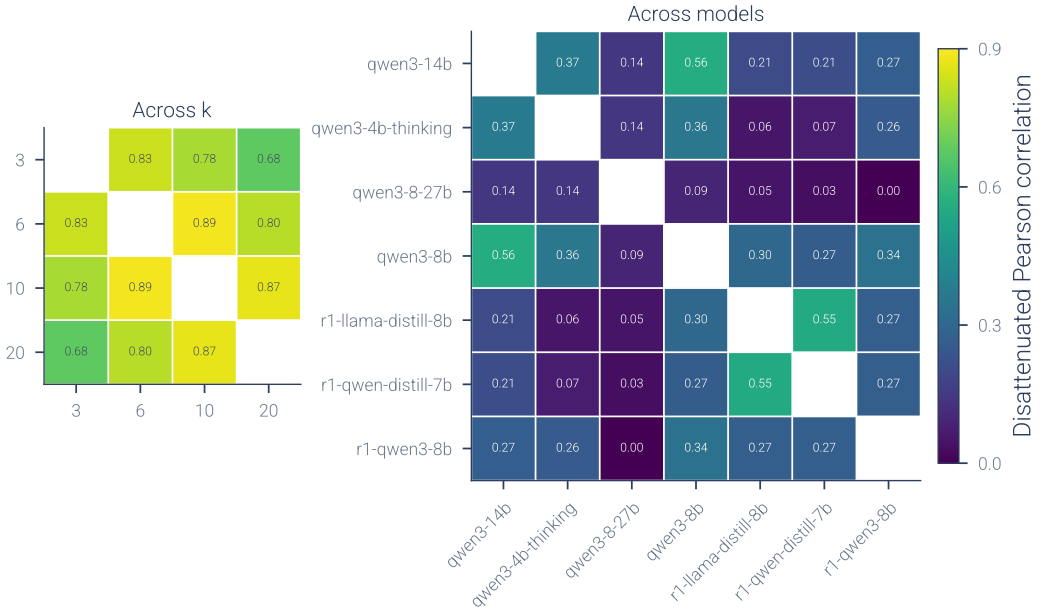


Figure 14: Disattenuated Pearson r correlation of accuracy degradation across different conditions, i.e. model and composition size. **Left:** The correlation is substantially larger when comparing accuracy degradation within the same model across different composition sizes k. **Right:** Contrary, the correlation between different models is low, especially across model families, suggesting different architectures and training recipes lead to different tasks being fragile.

A.1.5 FRAGILITY-AWARE BATCHING

Table 2 extends Table 1 to all seven local models. We note that we excluded AIME 24+25 from the experiments, because its compositional degradation across all tasks is too high to ensure the 0.5% accuracy drop budget, thus the fragility-batching is just defaulting to $k = 1$. To ensure comparability between datasets, we there did not include this dataset in the analysis. The pattern from the main text holds throughout. Random batching at k=6 saves many tokens but loses accuracy, most visibly on load-heavy domains. Fragility-aware batching keeps accuracy close to the isolated baseline while still using substantially fewer tokens, and by using the fragility predictions of the linear probe recovers part of this gain without access to measured fragility.

A.2 METHODOLOGICAL DETAILS

A.2.1 MEASURING PER-TASK THINKING LENGTH.

To measure how a model divides its thinking across the tasks of a composition, we prefix each task with a unique tag (`Q01`, `Q02`, ...) and instruct the model to name the tag whenever it turns to that task. Models follow this instruction reliably, so the tags mark where the reasoning on one task ends and the next begins. We assign every token after a tag to the corresponding task, up to the next tag or the end of the thinking trace. The thinking length $T_i(k)$ of task t_i in a composition of size k is the total number of tokens assigned to it, and the model may return to a task more than once.

```
<think> Q01:  The integrand splits into two parts ... so the
answer is B.
Q02:  Let x be the number of apples ... gives 42.
Back to Q01:  checking the sign again ... still B.
Q03:  ... </think>
```

Table 2: Item accuracy and output tokens per task under different batching policies for all models and datasets. All other columns report absolute values with the change with respect to k=1 in parentheses, as an absolute difference in percentage points for accuracy and as a relative change in % for tokens. Within each policy, the left column is accuracy and the right column is tokens. The Average row reports the mean over all seven models: absolute values for k=1 and the mean change for all other policies.

	Isolated, k=1		Random batching, k=6		Fragility-aware batching		Held-out (linear probe)	
	Acc.	Tok.	Acc.	Tok.	Acc.	Tok.	Acc.	Tok.
GPQA-Diamond								
Qwen3 4B T.	66.7%	7663	56.4% (−10.3)	2470 (−67.8%)	66.8% (+0.2)	5200 (−32.1%)	67.3% (+0.7)	7534 (−1.7%)
Qwen3 8B	58.7%	5973	53.4% (−5.3)	1950 (−67.3%)	60.0% (+1.3)	3520 (−41.1%)	56.9% (−1.8)	5106 (−14.5%)
Qwen3 14B	61.5%	4597	51.8% (−9.7)	1500 (−67.4%)	57.9% (−3.7)	2111 (−54.1%)	61.3% (−0.2)	3665 (−20.3%)
Qwen3.8 27B	83.2%	4303	79.2% (−4.0)	2740 (−36.3%)	84.0% (+0.8)	4102 (−4.7%)	85.3% (+2.1)	4324 (+0.5%)
R1 Qwen 7B	49.1%	6686	23.6% (−25.5)	1305 (−80.5%)	49.6% (+0.5)	4037 (−39.6%)	46.7% (−2.4)	5953 (−11.0%)
R1 Llama 8B	47.7%	7424	32.1% (−15.6)	1771 (−76.2%)	49.6% (+1.9)	4526 (−39.0%)	48.6% (+0.9)	5816 (−21.7%)
R1 Qwen3 8B	60.8%	9084	48.1% (−12.6)	2398 (−73.6%)	61.9% (+1.2)	6595 (−27.4%)	59.8% (−1.0)	8877 (−2.3%)
Average	61.1%	6533	−11.9	−67.0%	+0.3	−34.0%	−0.2	−10.1%
GSM8K								
Qwen3 4B T.	95.2%	1151	91.9% (−3.3)	472 (−59.0%)	93.4% (−1.8)	466 (−59.5%)	94.9% (−0.3)	793 (−31.1%)
Qwen3 8B	95.4%	769	95.2% (−0.2)	371 (−51.8%)	95.0% (−0.4)	279 (−63.8%)	95.1% (−0.3)	538 (−30.1%)
Qwen3 14B	96.1%	588	95.9% (−0.2)	253 (−56.9%)	95.9% (−0.2)	200 (−66.0%)	95.7% (−0.3)	314 (−46.6%)
Qwen3.8 27B	97.4%	263	97.5% (+0.1)	200 (−24.0%)	97.5% (+0.1)	158 (−40.1%)	97.5% (0.0)	157 (−40.1%)
R1 Qwen 7B	91.4%	834	87.5% (−3.9)	187 (−77.6%)	90.8% (−0.7)	308 (−63.1%)	91.6% (+0.1)	824 (−1.2%)
R1 Llama 8B	89.5%	685	81.8% (−7.8)	278 (−59.4%)	89.3% (−0.2)	672 (−1.9%)	89.4% (−0.1)	797 (+16.4%)
R1 Qwen3 8B	92.8%	2377	91.8% (−1.0)	794 (−66.6%)	93.0% (+0.2)	911 (−61.7%)	92.8% (0.0)	2163 (−9.0%)
Average	94.0%	952	−2.3	−56.5%	−0.4	−50.8%	−0.1	−20.3%
MATH-500								
Qwen3 4B T.	97.6%	4064	96.4% (−1.1)	2148 (−47.1%)	97.7% (+0.1)	3716 (−8.6%)	96.8% (−0.8)	3605 (−11.3%)
Qwen3 8B	95.9%	3449	95.6% (−0.3)	1751 (−49.2%)	95.1% (−0.8)	2173 (−37.0%)	95.9% (+0.1)	2775 (−19.6%)
Qwen3 14B	96.9%	2604	95.9% (−1.0)	1375 (−47.2%)	96.6% (−0.2)	2454 (−5.8%)	96.6% (−0.2)	2444 (−6.2%)
Qwen3.8 27B	98.1%	1014	96.7% (−1.5)	748 (−26.2%)	97.8% (−0.4)	943 (−7.1%)	97.2% (−0.9)	994 (−2.0%)
R1 Qwen 7B	93.0%	2383	69.8% (−23.2)	1658 (−30.4%)	93.2% (+0.2)	2205 (−7.5%)	93.4% (+0.4)	2381 (−0.1%)
R1 Llama 8B	88.7%	3019	79.4% (−9.3)	1569 (−48.0%)	88.5% (−0.2)	2948 (−2.3%)	88.8% (+0.2)	3333 (+10.4%)
R1 Qwen3 8B	96.8%	6767	93.0% (−3.8)	3705 (−45.2%)	96.6% (−0.2)	6608 (−2.4%)	96.7% (−0.2)	6657 (−1.6%)
Average	95.3%	3329	−5.7	−41.9%	−0.2	−10.1%	−0.2	−4.3%
Integrals								
Qwen3 4B T.	99.9%	1914	99.2% (−0.7)	1028 (−46.3%)	99.7% (−0.2)	1411 (−26.3%)	99.6% (−0.3)	908 (−52.6%)
Qwen3 8B	99.7%	2887	98.8% (−0.9)	946 (−67.2%)	99.8% (+0.1)	2717 (−5.9%)	98.8% (−0.9)	1192 (−58.7%)
Qwen3 14B	99.8%	2321	99.0% (−0.8)	670 (−71.1%)	99.5% (−0.3)	1336 (−42.5%)	99.6% (−0.2)	756 (−67.4%)
Qwen3.8 27B	99.8%	724	98.8% (−1.0)	564 (−22.1%)	99.5% (−0.3)	624 (−13.8%)	99.6% (−0.2)	550 (−24.0%)
R1 Qwen 7B	96.4%	2519	90.8% (−5.5)	547 (−78.3%)	96.0% (−0.4)	2277 (−9.6%)	96.3% (0.0)	2291 (−9.0%)
R1 Llama 8B	93.7%	2462	86.7% (−7.0)	613 (−75.1%)	93.1% (−0.6)	2161 (−12.2%)	93.6% (−0.1)	1926 (−21.8%)
R1 Qwen3 8B	98.5%	4082	96.7% (−1.8)	1339 (−67.2%)	97.9% (−0.6)	4344 (+6.4%)	98.5% (0.0)	2701 (−33.8%)
Average	98.3%	2416	−2.5	−61.0%	−0.3	−14.8%	−0.3	−38.2%
OneWordQA								
Qwen3 4B T.	92.0%	668	92.1% (+0.1)	384 (−42.6%)	91.7% (−0.3)	179 (−73.2%)	91.8% (−0.2)	185 (−72.4%)
Qwen3 8B	91.7%	212	92.0% (+0.3)	71 (−66.7%)	92.1% (+0.4)	46 (−78.2%)	92.2% (+0.5)	46 (−78.1%)
Qwen3 14B	92.4%	206	92.8% (+0.4)	73 (−64.6%)	93.2% (+0.7)	48 (−76.6%)	93.3% (+0.9)	47 (−77.1%)
Qwen3.8 27B	93.3%	71	93.4% (+0.1)	40 (−43.8%)	93.4% (+0.2)	36 (−50.0%)	93.4% (+0.2)	35 (−51.3%)
R1 Qwen 7B	72.0%	454	76.5% (+4.4)	106 (−76.8%)	76.5% (+4.4)	98 (−78.3%)	77.1% (+5.0)	89 (−80.4%)
R1 Llama 8B	78.0%	484	84.0% (+6.0)	86 (−82.2%)	83.5% (+5.5)	74 (−84.6%)	84.0% (+6.0)	81 (−83.3%)
R1 Qwen3 8B	81.5%	328	28.6% (−52.9)	136 (−58.6%)	82.1% (+0.6)	354 (+7.8%)	82.3% (+0.8)	322 (−1.7%)
Average	85.9%	346	−5.9	−62.2%	+1.6	−61.9%	+1.9	−63.5%
StrategyQA								
Qwen3 4B T.	63.2%	836	67.7% (+4.5)	509 (−39.1%)	68.2% (+5.0)	248 (−70.4%)	67.4% (+4.1)	246 (−70.6%)
Qwen3 8B	70.1%	394	72.1% (+1.9)	96 (−75.6%)	71.8% (+1.7)	57 (−85.4%)	70.8% (+0.7)	52 (−86.8%)
Qwen3 14B	70.0%	366	73.2% (+3.3)	84 (−77.1%)	73.5% (+3.5)	60 (−83.6%)	73.4% (+3.4)	54 (−85.3%)
Qwen3.8 27B	79.4%	224	80.2% (+0.8)	151 (−32.5%)	81.8% (+2.4)	146 (−35.0%)	81.0% (+1.6)	144 (−35.5%)
R1 Qwen 7B	56.9%	651	58.9% (+2.0)	84 (−87.1%)	58.5% (+1.6)	57 (−91.2%)	57.8% (+0.8)	47 (−92.8%)
R1 Llama 8B	58.9%	446	65.6% (+6.7)	81 (−81.9%)	65.2% (+6.3)	95 (−78.7%)	64.6% (+5.6)	51 (−88.5%)
R1 Qwen3 8B	70.0%	606	67.7% (−2.3)	89 (−85.4%)	71.1% (+1.1)	424 (−29.9%)	69.9% (−0.1)	629 (+3.8%)
Average	66.9%	503	+2.4	−68.4%	+3.1	−67.7%	+2.3	−65.1%
MMLU-Redux								
Qwen3 4B T.	84.7%	1944	82.9% (−1.8)	527 (−72.9%)	85.3% (+0.6)	527 (−72.9%)	84.9% (+0.2)	1663 (−14.5%)
Qwen3 8B	87.2%	921	84.0% (−3.3)	306 (−66.7%)	86.1% (−1.1)	285 (−69.0%)	86.9% (−0.3)	922 (+0.2%)
Qwen3 14B	89.0%	718	85.8% (−3.3)	212 (−70.5%)	88.2% (−0.9)	319 (−55.6%)	88.3% (−0.7)	678 (−5.5%)
Qwen3.8 27B	92.4%	488	91.9% (−0.5)	262 (−46.4%)	91.8% (−0.6)	196 (−59.8%)	92.1% (−0.3)	352 (−27.9%)
R1 Qwen 7B	70.1%	1078	60.6% (−9.5)	397 (−63.1%)	69.7% (−0.3)	601 (−44.3%)	70.4% (+0.3)	1127 (+4.5%)
R1 Llama 8B	75.5%	1458	58.6% (−16.9)	431 (−70.4%)	75.9% (+0.3)	896 (−38.5%)	75.3% (−0.2)	1430 (−1.9%)
R1 Qwen3 8B	81.6%	1710	64.5% (−17.1)	279 (−83.7%)	82.1% (+0.5)	1273 (−25.6%)	82.2% (+0.6)	1714 (+0.2%)
Average	82.9%	1188	−7.5	−67.7%	−0.2	−52.2%	−0.1	−6.4%

Here the tokens after both `Q01` tags count towards t_1, while t_2 receives the tokens between its tag and the second `Q01`.

However, this rule breaks down for the last task in a composition. Its segment runs until the thinking trace ends, and models often use these final tokens to re-check earlier answers or draft the final response. That work lands in the last segment even though it is unrelated to the last task. We therefore append a random dummy task from the same dataset at the final position and exclude it from the analysis, so that every task we measure is bounded by a tag on each side.

A.2.2 Sampling Parameters and Context Length

Every model is decoded with the same settings: temperature 0.6 and top-p 0.95, with no top-k cutoff and no repetition penalty. These are the values Qwen recommends for its models in thinking mode Yang et al. (2025a). We use them for every model because the experiments compare models against each other. Repeated trials of the same prompt share the engine's random seed, but no per-request seed is set, so each trial draws a fresh sample. The context window is $40{,}960$ tokens, shared by the prompt and the model's response, including its thinking. We found this limit to be enough for all experiments.

A.2.3 Correlating accuracy degradation across conditions

Conditions and degradation variables. We define a *condition* as a tuple $c = (m, k)$ of a model m and a composition size k, e.g. (Qwen3 8B, $k = 20$). For a condition $A = (m_A, k_A)$ we write Δ_A for the random variable that assigns to each task t_i its accuracy degradation $\Delta_i^{m_A}(k_A)$ under that condition, and analogously Δ_B for the second condition $B = (m_B, k_B)$ that we correlate against. All correlations below are computed over all tasks of a dataset $t_i \in \mathcal{D}$.

Stratification by isolated accuracy. Since accuracy in isolation may not be independent of the accuracy drop during composition, we stratify our analysis by it. Tasks are binned by $\mathrm{acc}_i^{m_A}(1)$ in steps of 0.2. Since the isolated accuracy is estimated from 5 independent trials, this is equivalent to binning by the number of successes in isolation $s_i \in \{0, 1, \ldots, 5\}$.

Within each bin b we compute the Pearson correlation

$$\rho_b = \mathrm{corr}(\Delta_A, \Delta_B \mid s_i = b)\,, \tag{4}$$

drop bins containing fewer than 10 tasks, and pool the remaining bins on the Fisher-z scale,

$$\bar{z} = \frac{\sum_b (n_b - 3)\,\mathrm{artanh}(\rho_b)}{\sum_b (n_b - 3)}, \qquad \bar{\rho} = \tanh(\bar{z}), \tag{5}$$

where n_b is the number of tasks in bin b. Because Δ_A and Δ_B are estimated from a finite number of trials, measurement noise attenuates $\bar{\rho}$ toward zero. We therefore apply Spearman's correction for attenuation,

$$\bar{\rho}^{\,*} = \frac{\bar{\rho}}{\sqrt{r_{AA}\, r_{BB}}}, \tag{6}$$

where r_{AA} and r_{BB} are the reliabilities of Δ_A and Δ_B, estimated as the correlation between two splits of trials (Spearman-Brown corrected). We report the back-transformed and disattenuated $\bar{\rho}^{\,*}$ througout.

Avoiding shared measurement noise. Because $\Delta_i^m(k) = \mathrm{acc}_i^m(k) - \mathrm{acc}_i^m(1)$, both degradation variables contain an isolated accuracy term. When A and B share the same model ($m_A = m_B$, e.g. when correlating across composition sizes), using a single estimate of $\mathrm{acc}_i^m(1)$ for both would introduce its estimation error as a common positive term into the covariance and inflate $\bar{\rho}$. We therefore always use two separate, independently measured estimates of $\mathrm{acc}_i^m(1)$ in this case, one for Δ_A and one for Δ_B. For cross-model comparisons the isolated estimates are model-specific and measured independently by construction, so no such correction is needed.

Confidence intervals. We report 95% confidence from a clustered bootstrap over tasks. In each of 5000 replicates we draw a sample of tasks of the original size with replacement and re-run the complete procedure on it, including the binning, the ≥ 10-task bin filter and the Fisher-z pooling, so that variability in bin composition and bin retention is propagated into the interval. We take the 2.5th and 97.5th percentiles of the resulting distribution of $\bar{\rho}^{\,*}$ as the lower and upper bounds.

A.2.4 IMPLEMENTATION DETAILS OF THE FRAGILITY-AWARE BATCHING ALGORITHM

Estimating $\mathbb{E}[\Delta_i(k)]$. For the fragility-aware batching intervention in Section 4.1 we estimate $\mathbb{E}[\Delta_i(k)]$ empirically as $\mathrm{acc}_i^m(k) - \mathrm{acc}_i^m(1)$ from the measured accuracies of 10 trials.

The uncertainty margin. The term $\mathrm{std}(\bar{\Delta}(\lambda))$ in Equation 3 is the standard error of the mean expected drop under the assignment $k(\lambda) = (k_1(\lambda), \ldots, k_n(\lambda))$,

$$\mathrm{std}\big(\bar{\Delta}(\lambda)\big) = \frac{1}{\sqrt{n}} \cdot \mathrm{sd}\Big(\big\{\mathbb{E}[\Delta_i(k_i(\lambda))]\big\}_{i=1}^{n}\Big), \tag{7}$$

that is, the sample standard deviation of the per-task expected drops selected by the assignment, scaled by $\sqrt{n}$.

Estimating cost_k. The ratio cost_k does not depend on the individual task and is estimated once per model and dataset as the mean number of output tokens attributable to a single task inside a composition of size k, divided by the mean number of output tokens used for isolated solving:

$$\mathrm{cost}_k = \frac{\frac{1}{k}\,\overline{T}(k)}{\overline{T}(1)}, \tag{8}$$

where $\overline{T}(k)$ is the mean total output length of a composition of size k. In each experiment, we estimate it on the training fold

We estimate it on a held-out subset of n_{cost} compositions per k, so that it requires no additional measurement on the tasks being assigned.

Selecting λ^* by repeated cross-fitting. Per-task accuracy estimates are noisy and our pool of tasks is limited, so a λ^* selected on the same data used to estimate $\mathbb{E}[\Delta_i(k)]$ would be optimistically biased. We therefore select the exchange rate by repeated 5-fold cross-fitting with isotonic smoothing.

We use nested cross-validation. The outer loop is a 5-fold split of the task pool. Within each outer training set, we run an inner 5-fold CV, repeated four times under different random shuffles. On each inner training fold we fit one isotonic map per $k \in \mathcal{K}$ to the measured accuracy and apply it to the corresponding held-out fold; since every map is fit on the training fold disjoint from the test fold it scores, the smoothing suppresses noise without leaking. Averaging over the four repetitions yields smoothed accuracy drop estimates for the outer training set.

We then find the optimal λ through a grid search of 121 exchange rates ($\lambda = 0$ together with 120 values spaced logarithmically over $[10^{-4}, 3.0]$). At each λ we assign a composition size to every task according to Equation 2 and record the realized accuracy and token ratio, tracing out accuracy as a function of λ. At the end, for each of the 5 outer fold, we have 4 λ against accuracy drop curves on that outer folds test fold.

These 20 curves are averaged per rate and the averaged curve is smoothed through isotonic regression in λ. This is valid because accuracy is non-increasing in λ by construction: a larger exchange rate can only move tasks to larger composition sizes, which empirically have larger expected drops. We then select the cheapest rate whose accuracy clears $\overline{\mathrm{acc}}(1) - B$ by one standard error, where $\overline{\mathrm{acc}}(1)$ is the mean isolated accuracy over the pool; if no rate on the grid clears this threshold, we fall back to the most accurate rate.

Finally, the isotonic map is refit on each task of the entire training fold, between one half of the 10 accuracy drop measurements and the other half, on the outer CV and applied to the test fold, and each task in the test fold is assigned its composition size at the selected λ^*. Only this test-fold assignment enters the reported results.

Forming compositions. The assignment yields a per-task upper bound $k_i(\lambda^*)$ on the size of the composition that task may be placed in, and compositions must be formed such that $|C| \leq \min_{t_i \in C} k_i(\lambda^*)$ for every composition C. Placing a task in a composition smaller than its assigned size is admissible, since its expected drop is then much more likely to be smaller than larger, so the realized mean drop can only fall below the planned one. We use the greedy procedure in Algorithm 1: proceeding from the smallest assigned size upwards, we form full compositions from

Algorithm 1 Forming compositions from a size assignment

Require: Tasks $t_1, \ldots, t_n$ with assigned sizes $k_i \in \mathcal{K}$
Ensure: A set of compositions $\mathcal{B}$
$\mathcal{B} \leftarrow \emptyset$; $U \leftarrow \{t_1, \ldots, t_n\}$
for $k \in \mathcal{K}$ in increasing order **do**
  $P \leftarrow \{\, t_i \in U : k_i = k \,\}$
  while $|P| \geq k$ **do**
    remove k randomly chosen tasks from P and from U, and add them as composition $\mathcal{B} \leftarrow \mathcal{B} \cup \{(t_1, ..., t_k)\}$
  end while
  if $P \neq \emptyset$ **then**
    $F \leftarrow$ up to $k - |P|$ tasks drawn uniformly at random from $\{\, t_i \in U : k_i > k \,\}$
    remove $P \cup F$ from U and add $P \cup F$ as a composition to $\mathcal{B}$
  end if
end for
return $\mathcal{B}$

the tasks assigned to the current size and fill any remaining partial composition with tasks drawn uniformly at random from those assigned to strictly larger sizes, which are by construction more robust and whose bound is therefore not violated.

If fewer than $k - |P|$ tasks with a larger assigned size remain, the leftover tasks form a composition smaller than k. Since filler tasks are consumed from the pools of larger assigned sizes, the realized mean composition size is slightly below the assigned one, so the reported token savings are a conservative estimate of what the assignment permits.

A.2.5 Predicting fragility from the task

Architecture. The predictor takes the task description and embeds it with Qwen3-Embedding-8B using last-token pooling. A linear head maps the embedding to a single logit $\ell_i(k_1)$, the logit for the smallest composition size $k_1 \in \mathcal{K}$. The logits for the remaining sizes are obtained by accumulating non-negative offsets along increasing k,

$$\ell_i(k_1) = w^\top e_i + b, \qquad \ell_i(k_j) = \ell_i(k_{j-1}) - \mathrm{softplus}(\theta_j), \quad j = 2, \ldots, |\mathcal{K}|, \tag{9}$$

where e_i is the embedding of task t_i, $k_1 < \cdots < k_{|\mathcal{K}|}$ are the composition sizes in $\mathcal{K} = \{3, 6, 10, 20\}$, and θ_j are trainable scalars shared across all tasks. The softplus constrains every offset to be non-negative, so the predicted probability is non-increasing in k by construction, matching the most behaviour established in Section 3. Since collecting accuracy degradation estimates from LLMs is expensive, the predictor must be trainable using only a small set of supervised data. Thus we chose a linear model since its parameter-efficient and encoded the monotonicity assumption of the accuracy degradation over k in its architecture which we observed holds for most tasks.

Training. For every task and every $k \in \mathcal{K}$ we form a binary target indicating that the task performs no worse under composition than in isolation, using the 10 composed and 10 isolated trials measured for that task. The head is trained with a binary cross-entropy loss summed over all composition sizes, so a single coefficient vector is shared across the ladder and the sizes differ only through the learned offsets. We optimize with AdamW at a learning rate of 3×10^{-3} and weight decay 1.0 for 2000 steps with batch size 64 and gradient clipping at 1.0.

Nested cross-fitting. The predictor and the exchange rate λ^* must both be estimated without access to the data on which the intervention is evaluated, so we nest the fitting of the head inside the cross-fitting scheme of Appendix A.2.4.

We split the dataset into 5 outer folds. For each outer fold, the $4/5$ training slice is split again into inner folds; one head is trained on each inner training split and predicts on the corresponding inner test split. Concatenating these predictions yields a fully out-of-training logit for every task in the

outer training slice, which takes the place of the empirically measured drops used in Appendix A.2.4. The remainder of the intervention is unchanged.

A.2.6 Further Details on Datasets

Indefinite integrals. We generate the integral pool ourselves, to ensure that every problem is known to be solvable and simple enough for the most small open source models to solve confidently. It holds 1,000 atomic integrands, 200 from each of five function classes:

- *polynomial*: $a\,x^n$;
- *exponential*: $a\,e^{bx}$;
- *trigonometric*: $a\,f(bx+c)$ with $f \in \{\sin, \cos, \tan\}$;
- *logarithmic and rational*: $a\ln(bx+c)$ or $a/(bx+c)$;
- *integration by parts*: $a\,x\,f(bx+c)$ with $f \in \{\exp, \sin, \cos, \ln, \arctan\}$.

The coefficient a is a nonzero integer in $[-5, 5]$. The exponent n and the frequency b are nonzero integers in $[-15, 15]$, and the shift c is an integer in $[-15, 15]$. All parameters are drawn with a fixed seed. Structural duplicates are discarded.

An integrand is kept only if SymPy finds an elementary antiderivative for it, and only if that antiderivative passes the same checker that grades the models. The checker works symbolically rather than by string matching. It parses the boxed LaTeX answer, differentiates it, and tests whether the derivative equals the integrand, first structurally, then numerically at random points, and finally with a symbolic proof under a timeout. An additive constant is accepted, but any other free symbol is rejected. Answers that cannot be parsed, or whose check times out, are recorded separately rather than counted as wrong.

OneWordQA. OneWordQA is a set of 1,548 questions that we generated using Claude Opus 5 and checked manually. They cover 23 categories of simple factual lookups whose answers leave little room for interpretation. Examples include country capitals, dialling and ISO codes, chemical symbols and atomic numbers, Roman numerals, polygon sides and the number of players per team in a sport. Each answer is a single word, a number or a short code. Grading normalises only how an answer is written (case, diacritics and surrounding punctuation). Where a question has more than one correct answer, the alternatives are listed explicitly in the dataset.

Benchmarks. The six public benchmarks are loaded from the Hugging Face Hub: MMLU-Redux, GPQA-Diamond, GSM8K, Math-500, AIME 24+25 and StrategyQA. In MMLU-Redux, we keep only questions that the annotators judged error-free, and draw the same number from each of 19 subjects (74 per subject, 1,406 questions). The answer option order is left as published.

Prompts. We now list the prompt every model receives, for each of the seven datasets used in the main experiments: indefinite integrals, MMLU-Redux, GPQA-Diamond, OneWordQA, GSM8K, MATH-500, AIME and StrategyQA.

Conventions shared by all datasets:

- An atomic task is treated as a composition of size $k = 1$ and rendered with the same template as a k-task composition.
- The prompt is sent as a single user message through the model's chat template, with no few-shot examples and no system message.
- The tasks are shown in the same order as the parser grades them. Thus LLMs are required to answer in the same order.

Indefinite integrals

A composition of k atomic integrands is shown as one integral whose integrand is their sum, rendered in LaTeX by SymPy. ⟨*integral*⟩ is that LaTeX string, for example `\int \left(x e^{x} + \sin^{2}{\left(x \right)}\right)\, dx`.

The prompt shows the whole sum unchanged, but the model reports one antiderivative per term. The terms are listed in the order in which they appear in the rendered integral. Answer i is checked symbolically against term i.

Indefinite integrals

```
Find an antiderivative for each term of the following indefinite
    integral: ⟨integral⟩.

The integrand is a sum of ⟨k⟩ terms, listed below in the order they
    appear in the integral:
1. \int ⟨term 1⟩ \, dx
2. \int ⟨term 2⟩ \, dx
...
⟨k⟩. \int ⟨term k⟩ \, dx
Do not add the antiderivatives together, and do not give an
    antiderivative of the whole sum. Solve each term on its own and
    report ⟨k⟩ separate antiderivatives, one per term, in the order
    above.

Give symbolic answers only in standard LaTeX format. Do not give a
    decimal approximation. The constant of integration may be omitted.
Return your answers in the following format in the last ⟨k⟩ lines,
    with no additional text: one line per term, in the order above,
    each line holding exactly one \boxed{}.

1. \boxed{<antiderivative of term 1>}
2. \boxed{<antiderivative of term 2>}
...
⟨k⟩. \boxed{<antiderivative of term ⟨k⟩>}

Before you finish, re-read your answer block and check that it holds
    exactly ⟨k⟩ lines, each with exactly one \boxed{}, in the order
    the terms are listed above.
```

MMLU-Redux and GPQA (multiple choice)

Both datasets use the same multiple-choice layout. Every question is followed by its lettered options, and consecutive questions are separated by a blank line:

Question block (⟨*question block*⟩) for both datasets

```
1. ⟨question 1⟩
A. ⟨option A⟩
B. ⟨option B⟩
C. ⟨option C⟩
D. ⟨option D⟩

2. ⟨question 2⟩
A. ...
```

In GPQA, the four options of each question are shuffled once, with a fixed seed, when the dataset is built, so the correct answer does not always sit in the same position. ⟨*example*⟩ is the letters offered, cycled to length k (for example `A, B, C, D, A, B` for $k = 6$). For $k = 1$ it is `A`.

MMLU-Redux ($k > 1$)

```
Answer the following multiple-choice questions.

⟨question block⟩

For each question, give only the letter of the single best option.
```

```
End your response with a line of the following form, and write
    nothing after it: a comma-separated list of ⟨k⟩ letters, one per
    question, in the order the questions were asked.
Answer: ⟨example⟩
```

MMLU-Redux ($k = 1$)

```
Answer the following multiple-choice question.

⟨question block⟩

For each question, give only the letter of the single best option.
End your response with a line of the following form, and write
    nothing after it.
Answer: A
```

The GPQA prompt is the MMLU-Redux prompt word for word, with one extra closing sentence:

GPQA: closing sentence appended to the MMLU-Redux prompt

```
Before you finish, re-read that final line and check that it holds
    exactly ⟨k⟩ letters, in the order the questions were asked, with
    nothing written after them.
```

For $k = 1$ this sentence reads "…holds exactly 1 letter, with nothing written after it."

One-word QA and StrategyQA

Both datasets use the same layout. Each question sits on a single line, numbered, and consecutive questions are separated by a single newline (⟨*question block*⟩). The answer is a numbered block of k lines under the heading `Answer:`.

The StrategyQA prompt is the one-word QA prompt word for word, apart from two sentences that restrict the answer to yes or no:

StrategyQA: the two sentences that differ from one-word QA

```
Answer the following questions. Each answer is either yes or no.
...
Give the answer only, as the single word yes or no: no explanation,
    and no answer other than those two words.
```

The instruction is not enforced strictly. Case and punctuation are ignored, and `true` and `false` are accepted alongside `yes` and `no`. For $k = 1$ the first sentence reads "Answer the following question."

GSM8K, MATH-500 and AIME (short mathematical answers)

The three datasets use the same answer contract as one-word QA: a numbered block of k lines under the heading `Answer:`. The difference is in the problems themselves, which are much longer (several sentences, displayed equations, and for 42 MATH-500 problems an Asymptote figure). Consecutive problems are therefore separated by a blank line rather than a single newline. Blank lines inside a problem are collapsed when the pool is built, so a blank line in ⟨*problem block*⟩ always marks the start of the next problem. For $k = 1$ the answer block has the single line `1. <your answer>` (for MATH-500, `1. \boxed{<your answer>}`), and the closing sentence reads "…holds exactly one answer line, with nothing written after it." The three prompts differ only in their first line and in the sentence that describes the answer format.

GSM8K ($k > 1$)

```
Solve the following problems. Each answer is a single number.

⟨problem block⟩

Give the final answer to each problem as a number only: no units, no
    currency symbol, no thousands separator and no explanation.
End your response with an answer block: the line Answer: followed by
    exactly ⟨k⟩ lines, one answer per line, numbered, in the order the
    problems were asked. Write nothing after the block.

Answer:
1. <your answer to problem 1>
2. <your answer to problem 2>
...
⟨k⟩. <your answer to problem ⟨k⟩>
Before you finish, re-read your answer block and check that it holds
    exactly ⟨k⟩ answer lines, in the order the problems were asked,
    with nothing written after them.
```

In MATH-500 each answer is boxed. Many answers contain spaces or commas, such as tuples, intervals, or `10,\!080`, so the box marks exactly where an answer begins and ends. The instruction "in the form the problem asks for" matters because many MATH-500 problems set their own answer format (for example "Express your answer as a common fraction"), and the reference answer follows that format.

MATH-500 ($k > 1$)

```
Solve the following problems.

⟨problem block⟩

Give the final answer to each problem only, with no explanation.
    Write each answer in LaTeX, in the form the problem asks for,
    wrapped in \boxed{...}.
End your response with an answer block: the line Answer: followed by
    exactly ⟨k⟩ lines, one answer per line, numbered, in the order the
    problems were asked. Write nothing after the block.

Answer:
1. \boxed{<your answer to problem 1>}
2. \boxed{<your answer to problem 2>}
...
⟨k⟩. \boxed{<your answer to problem ⟨k⟩>}
Before you finish, re-read your answer block and check that it holds
    exactly ⟨k⟩ answer lines, in the order the problems were asked,
    with nothing written after them.
```

AIME ($k > 1$)

```
Solve the following competition problems. Each answer is an integer
    between 0 and 999.

⟨problem block⟩

Give the final answer to each problem as an integer between 0 and
    999: digits only, no units, no thousands separator and no
    explanation.
```

```
End your response with an answer block: the line Answer: followed by
    exactly ⟨k⟩ lines, one answer per line, numbered, in the order the
    problems were asked. Write nothing after the block.

Answer:
1. <your answer to problem 1>
2. <your answer to problem 2>
...
⟨k⟩. <your answer to problem ⟨k⟩>
Before you finish, re-read your answer block and check that it holds
    exactly ⟨k⟩ answer lines, in the order the problems were asked,
    with nothing written after them.
```

For $k \leq 3$ the example answer block in all three prompts lists every line, with no ellipsis.