THANKS FOR FAILING: BETTER TEST-TIME SEARCH FROM WRONG ATTEMPTS

Anonymous authors

ABSTRACT

Test-time scaling (TTS) methods are central to both LLM evaluation and post-training pipelines. They not only improve inference-time accuracy but also support rejection sampling and search- or verifier-guided trajectory generation for reinforcement learning. For rollout generation, TTS methods can often be decomposed into two primitives: independent sampling from a shared context, and sequential sampling conditioned on previous attempts. Recent work finds that independent sampling often outperforms sequential sampling, yet we observe that these two primitives have competing strengths. Independent sampling explores more diverse solutions, but cannot leverage prior failures. Sequential revision depends on prior failures, but becomes less diverse and is often trapped by failed reasoning. In this paper, we systematically study four methods to combine the predictions of independent and sequential sampling strategies. Through this analysis, we propose **Failure Logit Extrapolation** (**FLEx**), a simple, training-free, contrastive decoding algorithm that extrapolates the difference between token-level logits generated independently and sequentially. Across six math, coding, and instruction-following benchmarks, and eight models ranging from 4B to 235B parameters, FLEx improves solve@8 over independent sampling by 3.3pp and 11.5pp over sequential sampling on average. On the four mathematics benchmarks, it also increases answer diversity, measured by the fraction of unique answers, by 5.9pp; on problems that all three methods fail to solve, this diversity advantage grows to 7.3pp. Its gains consistently increase as more failed attempts are retained in its context and as more sampling budget is provided. Finally, we show that FLEx directly benefits post-training pipelines. When used for rejection sampling, FLEx-collected trajectories result in higher downstream accuracy than independently sampled trajectories.

1 INTRODUCTION

Large language models are increasingly used to solve reasoning-intensive problems in mathematics, (Hendrycks et al., 2021; Balunovic et al., 2026a), coding, (Merrill et al., 2026), STEM, (Phan et al., 2025; Zhu et al., 2025), and open-ended optimization (Chan et al., 2025; Mang et al., 2025; Ouyang et al., 2025). Although candidate solutions to these problems are often inexpensive to verify, generating them requires reasoning and iterative refinement (Wei et al., 2022; Yao et al., 2023; Besta et al., 2024; Novikov et al., 2025; Lange et al., 2026). Test-time scaling (TTS) therefore plays an increasingly important role in improving inference-time accuracy (Snell et al., 2024; Muennighoff et al., 2025). The same sampling and search primitives are also central to post-training, where they generate trajectories for rejection sampling (Gulcehre et al., 2023; Xiong et al., 2025) and reinforcement learning (Wu et al., 2025b; Zhang et al., 2024; Kang & Zhang, 2025).

Rollout generation largely relies on two primitives. **Independent sampling** (also called *pass@k*) generates multiple parallel solutions from the original problem context. **Sequential sampling**, which we denote as *rev@k*, conditions each new attempt on previous attempts and their feedback. Recent work finds that naïve sequential revision often underperforms independent sampling. This is because prior attempts can bias subsequent generations to similar errors (Cheng et al., 2026), reduce exploration (Gu et al., 2026), and reinforce existing behavior (Simhi et al., 2026). Independent sampling avoids this failure-induced bias, but it also cannot leverage earlier attempts and may repeatedly explore the same dominant reasoning modes (Wu et al., 2025a; Wang et al., 2025).

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.

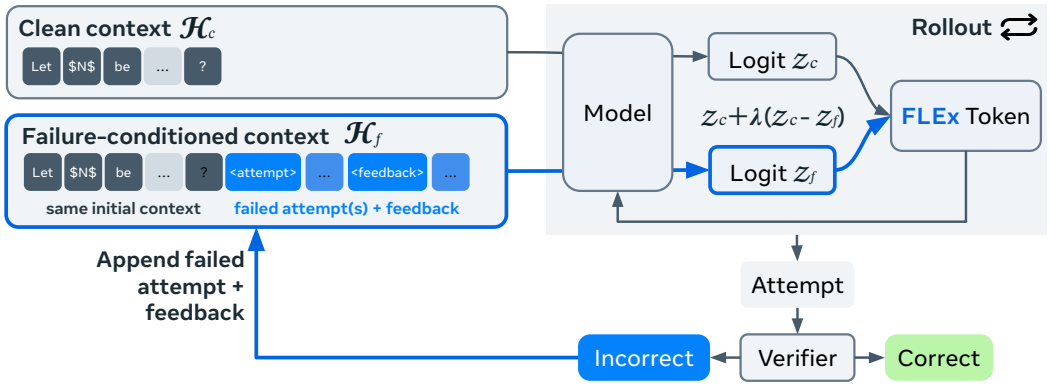

Figure 1: **By default, FLEx decodes a fresh attempt away from the model's own failures**. The same model is run on two contexts: a clean context holding only the problem statement that resembles independent sampling, and a failure-conditioned context holding the problem plus previous failed attempt and its feedback that resembles sequential revision. Every token is sampled from the extrapolation of logits produced by these two contexts. Verified-correct attempts end the chain, while incorrect ones are appended with the feedback to the failure-conditioned context. Both branches share one set of model weights, so no auxiliary model is needed.

Our experiments expose a fundamental tension between these primitives. Independent sampling produces more diverse solutions, but cannot use prior failures. Sequential revision has access to those failures, but produces less diverse solutions and is frequently trapped by failed reasoning. The central question of this work is therefore: *Can a model learn from failed experience without becoming trapped by it?*

We systematically study four ways (§ 3.1) of combining the model's predictions with and without failed attempts in context. This analysis leads to **Failure Logit Extrapolation (FLEx)**, a simple, training-free contrastive decoding method. At each decoding step, FLEx compares logits from a clean context against logits conditioned on previous failures, then extrapolates away from the predictive shift induced by those failures while retaining only tokens plausible under the clean policy. Unlike contrastive decoding methods that compare different models (Li et al., 2023), different layers (Chuang et al., 2024), or conditional and unconditional predictions (Ho & Salimans, 2022), FLEx constructs its contrast from the model's own search history.

Across six mathematics, coding, and instruction-following benchmarks and eight models ranging from 4B to 235B parameters, FLEx achieves the strongest average accuracy relative to both sampling primitives. More importantly, FLEx combines and improves their complementary strengths: it uses prior failures like sequential sampling while producing greater solution diversity than independent sampling. Its accuracy also improves as more failures are retained and continues scaling with larger attempt budgets, demonstrating that FLEx can continue extracting benefit from more failures.

Sequential sampling can also be strengthened by providing richer signals, such as environmental feedback, rewards, or hints (Madaan et al., 2023; Shinn et al., 2023; Gou et al., 2024). By passing environment feedback for competitive coding evaluations, we show that a simple directional generalization of FLEx can exploit this informative shift and outperform feedback-augmented sequential baselines. Thus, FLEx is not restricted to settings where revision is weak: its underlying contrast can also complement deliberately engineered sequential-sampling methods.

Finally, we demonstrate that FLEx benefits post-training directly. When used to collect trajectories for rejection sampling, FLEx produces training data that yields higher downstream accuracy than independently sampled trajectories under matched training-token budgets. This connects improved test-time exploration to a practical improvement in the models trained from those rollouts. Our contributions are threefold:

- We analyze the complementary strengths of independent and sequential sampling and study four ways to combine their predictions, leading to **FLEx**, a training-free method that converts failed attempts into token-level guidance without sacrificing exploration.

- We show across eight models and six benchmarks that FLEx improves solve@8 over independent sampling by 3.3pp on average. On four mathematics benchmarks, it increases answer diversity, as measured by the fraction of unique solutions, by 5.9pp, with the diversity advantage growing to 7.3pp on problems that all methods fail to solve. FLEx also scales with accumulated failures and sampling budget, and generalizes to sequential methods strengthened by actionable feedback.
- We show that FLEx-generated trajectories improve post-trained models using rejection-sampling: under matched training-token budgets, the all-samples setting improves average out-of-distribution pass@8 by 8.7pp compared to training on independently sampled trajectories.

2 Failure Logit Extrapolation (FLEx)

We introduce the default form of **Failure Logit Extrapolation (FLEx)**, a training-free decoding method that uses failed attempts as a negative guidance signal for subsequent generation (Fig. 1, Alg. 1). The method is built upon the hypothesis that if conditioning on a failed attempt shifts the model's next-token distribution in some direction, then extrapolating in the opposite direction may steer generation away from the failure mode induced by that attempt. FLEx realizes this idea by maintaining two parallel contexts and contrasting their token-level predictions during decoding. We later consider a directional variant for settings in which the failure-conditioned context contains sufficiently informative corrective feedback (§ A.5).

2.1 Clean and Failure-Conditioned Contexts

Consider a problem with initial context x, containing the system prompt and problem statement. FLEx maintains two copies of the context, H_c and H_f, which denote the clean context and the failure-conditioned context respectively. For the first attempt, the two contexts are identical, i.e., $H_c = H_f = x$, so FLEx reduces to ordinary sampling.

Suppose an attempt $y^{(1)}$ is generated and determined by a verifier to be incorrect. For subsequent attempts, we preserve H_c as the original problem context, while appending the failed attempt and its feedback to H_f. For example, after the first failure, $H_f = [x; y^{(1)}; f^{(1)}]$, where $f^{(1)}$ denotes the corresponding verifier feedback. As additional attempts fail, their trajectories and feedback are likewise accumulated in H_f (e.g., $H_f = [x; y^{(1)}; f^{(1)}; y^{(2)}; f^{(2)}; ...; y^{(n)}; f^{(n)}]$, while H_c remains unchanged (i.e., $H_c = x$).

Algorithm 1 Failure Logit Extrapolation

Require: context x, model M, verifier V, guidance strength λ, plausibility threshold α, attempt budget K

$\mathcal{H}_c \leftarrow x$
$\mathcal{H}_f \leftarrow x$
for $i = 1, \ldots, K$ **do**
  $y^{(i)} \leftarrow [\,]$
  while $y^{(i)}$ is not terminated **do**
    $z_c \leftarrow M(\mathcal{H}_c; y^{(i)})$
    $z_f \leftarrow M(\mathcal{H}_f; y^{(i)})$
    $z \leftarrow z_c + \lambda(z_c - z_f)$
    $z \leftarrow \mathrm{Mask}_\alpha(z; z_c)$
    $w \sim \mathrm{softmax}(z)$
    $y^{(i)} \leftarrow [y^{(i)}; w]$
  end while
  $(r^{(i)}, f^{(i)}) \leftarrow V(x, y^{(i)})$
  if $r^{(i)}$ is correct **then**
    return $y^{(i)}$
  end if
  $\mathcal{H}_f \leftarrow [\mathcal{H}_f; y^{(i)}; f^{(i)}]$
end for

The two contexts therefore provide useful complementary views of the same problem. The clean context captures how the model would continue without observing prior failed attempts, while the failure-conditioned context captures how the model behaves after observing its own failed attempts. FLEx uses this difference as a token-level signal to steer the next attempt.

2.2 Failure Logit Extrapolation (FLEx)

At each decoding step of an attempt, FLEx runs the same model on both contexts H_c and H_f. Let $\mathbf{z}_c$ and $\mathbf{z}_f$ denote the next-token logits produced from these contexts, respectively. To steer the model generation, we construct the guided logits as $\mathbf{z}_{\text{FLEX}} = (1 + \lambda)\mathbf{z}_c - \lambda\mathbf{z}_f$, where $\lambda \geq 0$ controls the strength of failure guidance. We then sample from $\mathbf{z}_{\text{FLEX}}$ and append the sampled token to both H_c and H_f. Equivalently, FLEx starts from the clean prediction and moves in the direction opposite to the predictive shift induced by the failure-conditioned context. When $\lambda = 0$, FLEx reduces to ordinary sampling from the clean context; increasing λ more strongly steers generation

away from behaviors associated with observing previous failures. This formulation allows failed attempts to influence subsequent generations without asking the model to reason directly from them. Instead, prior failures provide an instance-specific contrastive condition whose effect on the model distribution can guide subsequent search. Following contrastive decoding (Li et al., 2023), we apply the plausibility mask α on $\mathbf{z}_{\text{FLEX}}$ based on a threshold relative to the top logit from $\mathbf{z}_c$.

Once the current attempt terminates, it is evaluated by the verifier. If it succeeds, sampling stops. If it fails, the current attempt and its feedback are added to H_f, and FLEx is applied again for the next revision with H_c and H_f. In our default setting, H_f accumulates all previous failures, allowing the guidance signal to incorporate an increasingly complete history of unsuccessful searches. When the history approaches the context limit, we randomly subsample a fixed number of prior failures; for example, retaining seven failures allows FLEx to scale to $k = 256$ attempts (Fig. 5).

Unlike prior contrastive guidance methods that construct the contrast from different models (O'Brien & Lewis, 2023; Li et al., 2023), layers (Chuang et al., 2024), or predefined conditioning signals (Ho & Salimans, 2022), FLEx constructs it online from the same model's own failure history.

3 Experiments

We empirically study FLEx along four dimensions: whether failure-guided decoding improves sequential test-time scaling across model scales, families, and domains (§ 3.2, Fig. 2, Tab. 1); whether FLEx explores a more diverse set of candidate solutions than independent sampling and sequential revision (§ 3.2, Fig. 3); whether its gains persist as the number of attempts (§ 3.4, Fig. 5) or retained failures increases (§ 3.3, Fig. 4); and whether FLEx-generated trajectories improve downstream post-trained models through rejection sampling (§ 4, Tab. 2).

3.1 Experimental Setup

Models: We evaluate the Qwen3 family (Yang et al., 2025) across a wide range of model scales and architectures, including 4B, 8B, 14B and 32B dense models and 30B-A3B and 235B-A22B mixture-of-experts models. At the 8B scale, we additionally evaluate on Ministral 3 (Liu et al., 2026a) and Granite 3.3 (Granite Team, 2024) (more details in § A.1).

Tasks: We consider three domains with programmatically verifiable outcomes. For mathematical reasoning, we evaluate models on AIME 2025 (avg@5) (Balunovic et al., 2026b), AIME 2026 (avg@5) (Balunovic et al., 2026b), HMMT Feb 2026 (avg@5) (Balunovic et al., 2026b) and IMO-AnswerBench (Luong et al., 2025), and the feedback consists only of the correctness of the answer. For instruction following, we use the 300 examples from IFBench (Pyatkin et al., 2025), and the feedback consists of a list of instruction violations. For coding, we evaluate on the 1,055 problems in LiveCodeBench v6 (Jain et al., 2025), and the feedback consists of interpreter evaluations on the test cases (more details in § A.2 and § A.3).

Baselines and metrics: We first compare FLEx against its two source methods: independent sampling (i.e., pass@(k)) and sequential revision (i.e., rev@(k)). We then evaluate three baselines that combine these two sources in different ways. These baselines are in part inspired by existing or concurrent work (Li et al., 2023; Pei et al., 2026). They also serve as ablations of key FLEx design choices, including the use of instance-specific failures and the use of extrapolation. The interpolation baseline tests whether the logits from the two sources should be interpolated rather than extrapolated. The contrastive decoding baseline (Li et al., 2023) tests whether directly subtracting the two logits is sufficient, instead of extrapolating from one toward the other. We further study how its negative branch should be constructed. Specifically, we compare a fixed negative prompt (see § A.6 for details) against online failures produced by the model on the specific problem instance. We report cumulative *solve@8*, defined as the percentage of problems for which a method produces at least one verified solution within its first eight attempts.

3.2 Results

We show the performance of FLEx against its 5 baselines across 8 models and 6 tasks in Tab. 1. We additionally provide the average metric over the 6 tasks and the confidence interval on the difference

Table 1: **Test-time scaling across model sizes and families on evaluations across three domains.** We compare independent sampling (pass), sequential revision (rev), interpolation of the two, subtractive contrastive decoding with either a fixed negative prompt or online failures, as well as FLEx. Bold benchmark values denote the best result for each model and benchmark, and shaded rows denote FLEx. The final column reports the 95% confidence interval of the sample-matched difference from independent sampling; intervals are bolded when lower bound is positive (Details in § A.7).

Method	Online Failures	Logit Operator	Math				IF	Coding	Avg.	Δ vs. Indep. 95% CI
			AIME25	AIME26	HMMT26	IMO	IFBench	LCBv6		
Qwen-3 4B										
Independent Sampling	–	–	38.00	40.00	30.30	42.00	44.00	48.53	40.47	–
Sequential Revision	–	–	24.67	23.33	22.42	25.25	36.83	49.67	30.36	[-12.98,-7.57]
Interpolation	✓	Interpolation	30.67	30.67	27.88	32.75	32.33	46.16	33.41	[-8.74,-4.89]
Contrastive Decoding	✗	Subtraction	44.00	38.00	33.33	40.75	41.00	45.50	40.43	[-1.22,+2.14]
Contrastive Decoding	✓	Subtraction	**45.33**	**44.67**	33.33	42.25	44.00	50.05	**43.27**	**[+0.46,+5.14]**
FLEx	✓	Extrapolation	44.00	38.00	**35.15**	**43.75**	**46.00**	**51.18**	43.01	**[+0.31,+4.77]**
Granite 3.3 8B										
Independent Sampling	–	–	16.00	11.33	4.24	22.00	41.67	26.07	20.22	–
Sequential Revision	–	–	6.67	4.00	3.64	14.75	33.83	17.35	13.37	[-9.14,-4.83]
Interpolation	✓	Interpolation	15.33	8.00	6.67	16.50	33.67	21.52	16.95	[-4.99,-1.78]
Contrastive Decoding	✗	Subtraction	**18.67**	12.00	6.67	22.00	39.67	24.83	20.64	[-1.46,+1.83]
Contrastive Decoding	✓	Subtraction	16.00	13.33	8.48	24.75	**46.67**	24.36	22.27	[-0.01,+3.39]
FLEx	✓	Extrapolation	18.00	**14.00**	**9.70**	**25.75**	**46.67**	**26.92**	**23.51**	**[+1.34,+5.02]**
Ministral 3 8B										
Independent Sampling	–	–	48.67	46.67	35.15	42.50	46.00	59.24	46.37	–
Sequential Revision	–	–	34.00	36.67	26.67	27.75	45.00	54.03	37.35	[-11.33,-6.70]
Interpolation	✓	Interpolation	47.33	53.33	30.30	36.75	46.00	57.82	45.26	[-3.33,+1.10]
Contrastive Decoding	✗	Subtraction	**52.00**	47.33	32.73	42.00	45.33	60.28	46.61	[-1.77,+2.25]
Contrastive Decoding	✓	Subtraction	42.00	50.00	33.33	41.25	45.33	58.58	45.08	[-2.98,+0.40]
FLEx	✓	Extrapolation	48.67	**56.00**	**35.76**	**43.75**	**46.67**	**61.61**	**48.74**	**[+0.52,+4.22]**
Qwen-3 8B										
Independent Sampling	–	–	34.67	42.67	34.55	42.50	42.67	54.60	41.94	–
Sequential Revision	–	–	26.67	29.33	26.67	28.50	37.33	54.69	33.87	[-10.51,-5.26]
Interpolation	✓	Interpolation	36.00	32.00	30.91	36.25	38.00	53.36	37.75	[-6.31,-1.90]
Contrastive Decoding	✗	Subtraction	38.00	39.33	**36.97**	42.50	43.00	50.81	41.77	[-2.52,+2.34]
Contrastive Decoding	✓	Subtraction	41.33	**48.00**	32.12	40.75	45.67	55.92	43.97	[-0.25,+4.46]
FLEx	✓	Extrapolation	**42.67**	41.33	33.94	**44.25**	**48.00**	**56.78**	**44.50**	**[+0.52,+4.74]**
Qwen-3 14B										
Independent Sampling	–	–	47.33	48.00	35.76	44.25	45.00	58.48	46.47	–
Sequential Revision	–	–	32.67	36.67	25.45	35.50	40.67	60.19	38.52	[-10.77,-4.76]
Interpolation	✓	Interpolation	38.00	42.67	35.15	39.25	41.33	56.68	42.18	[-6.48,-2.07]
Contrastive Decoding	✗	Subtraction	47.33	51.33	36.97	43.75	47.33	55.92	47.10	[-1.12,+2.43]
Contrastive Decoding	✓	Subtraction	50.67	52.00	**37.58**	**46.25**	47.67	**64.27**	49.74	**[+0.93,+5.64]**
FLEx	✓	Extrapolation	**52.67**	**58.67**	35.76	46.00	**50.67**	62.75	**51.09**	**[+2.33,+6.92]**
Qwen-3 32B										
Independent Sampling	–	–	41.33	49.33	36.36	47.50	50.67	62.75	47.99	–
Sequential Revision	–	–	34.67	32.67	30.91	36.00	47.00	64.55	40.97	[-9.75,-4.67]
Interpolation	✓	Interpolation	39.33	47.33	32.73	42.00	47.33	63.51	45.37	[-4.22,-1.02]
Contrastive Decoding	✗	Subtraction	40.00	49.33	33.33	42.50	47.67	57.91	45.12	[-4.97,-0.76]
Contrastive Decoding	✓	Subtraction	48.00	51.33	36.97	45.25	52.33	62.94	49.47	[-0.98,+3.94]
FLEx	✓	Extrapolation	**51.33**	**54.00**	**38.18**	**48.00**	**53.33**	**64.74**	**51.60**	**[+1.26,+5.95]**
Qwen-3 30B-A3B (MoE)										
Independent Sampling	–	–	46.67	50.00	31.52	44.75	47.67	57.73	46.39	–
Sequential Revision	–	–	29.33	34.67	25.45	29.50	39.83	59.91	36.45	[-12.79,-7.45]
Interpolation	✓	Interpolation	40.00	44.00	27.88	39.00	40.00	54.79	40.95	[-7.69,-3.66]
Contrastive Decoding	✗	Subtraction	46.67	57.33	35.15	**47.50**	45.33	53.93	47.65	[-1.00,+3.57]
Contrastive Decoding	✓	Subtraction	48.00	**59.33**	32.12	45.75	51.00	**63.51**	49.95	**[+1.36,+5.64]**
FLEx	✓	Extrapolation	**48.67**	54.67	**36.97**	45.50	**55.00**	62.09	**50.48**	**[+1.93,+6.13]**
Qwen-3 235B-A22B (MoE)										
Independent Sampling	–	–	46.67	62.00	35.15	51.00	52.33	68.63	52.63	–
Sequential Revision	–	–	40.00	44.00	29.70	43.25	48.67	70.90	46.09	[-9.39,-4.24]
Interpolation	✓	Interpolation	45.33	56.67	35.15	45.25	52.33	69.76	50.75	[-4.30,+0.07]
Contrastive Decoding	✗	Subtraction	**54.67**	60.00	38.18	**52.25**	52.33	65.40	53.81	[-1.17,+3.05]
Contrastive Decoding	✓	Subtraction	44.67	58.67	37.58	50.25	53.67	69.67	52.42	[-2.60,+1.70]
FLEx	✓	Extrapolation	53.33	**62.67**	**41.82**	51.50	**56.33**	**71.85**	**56.25**	**[+1.06,+5.71]**

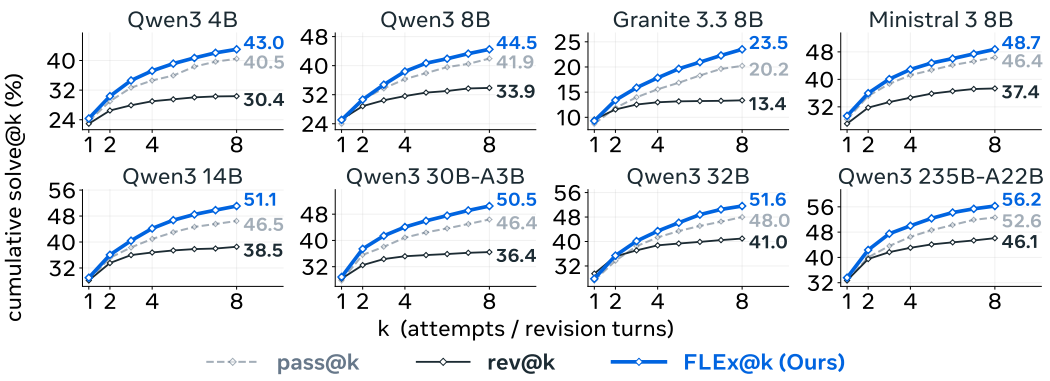

Figure 2: **Cumulative solve rate for test-time scaling among FLEx, independent sampling and sequential revision.** Averaged across 6 evaluations in 3 reasoning domains, FLEx beats both baselines at every scale, in both dense and MoE Qwen3 models, and in two other model families.

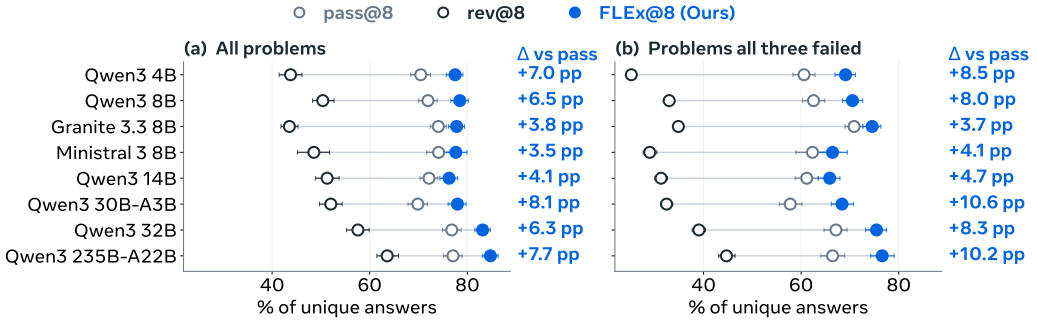

Figure 3: **Answer diversity among FLEx, independent sampling and sequential revision.** Given the same number of attempts, FLEx spends them on more distinct answers than independent sampling i.e., pass and sequential revision i.e., rev (left), and it still does so for problems all three methods cannot solve (right).

between each method and independent sampling. FLEx achieves the best performance on average compared to the other methods across 7/8 different models and sizes. More importantly, FLEx is the *only* method whose confidence interval relative to independent sampling excludes zero for every model and size, indicating consistent improvements over independent sampling across all evaluated model configurations. Averaged equally across the eight model configurations, FLEx improves mean solve@8 from 42.81% to 46.15%, a gain of 3.34 percentage points over independent sampling.

One exception is coding with Qwen3, for which the selected configuration uses the reverse FLEx orientation: we extrapolate from the failure-conditioned logits away from the clean-context logits. In § A.5, we analyze this behavior and show that this is because adding environment/execution feedback makes sequential sampling stronger than independent sampling. This does not happen when we replace execution feedback with just binary correct/incorrect feedback. Through this example, we show that a simple directional generalization of FLEx can still outperform feedback-augmented sequential sampling baselines.

We additionally show the cumulative *solve@8* curves for independent sampling (pass@k), sequential revision (rev@k), and FLEx, averaged across tasks for each model in Fig. 2. Across models, FLEx consistently improves over both baselines throughout the sampling trajectory. The gains are sustained as k increases. The FLEx curves remain separated from the baselines rather than crossing them or converging back toward them at larger sampling budgets. This suggests that the benefit of FLEx is persistent across attempts, rather than being concentrated only in the first few samples or diminishing as more samples are generated. We further demonstrate this point in § 3.4.

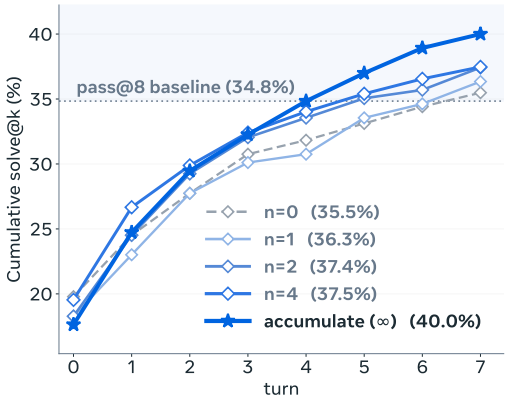

Figure 4: **Scaling the number of model's failures.** FLEx keeps improving performance as more prior failures are retained in the failure-conditioned context.

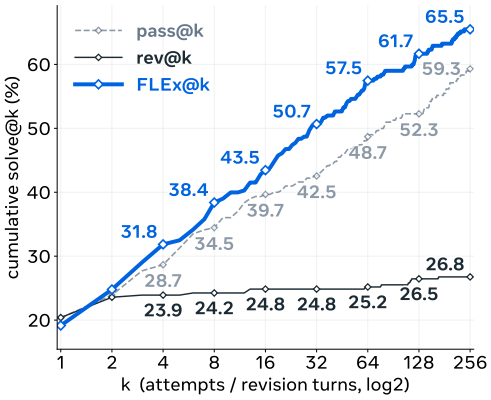

Figure 5: **Scaling the number of attempts to 256.** FLEx maintains a higher cumulative solve rate than independent sampling and sequential revision as the number of attempts increases.

To better understand where these gains come from, we examine the diversity of answers explored by each method in Fig. 3. Because answer novelty mechanically decreases with the number of attempts, we first match the attempt budget per problem across methods. We measure answer diversity as the fraction of attempts that produce an answer not previously seen in the same stream, pooling the four math evaluations (i.e., AIME2025, AIME2026, HMMT2026 and IMO-AnswerBench) across the same eight models used in our main experiments. For each stream of attempts, we evaluate both revision methods on their first $n = \min(k_{\text{FLEx}}, k_{\text{rev}})$ attempts and compare against the corresponding expected pass@n diversity under independent sampling. Under this matched comparison, FLEx produces previously unseen answers at a higher rate than independent sampling by 5.9 percentage points on average, with a positive 95% confidence interval for all 8 models, while sequential revision produces substantially fewer distinct answers. This effect is not explained by differences in solving behavior. Restricting to the subset of problems for which all three methods exhaust all eight attempts without ever finding a correct solution increases the FLEx advantage to 7.3 points, again with positive confidence intervals for all 8 models. Thus, even when all methods are equally unsuccessful and are compared after the same number of attempts, FLEx explores a broader set of candidate answers, consistent with its use of previous failures as a signal for where not to search again.

We provide qualitative examples by domain and their analysis in § B.

3.3 Scaling the number of failures to contrast

We next evaluate whether the gains of FLEx scale as we increase the number of failures in the failure-conditioned context for contrast. To experiment with this, we evaluate Qwen3-4B over AIME 2025, AIME 2026, and HMMT 2026 on Avg@5, and only keep n number (i.e., 0, 1, 2, 4) of failed attempts for up to k turns that result in solve@k. When $k > n$, we randomly compose n attempts to serve as the failure-conditioned context, and present evaluation results in Fig. 4. As shown, solve@8 increases monotonically with the amount of accumulated failure, where not accumulating failure at all (i.e., $n = 0$) reduces the performance close to the pass@8 baseline. This result highlights a key distinction between directly conditioning on failed reasoning and using it contrastively. Additional failed context may trap sequential revision in previously explored behaviors, whereas FLEx can extract an increasingly informative signal for where not to explore. Consequently, these failures become more useful as they accumulate when used through contrastive guidance.

3.4 Scaling the number of attempts

Beyond scaling the number of failures in a failure-conditioned context, we study whether the gain from FLEx persists if we scale solve@k at a much larger k. We evaluate Qwen3-4B on AIME 2025 and HMMT 2026 on Avg@5, and evaluate the model's solve@k for $k = 256$. To handle

Table 2: **Downstream performance (%) after rejection sampling and supervised fine-tuning on Qwen3-4B.** S/P denotes number of solved samples collected per prompt, where "One" retains one randomly selected correct trajectory and All retains every correct trajectory. Token-matched (i.e., "Token") models are evaluated at equal training-token budgets while Epoch-matched (i.e., "Epoch") models are evaluated at the end of 1 epoch. All evaluations use independent sampling. Bold indicates the higher accuracy within each matched Pass@8 versus FLEx@8 pair.

S/P	Collector	Budget	Steps	OpenR1		AIME25		AIME26		HMMT26		IMO-AnswerBench	
				pass@1	pass@8	pass@1	pass@8	pass@1	pass@8	pass@1	pass@8	pass@1	pass@8
–	Base	–	–	54.04	80.00	21.08	40.67	16.75	42.00	17.95	29.70	15.47	35.00
One	Pass@8	Token	40	56.71	80.67	24.75	38.67	16.50	40.67	19.32	30.30	17.78	39.00
One	FLEx@8	Token	40	**58.08**	**82.33**	**25.17**	**46.00**	**20.08**	**46.67**	**21.52**	**33.94**	**20.22**	**43.50**
One	Pass@8	Epoch	89	56.46	81.00	23.25	36.67	16.83	39.33	19.32	30.91	17.00	35.50
One	FLEx@8	Epoch	123	**60.46**	**84.00**	**24.50**	**46.00**	**22.17**	**45.33**	**21.14**	**31.52**	**20.38**	**44.75**
All	Pass@8	Token	40	55.88	**82.33**	22.17	38.67	17.67	42.00	17.58	27.27	17.66	39.25
All	FLEx@8	Token	40	**58.25**	82.00	**25.58**	**48.00**	**23.92**	**56.67**	**20.23**	**32.73**	**21.53**	**44.75**
All	Pass@8	Epoch	429	57.21	82.33	22.08	36.67	16.92	41.33	17.35	29.09	19.03	40.00
All	FLEx@8	Epoch	515	**58.79**	**84.00**	**22.75**	**44.67**	**21.50**	**50.00**	**20.23**	**34.55**	**20.62**	**44.25**

the unbounded growing context issue with methods based on sequential revision (i.e., rev@k and FLEx@k), we randomly compose a subset of 7 turns (i.e., failed attempts) as history for $k \geq 8$. Then, we compare the performance among pass@k, rev@k and FLEx@k as they scale to $k = 256$. We show the scaling curve in Fig. 5. As shown, from $k = 2$ onward, FLEx@k is strictly above pass@k, and the advantage widens through the mid-range. The largest gap of 9.4pp occurs at $k = 125$ and narrows slightly at the far end as pass@k catches up on the easy problems. FLEx reaches the coverage obtained by independent sampling at 256 attempts after approximately 104 attempts, demonstrating substantially better sample efficiency under an attempt-count budget.

4 FLEx IN POST-TRAINING

We next investigate whether FLEx's improved solution discovery translates into higher-quality training data for supervised finetuning via rejection sampling. We construct a training corpus of 20,000 mathematical problems from OpenR1-Math-220k (Hugging Face, 2025). Using Qwen3-4B, we collect 8 trajectories per problem through either independent sampling (pass@8) or FLEx (FLEx@8), retaining all 8 attempts without early stopping. We construct two datasets for each collection method: *one-per-problem*, containing one random correct trajectory per problem, and *all-samples*, containing all correct trajectories. We evaluate both at matched training-token budgets (i.e., 40 steps with 262,144 tokens per step) and after one epoch on a held-out OpenR1 subset and four out-of-distribution mathematical reasoning benchmarks: AIME 2025, AIME 2026, HMMT 2026, and IMO-AnswerBench (Luong et al., 2025) (more setup details in § A.8).

As shown in Tab. 2, FLEx-collected trajectories generally outperform independently sampled trajectories under matched training-token budgets, with gains across all evaluated benchmarks except OpenR1 pass@8 in the all-samples setting. The all-samples FLEx model improves average out-of-distribution (AIME25, AIME26, HMMT26, IMO-AnswerBench) pass@1 and pass@8 over the base model by +5.26 and +8.90 percentage points, respectively. In contrast, independently sampled training data yield no statistically significant improvement in pass@8.

Retaining all correct trajectories further improves FLEx-trained models, whereas independently sampled models benefit little from additional trajectories. The advantage persists when retaining only one correct trajectory per solved prompt, approximately controlling for dataset size. Moreover, FLEx achieves its strongest results at matched training-token budgets rather than after a full epoch, showing that the gains cannot be attributed simply to training on more tokens. Notably, in the one-epoch comparison, models trained on FLEx-collected data achieve better performance than those trained on independently sampled data despite requiring fewer than $0.5\times$ as many supervised-finetuning steps in the one-sample-per-prompt setting.

5 Related Works

Test-Time Scaling in Large Language Models. Test-time scaling (TTS) improves the reasoning and task accuracy of large language models (LLMs) by allocating additional inference compute to generating, evaluating, and refining candidate solutions (Wei et al., 2022; Wang et al., 2022; Snell et al., 2024; Muennighoff et al., 2025). Existing TTS methods broadly fall into two regimes: parallel and sequential. *Parallel methods* (Cobbe et al., 2021; Wang et al., 2022; Chen et al., 2023; Ning et al., 2024; Wu et al., 2025a) independently sample multiple reasoning traces without sharing context and aggregate their resulting answers. Although these methods achieve promising performance gains, their sampling efficiency is limited because different samples often concentrate on a small number of reasoning patterns. *Sequential approaches* (Madaan et al., 2023; Li et al., 2026) condition on the model's previous solutions and corresponding feedback to improve subsequent reasoning. These methods demonstrate that iterative refinement can improve generation quality, particularly when reliable external feedback or tools are available (Madaan et al., 2023; Shinn et al., 2023; Gou et al., 2024), although recent work has shown that iterative refinement often underperforms parallel methods under the same number of attempts (Liu et al., 2026b; Cheng et al., 2026; Gu et al., 2026). FLEx tackles TTS in a sequential way by treating sequential sampling as the branch to be contrasted.

Adaptive and Structured Reasoning Methods. Beyond fixed parallel or sequential strategies, adaptive and structured reasoning methods dynamically organize inference-time computation according to the structure and difficulty of each problem. Tree of Thoughts (Yao et al., 2023), Graph of Thoughts (Besta et al., 2024), and Language Agent Tree Search Zhou et al. (2023) explore intermediate reasoning states through structured search. Other approaches adapt the amount of inference-time computation by terminating sampling based on answer consistency, selecting search strategies with problem difficulty, and adjusting reasoning depth using verifier rewards (Aggarwal et al., 2023; Snell et al., 2024; Zhao et al., 2025). Recent methods further exploit parallelizable structures in reasoning traces by allowing models to determine when to branch into parallel threads and when to merge their results (Pan et al., 2025; Yang et al., 2026; Jin et al., 2025; Zheng et al., 2026; Lian et al., 2025). These methods extend from the two primitives of sampling, i.e., parallel and sequential, by introducing higher-level structures for organizing inference-time computation. In this work, we focus on the underlying sequential sampling primitive itself, leaving its integration with more structured or adaptive inference procedures to future work.

Contrastive and Guided Decoding. Contrastive and guided decoding methods steer generation at inference time by combining predictive distributions associated with different behaviors or conditions. Existing methods primarily differ in how they construct this contrast. Model-based approaches combine expert and anti-expert models or contrast stronger and weaker language models (Liu et al., 2021; Li et al., 2023; O'Brien & Lewis, 2023), while layer-based methods contrast predictions from different model layers (Chuang et al., 2024). Condition-based approaches instead contrast the same model under conditional and unconditional context (Ho & Salimans, 2022; Sanchez et al., 2023), with and without external evidence (Zhao et al., 2024; Shi et al., 2024), or under positive and negative prompts (Zhong et al., 2024; Peng et al., 2025; Bi et al., 2026). These methods construct the contrast *a priori*, independently of the model's search trajectory. FLEx instead constructs its contrast *from the search itself*: each verified failure and its feedback define a new contrastive condition, whose induced predictive shift is used to steer the next attempt.

6 Conclusion

We introduced Failure Logit Extrapolation (FLEx), a training-free decoding method that uses the contrast between clean-context and failure-conditioned predictions to guide subsequent attempts. Across models, scales, and reasoning domains, FLEx improves solution discovery while maintaining greater exploration than independent sampling and sequential revision. Our results show that failed attempts can provide a useful test-time search signal when used contrastively, and that richer feedback can further change how this signal should be exploited. Finally, FLEx-collected trajectories improve downstream rejection-sampling training, suggesting that better test-time exploration can also translate into better post-training data. We believe an interesting extension to this work is to check if FLEx can be used for rollout generation in reinforcement learning and benchmark FLEx on more model families including frontier models.

AI USE STATEMENT

In this work, we used generative AI tools for refining but not proposing hypotheses/methods, implementing methods, cleaning and reformatting the dataset, supporting qualitative and thematic data analysis, creating result figures, editing software code, drafting parts of the appendix, analyzing existing literature, identifying relevant literature, and editing the main paper for readability.

We have not used generative AI tools for proposing hypotheses/methods, designing or providing feedback on research methodology or experiments, interpreting results, designing the method figure, and drafting the main paper.

Finally, generating synthetic datasets, helping develop theoretical models or conceptual frameworks, formulating mathematical claims, providing critical ingredients for proving mathematical claims, and assisting in writing proofs are not applicable to this work.

We have reviewed all AI-assisted work by manually reviewing the literature suggested by AI ourselves for relevance and inspected the correctness of the code that AI generated. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We provide the information needed to reproduce our experiments in Appendix A. Specifically, we document the evaluated model checkpoints and parallelism, benchmarks and verifiers, verifier feedback formats, exact prompts, decoding settings and generation budgets, FLEx hyperparameters and their selection protocol, baseline implementations, and statistical procedures. Algorithm 1 specifies the FLEx decoding procedure. For the post-training experiments, we additionally report the data-collection procedure, dataset construction, supervised fine-tuning configuration, and evaluation protocol. All methods are evaluated with matched prompts, sampling settings, verifiers, feedback, and attempt budgets, and are implemented through uniform vLLM-based decoding where applicable.

REFERENCES

Pranjal Aggarwal, Aman Madaan, Yiming Yang, et al. Let's sample step by step: Adaptive-consistency for efficient reasoning and coding with llms. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 12375–12396, 2023.

Mislav Balunovic, Jasper Dekoninck, Ivo Petrov, Nikola Jovanović, and Martin Vechev. Matharena: Evaluating llms on uncontaminated math competitions. *Advances in Neural Information Processing Systems*, 38, 2026a.

Mislav Balunovic, Jasper Dekoninck, Ivo Petrov, Nikola Jovanović, and Martin Vechev. Matharena: Evaluating LLMs on uncontaminated math competitions. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2026b. URL https://openreview.net/forum?id=y0zL9IZxZ7.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, et al. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pp. 17682–17690, 2024.

Baolong Bi, Yuyao Ge, Shenghua Liu, Yuchen He, Siqian Tong, Lizhe Chen, Lingrui Mei, Zehao Li, Yiwei Wang, Yujun Cai, Ming-Hsuan Yang, and Xueqi Cheng. Promptcd: Test-time behavior enhancement via polarity-prompt contrastive decoding, 2026. URL https://arxiv.org/abs/2602.20696.

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, et al. Mle-bench: Evaluating machine learning agents on machine learning engineering. In *International Conference on Learning Representations*, volume 2025, pp. 50466–50494, 2025.

Xinyun Chen, Renat Aksitov, Uri Alon, Jie Ren, Kefan Xiao, Pengcheng Yin, Sushant Prakash, Charles Sutton, Xuezhi Wang, and Denny Zhou. Universal self-consistency for large language model generation. *arXiv preprint arXiv:2311.17311*, 2023.

Yun Cheng, Xingyu Zhu, Haoyu Zhao, and Sanjeev Arora. Contextual drag: How errors in the context affect llm reasoning. *arXiv preprint arXiv:2602.04288*, 2026.

Yung-Sung Chuang, Yujia Xie, Hongyin Luo, Yoon Kim, James R Glass, and Pengcheng He. Dola: Decoding by contrasting layers improves factuality in large language models. In *International Conference on Learning Representations*, volume 2024, pp. 54158–54183, 2024.

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.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yujiu Yang, Nan Duan, Weizhu Chen, et al. Critic: Large language models can self-correct with tool-interactive critiquing. In *International Conference on Learning Representations*, volume 2024, pp. 57734–57811, 2024.

IBM Granite Team. Granite 3.0 language models, October 2024. URL https://github.com/ibm-granite/granite-3.0-language-models/.

Xiangming Gu, Soham De, Larisa Markeeva, Petar Veličković, and Razvan Pascanu. Understanding performance gap between parallel and sequential sampling in large reasoning models. *arXiv preprint arXiv:2604.05868*, 2026.

Caglar Gulcehre, Tom Le Paine, Srivatsan Srinivasan, Ksenia Konyushkova, Lotte Weerts, Abhishek Sharma, Aditya Siddhant, Alex Ahern, Miaosen Wang, Chenjie Gu, et al. Reinforced self-training (rest) for language modeling. *arXiv preprint arXiv:2308.08998*, 2023.

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. *arXiv preprint arXiv:2103.03874*, 2021.

Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. *arXiv preprint arXiv:2207.12598*, 2022.

Ailin Huang, Ang Li, Aobo Kong, Bin Wang, Binxing Jiao, Bo Dong, Bojun Wang, Boyu Chen, Brian Li, Buyun Ma, et al. Step 3.5 flash: Open frontier-level intelligence with 11b active parameters. *arXiv preprint arXiv:2602.10604*, 2026.

Hugging Face. Open r1: A fully open reproduction of deepseek-r1, January 2025. URL https://github.com/huggingface/open-r1.

Naman Jain, 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. In *International Conference on Learning Representations*, volume 2025, pp. 58791–58831, 2025.

Tian Jin, Ellie Y Cheng, Zack Ankner, Nikunj Saunshi, Blake M Elias, Amir Yazdanbakhsh, Jonathan Ragan-Kelley, Suvinay Subramanian, and Michael Carbin. Learning to keep a promise: Scaling language model decoding parallelism with learned asynchronous decoding. *arXiv preprint arXiv:2502.11517*, 2025.

Shijia Kang and Muhan Zhang. The road less traveled: Enhancing exploration in llms via sequential sampling. *arXiv preprint arXiv:2510.15502*, 2025.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pp. 611–626, 2023.

Robert Lange, Yuki Imajuku, and Edoardo Cetin. Shinkaevolve: Towards open-ended and sample-efficient program evolution. In *International Conference on Learning Representations*, volume 2026, pp. 74026–74078, 2026.

Dacheng Li, Fangzhou Zhao, Shiyi Cao, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. KTPO: K-step test-time policy optimization for long- horizon discovery. In *Third Conference on Language Modeling*, 2026. URL https://openreview.net/forum?id=t2MIMXJq3W.

Xiang Lisa Li, Ari Holtzman, Daniel Fried, Percy Liang, Jason Eisner, Tatsunori B Hashimoto, Luke Zettlemoyer, and Mike Lewis. Contrastive decoding: Open-ended text generation as optimization. In *Proceedings of the 61st annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 12286–12312, 2023.

Long Lian, Sida Wang, Felix Juefei-Xu, Tsu-Jui Fu, Xiuyu Li, Adam Yala, Trevor Darrell, Alane Suhr, Yuandong Tian, and Xi Victoria Lin. Threadweaver: Adaptive threading for efficient parallel reasoning in language models. *arXiv preprint arXiv:2512.07843*, 2025.

Alexander H Liu, Kartik Khandelwal, Sandeep Subramanian, Victor Jouault, Abhinav Rastogi, Adrien Sadé, Alan Jeffares, Albert Jiang, Alexandre Cahill, Alexandre Gavaudan, et al. Ministral 3. *arXiv preprint arXiv:2601.08584*, 2026a.

Alisa Liu, Maarten Sap, Ximing Lu, Swabha Swayamdipta, Chandra Bhagavatula, Noah A. Smith, and Yejin Choi. DExperts: Decoding-time controlled text generation with experts and anti-experts. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli (eds.), *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 6691–6706, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.522. URL https://aclanthology.org/2021.acl-long.522/.

Kaiyuan Liu, Qiuyang Mang, Bo Peng, Wenhao Chai, Hanchen Li, Shreyas Pimpalgaonkar, Luke Zettlemoyer, Alex Dimakis, and Alvin Cheung. When agents slow down: Understanding llm agents' test-time strategies via elo-per-token analysis. *arXiv preprint arXiv:2609.15309*, 2026b.

Thang Luong, Dawsen Hwang, Hoang H. Nguyen, Golnaz Ghiasi, Yuri Chervonyi, Insuk Seo, Junsu Kim, Garrett Bingham, Jonathan Lee, Swaroop Mishra, Alex Zhai, Clara Huiyi Hu, Henryk Michalewski, Jimin Kim, Jeonghyun Ahn, Junhwi Bae, Xingyou Song, Trieu H. Trinh, Quoc V. Le, and Junehyuk Jung. Towards robust mathematical reasoning. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, 2025. URL https://aclanthology.org/2025.emnlp-main.1794/.

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

Qiuyang Mang, Wenhao Chai, Zhifei Li, Huanzhi Mao, Shang Zhou, Alexander Du, Hanchen Li, Shu Liu, Edwin Chen, Yichuan Wang, et al. Frontiercs: Evolving challenges for evolving intelligence. *arXiv preprint arXiv:2512.15699*, 2025.

Mike Merrill, Alexander Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Shin, Thomas Walshe, E Kelly Buchanan, et al. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. In *International Conference on Learning Representations*, volume 2026, pp. 40903–40986, 2026.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori B Hashimoto. s1: Simple test-time scaling. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 20286–20332, 2025.

Xuefei Ning, Zinan Lin, Zixuan Zhou, Zifu Wang, Huazhong Yang, and Yu Wang. Skeleton-of-thought: Prompting llms for efficient parallel generation. In *International Conference on Learning Representations*, volume 2024, pp. 917–967, 2024.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco JR Ruiz, Abbas Mehrabian, et al. Alphaevolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

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

Anne Ouyang, Simon Guo, Simran Arora, Alex L Zhang, William Hu, Christopher Ré, and Azalia Mirhoseini. Kernelbench: Can llms write efficient gpu kernels? *arXiv preprint arXiv:2502.10517*, 2025.

Jiayi Pan, Xiuyu Li, Long Lian, Charlie Snell, Yifei Zhou, Adam Yala, Trevor Darrell, Kurt Keutzer, and Alane Suhr. Learning adaptive parallel reasoning with language models. *arXiv preprint arXiv:2504.15466*, 2025.

Rongcan Pei, Zhepei Wei, Shuyao Xu, Xinyu Zhu, Wei-Lin Chen, and Yu Meng. Negative self-distillation: Learning to reason by avoiding flaws, 2026. URL https://arxiv.org/abs/2609.11699.

Keqin Peng, Liang Ding, Yuanxin Ouyang, Meng Fang, Yancheng Yuan, and Dacheng Tao. Enhancing input-label mapping in in-context learning with contrastive decoding. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 997–1004, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-252-7. doi: 10.18653/v1/2025.acl-short.77. URL https://aclanthology.org/2025.acl-short.77/.

Long Phan, Alice Gatti, Ziwen Han, Nathaniel Li, Josephina Hu, Hugh Zhang, Chen Bo Calvin Zhang, Mohamed Shaaban, John Ling, Sean Shi, et al. Humanity's last exam. *arXiv preprint arXiv:2501.14249*, 2025.

Valentina Pyatkin, Saumya Malik, Victoria Graf, Hamish Ivison, Shengyi Huang, Pradeep Dasigi, Nathan Lambert, and Hannaneh Hajishirzi. Generalizing verifiable instruction following, 2025.

Guillaume Sanchez, Honglu Fan, Alexander Spangher, Elad Levi, Pawan Sasanka Ammanamanchi, and Stella Biderman. Stay on topic with classifier-free guidance, 2023. URL https://arxiv.org/abs/2306.17806.

Weijia Shi, Xiaochuang Han, Mike Lewis, Yulia Tsvetkov, Luke Zettlemoyer, and Wen-tau Yih. Trusting your evidence: Hallucinate less with context-aware decoding. In Kevin Duh, Helena Gomez, and Steven Bethard (eds.), *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 2: Short Papers)*, pp. 783–791, Mexico City, Mexico, June 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.naacl-short.69. URL https://aclanthology.org/2024.naacl-short.69/.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Adi Simhi, Fazl Barez, Martin Tutek, Yonatan Belinkov, and Shay B Cohen. Old habits die hard: How conversational history geometrically traps llms. *arXiv preprint arXiv:2603.03308*, 2026.

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

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, et al. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*, 2025.

Tianchun Wang, Zichuan Liu, Yuanzhou Chen, Jonathan Light, Weiyang Liu, Haifeng Chen, Xiang Zhang, and Wei Cheng. On the effect of sampling diversity in scaling llm inference. *arXiv preprint arXiv:2502.11027*, 2025.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Chen Henry Wu, Sachin Goyal, and Aditi Raghunathan. Mode-conditioning unlocks superior test-time scaling. *arXiv preprint arXiv:2512.01127*, 2025a.

Fang Wu, Weihao Xuan, Heli Qi, Ximing Lu, Aaron Tu, Li Erran Li, and Yejin Choi. Deepsearch: Overcome the bottleneck of reinforcement learning with verifiable rewards via monte carlo tree search. *arXiv preprint arXiv:2509.25454*, 2025b.

Wei Xiong, Jiarui Yao, Yuhui Xu, Bo Pang, Lei Wang, Doyen Sahoo, Junnan Li, Nan Jiang, Tong Zhang, Caiming Xiong, et al. A minimalist approach to llm reasoning: from rejection sampling to reinforce. *arXiv preprint arXiv:2504.11343*, 2025.

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.

Xinyu Yang, Yuwei An, Hongyi Liu, Tianqi Chen, and Beidi Chen. Multiverse: Your language models secretly decide how to parallelize and merge generation. *Advances in Neural Information Processing Systems*, 38:100773–100804, 2026.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023.

Dan Zhang, Sining Zhoubian, Ziniu Hu, Yisong Yue, Yuxiao Dong, and Jie Tang. Rest-mcts*: Llm self-training via process reward guided tree search. *Advances in Neural Information Processing Systems*, 37:64735–64772, 2024.

Zheng Zhao, Emilio Monti, Jens Lehmann, and Haytham Assem. Enhancing contextual understanding in large language models through contrastive decoding. In Kevin Duh, Helena Gomez, and Steven Bethard (eds.), *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 4225–4237, Mexico City, Mexico, June 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.naacl-long.237. URL https://aclanthology.org/2024.naacl-long.237/.

Zhengyi Zhao, Shubo Zhang, Zezhong Wang, Huimin Wang, Yutian Zhao, Bin Liang, Yefeng Zheng, Binyang Li, Kam-Fai Wong, and Xian Wu. T2: An adaptive test-time scaling strategy for contextual question answering. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 3731–3756, 2025.

Tong Zheng, Hongming Zhang, Wenhao Yu, Xiaoyang Wang, He Xing, Runpeng Dai, Rui Liu, Huiwen Bao, Chengsong Huang, Heng Huang, et al. Parallel-r1: Towards parallel thinking via reinforcement learning. In *International Conference on Learning Representations*, volume 2026, pp. 121144–121166, 2026.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, and Zheyan Luo. Llamafactory: Unified efficient fine-tuning of 100+ language models. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 3: system demonstrations)*, pp. 400–410, 2024.

Qihuang Zhong, Liang Ding, Juhua Liu, Bo Du, and Dacheng Tao. ROSE doesn't do that: Boosting the safety of instruction-tuned large language models with reverse prompt contrastive decoding. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 13721–13736, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.814. URL https://aclanthology.org/2024.findings-acl.814/.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning acting and planning in language models. *arXiv preprint arXiv:2310.04406*, 2023.

Minhui Zhu, Minyang Tian, Xiaocheng Yang, Tianci Zhou, Lifan Yuan, Penghao Zhu, Eli Chertkov, Shengyan Liu, Yufeng Du, Ziming Ji, et al. Probing the critical point (critpt) of ai reasoning: a frontier physics research benchmark. *arXiv preprint arXiv:2509.26574*, 2025.

A Prompts, Hyperparameters and Setup for Experiments

A.1 Models

All models are instruction-tuned checkpoints evaluated in their default instruct mode, served with vLLM in `bfloat16`. Table 3 lists the eight models and the tensor parallelism used for each. The same weights serve both the clean and the failure-conditioned context: FLEx never loads a second model, so its memory footprint of the weight is identical to that of the baselines.

Model	Family	Params (active)	TP
Qwen3 4B	Qwen3	4B	1
Qwen3 8B	Qwen3	8B	1
Qwen3 14B	Qwen3	14B	2
Qwen3 32B	Qwen3	32B	4
Qwen3 30B-A3B	Qwen3 (MoE)	30B (3B)	4
Qwen3 235B-A22B	Qwen3 (MoE)	235B (22B)	8
Granite 3.3 8B	Granite 3.3	8B	1
Ministral 3 8B	Ministral 3	8B	1

Table 3: **Models evaluated.** Parenthesised counts are active parameters per token for the mixture-of-experts models.

A.2 Benchmarks, verifiers and feedback

We use six benchmarks in three domains, chosen because each admits a fully automatic and objective verifier and the three domains are commonly used for reasoning evaluations (Yang et al., 2025; Huang et al., 2026; Team et al., 2025). Tab. 4 summarizes them.

Benchmark	Domain	Problems	Seeds	Verifier / feedback signal
AIME 2025	Math	30	5	symbolic answer equivalence; correctness only
AIME 2026	Math	30	5	symbolic answer equivalence; correctness only
HMMT Feb 2026	Math	33	5	symbolic answer equivalence; correctness only
IMO-AnswerBench	Math	400	1	symbolic answer equivalence; correctness only
IFBench	IF	300	1	per-constraint checkers; list of violated constraints
LiveCodeBench v6	Code	1055	1	test-case execution; failing input/output or error

Table 4: **Benchmarks**. "Seeds" is the number of independent sampling seeds averaged over; the three competition-math sets are reported as the mean over five seeds (reported as avg@5 in the main text), the remaining three as a single seed.

Answer equivalence on math. Math answers are graded with HuggingFace's `math-verify` (v0.9.0), parsing both sides with its LaTeX and plain-expression extractors and comparing with its symbolic `verify`. A prediction is therefore scored correct if it is *symbolically* equivalent to the reference. Grading runs in a worker pool with a per-call timeout, since a small number of adversarial expressions can make symbolic comparison hang.

Both the reference and the prediction are wrapped in `$...$` identically before parsing, as `math-verify`'s LaTeX extractor only fires on delimited input, so passing a bare prediction against a delimited reference silently extracts nothing and returns *false* for a character-identical answer — e.g. reference `$\frac{1}{2}$` against prediction `\dfrac{1}{2}`. The issue is confined to non-integer references, of which IMO-AnswerBench has $173/400$ and HMMT Feb 2026 $17/33$; AIME answers are integers in $[0, 999]$ and parse identically either way, so those two columns are provably unaffected. All reported math numbers use the symmetric wrapping.

Verifiers on the other two domains. Instruction following uses IFBench's own per-constraint checkers, which return the set of violated constraint identifiers. Coding uses LiveCodeBench's execution harness, which runs the extracted program against the problem's test cases in a sandboxed subprocess and returns the first failing case or the interpreter error.

Feedback content. The feedback appended after a failed attempt is the weakest signal the domain naturally provides, and it differs by domain because the domains differ in what a verifier can cheaply report:

- **Math**: correctness only, naming the rejected answer.

```
Your previous answer of '{answer}' is incorrect. Please
try again, reasoning step by step, and put your final
answer in \boxed{}.
```

- **Instruction following**: the identifiers of the constraints that were violated.

```
Your response did not follow the following constraint(s):
{ids}. Please revise the response to satisfy all
constraints and try again.
```

- **Coding**: the execution outcome. For a wrong answer, the failing input, the produced output and the expected output; for a compilation error or timeout, the interpreter message. Long fields are truncated to 400 characters so the failure-conditioned context stays bounded:

```
The above code is incorrect and got a wrong answer.
Input: {inputs}
Generated Output: {output}
Expected: {expected}
```

Note that the math feedback is strictly less informative than the other two. It says only *that* the attempt failed. That FLEx still improves over independent sampling on math indicates the gain does not depend on the feedback carrying diagnostic content. The contrast is driven by conditioning on the failed trajectory itself, not by the model reading a useful critique.

A.3 Prompts

Every method in the paper uses the same system prompt, `"You are a helpful assistant."`, and the same first user turn: the problem statement followed by a short answer-format instruction. The format instruction is the only part that varies by domain, and it is held fixed across all methods so that the clean and failure-conditioned contexts differ only in the presence of prior attempts.

- **Math** (all four benchmarks): `" Please reason step by step, and put your final answer within \boxed{}."`
- **Coding**: `" Put your final complete solution in the last ```python fenced block of your response, and any other code snippets in plain ``` fenced blocks."`
- **Instruction following**: no suffix. IFBench prompts already carry their own constraints, and appending a format instruction would introduce an instruction the benchmark does not score.

The clean context H_c is exactly this two-message prompt and is never extended. The failure-conditioned context H_f starts identical and grows by two messages per failure: the assistant's failed attempt, then a user turn holding the feedback above.

A.4 Decoding and generation budget

Sampling is identical for every method and every model: temperature 0.7, top-p 1.0, and top-k disabled. Generation length and context window vary by domain, because the failure-conditioned context grows with each attempt and must still fit after eight turns (Tab. 5). Qwen3 models use YaRN RoPE scaling to reach the required context length; Granite 3.3 and Ministral 3 have sufficient native context and use none.

A single decoding implementation serves all methods. The baselines that do not require a contrast are run through the same code path with the contrast disabled rather than through a separate sampler. We implement the algorithm within the vLLM framework (Kwon et al., 2023), and all experiments are done within it. We commit to releasing the implementation.

Domain	Max new tokens	Context window	RoPE scaling (Qwen3 only)
Math (all four)	16,384	163,840	4×
Instruction following	4,096	32,768	2×
Coding	8,192	81,920	2×

Table 5: Generation budget per domain, shared by all methods. The context window is sized to accommodate the observed eight-attempt histories without truncation.

A.5 FLEx Hyperparameters

FLEx has two continuous hyperparameters: the guidance strength λ (O'Brien & Lewis, 2023) and the plausibility threshold α (Li et al., 2023), which retains only tokens whose clean-context logit is within $\log \alpha$ of the clean-context maximum. Tab. 6 reports the settings used for every reported result.

Family	Math (4 evals)	Inst. following	Coding
Qwen3 (all six sizes)	$\lambda = 0.8,\ \alpha = 10^{-3}$	$\lambda = 0.8,\ \alpha = 10^{-5}$	$\lambda = 0.2,\ \alpha = 10^{-6\dagger}$
Granite 3.3 8B	$\lambda = 0.2,\ \alpha = 10^{-4}$	$\lambda = 0.8,\ \alpha = 10^{-5}$	$\lambda = 0.1,\ \alpha = 10^{-4}$
Ministral 3 8B	$\lambda = 0.2,\ \alpha = 10^{-8}$	$\lambda = 0.2,\ \alpha = 10^{-2}$	$\lambda = 0.2,\ \alpha = 10^{-4}$

Table 6: **FLEx hyperparameters.** A single setting is shared by all six Qwen3 sizes within each domain; no per-size tuning is performed. †Qwen3 coding uses the reverse orientation described below.

Selection protocol. λ and α are selected from a small grid ($\lambda \in \{0.1, 0.2, 0.4, 0.6, 0.8\}$, $\alpha \in \{10^{-2}, \ldots, 10^{-8}\}$) for each model family and domain. We jointly select λ, α, and the extrapolation orientation on a held-out validation set using solve@8 as the selection metric. The selected configuration is then frozen before evaluation on the reported test benchmarks. For Qwen3, a single configuration is shared across all six model sizes within each domain, so the scale sweep in Fig. 2 involves no per-size tuning.

Coding uses a swapped contrast for Qwen3. For Qwen3 coding, the selected configuration uses the reverse FLEx orientation. The failure-conditioned logits become the base and the clean-context logits become the subtracted term, $z = (1+\lambda)z_f - \lambda z_c$, while the plausibility mask remains gated on z_c. This corresponds to extrapolating away from the failure-history-free clean-context prediction rather than away from the failure-conditioned prediction.

This behavior coincides with a qualitative difference between Qwen3 coding and the settings in which the default orientation is selected. On LiveCodeBench, sequential revision outperforms independent sampling across the Qwen3 scale ladder (e.g. 54.69 vs. 54.60 at 8B and 70.90 vs. 68.63 at 235B-A22B), indicating that the execution feedback makes conditioning on the failure history useful for this model family. In contrast, Granite 3.3 and Ministral 3 retain the default orientation on coding; for these models, sequential revision remains weaker than independent sampling (17.35 vs. 26.07 and 54.03 vs. 59.24, respectively).

The coding exception is associated with feedback richness rather than the domain itself. We test this interpretation by reducing the coding verifier feedback to the same type of binary correctness signal used in the mathematics evaluations. Specifically, we replace the full execution report (failing input, produced output, and expected output) with the message `"The above Python code is incorrect. Please try again."`, while leaving the remaining evaluation procedure unchanged. Tab. 7 reports the resulting sequential-revision performance across the Qwen3 scale ladder.

With execution feedback, sequential revision outperforms independent sampling on all six Qwen3 sizes. With binary correctness feedback, it does so on only one of the six, losing between 1.6 and 7.6pp on the other five, with a mean swing of −5.7pp. Coding is therefore not intrinsically a domain

Model	pass@8	rev@8		Δ vs. pass@8	
		execution fb.	binary fb.	execution	binary
Qwen3 4B	48.53	49.67	42.75	+1.14	−5.78
Qwen3 8B	54.60	54.69	52.99	+0.09	−1.61
Qwen3 14B	58.48	60.19	53.36	+1.71	−5.12
Qwen3 32B	62.75	64.55	55.17	+1.80	−7.58
Qwen3 30B-A3B	57.73	59.91	58.58	+2.18	+0.85
Qwen3 235B-A22B	68.63	70.90	62.94	+2.27	−5.69
models where revision beats independent sampling				6/6	1/6

Table 7: **Sequential revision on LiveCodeBench v6 (n=1055) under the full execution report versus a single correctness bit.** Independent sampling is unaffected by the feedback change because it never observes feedback, so the pass@8 column serves as a fixed reference for both conditions. Everything except the content of the feedback turn is held constant.

in which sequential revision is stronger. When the feedback is reduced to the binary signal used in the mathematics evaluations, the advantage of conditioning on the failure history largely disappears.

This result supports interpreting the selected orientation as a function of feedback informativeness rather than task type. In particular, sufficiently informative verifier feedback can make the failure-conditioned prediction a useful base for extrapolation, consistent with the reverse orientation selected for Qwen3 coding.

A.6 Baselines

All five comparison methods use the same prompts, sampling settings, generation budget, verifier, feedback, random seeds, and attempt budget described above. They differ only in whether and how the clean-context and failure-conditioned predictions are used. Together, these baselines isolate two design choices in FLEx: *what* provides the contrast and *how* that contrast is combined with the model's prediction.

- **Independent sampling (pass@k).** We draw k i.i.d. samples from the clean context. No failure history or feedback is provided, and no contrastive signal is used.
- **Sequential revision (rev@k).** We sample directly from the failure-conditioned context, which accumulates every previous failed attempt and its corresponding feedback. No contrast is applied. This is the standard self-refinement setting.
- **Interpolation.** We combine the two predictions as $z = \frac{1}{2}z_c + \frac{1}{2}z_f$. This uses the same two-stream construction as FLEx but forms a convex combination rather than extrapolating beyond either prediction. It therefore tests whether the benefit comes simply from mixing the clean and failure-conditioned views, rather than from extrapolation.
- **Contrastive decoding, fixed negative prompt.** We use $z = z_c - z_f$, where z_f is produced from a second copy of the clean prompt with a fixed negative instruction appended to the user turn:

  ```
  You are a careless student.  Use quick, intuitive
  reasoning, take shortcuts whenever possible, and do not
  carefully verify intermediate steps or the final answer.
  ```

  This context is constructed once and never updated: it contains no previous attempt, no verifier feedback, and no information specific to the model's failures on the current problem. It therefore ablates the use of *online, instance-specific failures*. If a generic negative condition were sufficient to provide the useful contrast, this baseline should perform similarly to contrastive decoding with online failures.
- **Contrastive decoding, online failures.** We again use $z = z_c - z_f$, but now z_f is obtained from the same accumulating failure-conditioned context used by FLEx. This keeps the contrast source fixed while replacing FLEx's extrapolation with direct logit subtraction, and therefore serves as an ablation of the *extrapolation operator*.

Plausibility threshold for subtractive contrastive decoding. Both contrastive-decoding baselines use $\alpha = 0.1$ rather than inheriting the substantially smaller α values used by FLEx. The plausibility mask plays a different role under pure subtraction. FLEx retains the clean-context prediction as an explicit base, $z_{\text{FLEx}} = z_c + \lambda(z_c - z_f)$, while interpolation forms a convex combination of z_c and z_f. In contrast, subtractive decoding uses the pure difference $z_c - z_f$ and therefore has no base logit vector anchoring the resulting scores to the model's original token distribution. A token can consequently receive a large contrastive score even when it is unlikely under both contexts, provided that one context assigns it a less negative logit than the other.

The plausibility mask is therefore particularly important for subtractive decoding, where it restricts sampling to tokens that remain sufficiently plausible under the clean-context distribution. Reusing FLEx's much smaller α would make this constraint substantially weaker and can cause the subtractive baseline to fail for reasons unrelated to the comparison between subtraction and extrapolation. Empirically, at $\alpha = 10^{-4}$ the subtractive baselines frequently fail to terminate properly. We therefore use $\alpha = 0.1$, which is also the value used by Li et al. (2023), from whom this baseline is adapted.

A.7 Metric and statistical protocol

Averaging. For the three competition-math benchmarks we run five seeds and report the mean. IMO-AnswerBench, IFBench and LiveCodeBench are run at a single seed; their problem counts (400, 300, 1055) make the per-benchmark standard error comparable to the five-seed average of a 30-problem set. The "Avg." column is the unweighted mean of the six benchmark values, so each task counts once regardless of size. Weighting by problem count would let LiveCodeBench and IMO-AnswerBench supply 79% of a number the table presents as an average over six equally-weighted tasks.

Confidence intervals. The final column of Tab. 1 is a 95% interval on the difference from independent sampling, paired *per problem* within each benchmark and then combined across benchmarks with equal task weight: $D = \frac{1}{E}\sum_e d_e$ and $\mathrm{Var}(D) = \frac{1}{E^2}\sum_e \mathrm{Var}(d_e)$, where d_e is the paired mean difference on benchmark e.

A.8 Rejection sampling and supervised fine-tuning setup

Collection. 20,000 problems from OpenR1-Math-220k, Qwen3 4B, eight trajectories per problem, generated either by independent sampling or by FLEx ($\lambda = 0.8$, $\alpha = 10^{-3}$). We do not stop FLEx when the correct sample is generated. Instead, we let it continue so each method spends exactly eight generations per problem, and any difference in dataset size reflects how many of those generations were correct and the length of these generations.

Datasets. ONE keeps one uniformly-chosen correct trajectory per solved problem, approximately controlling for dataset size; ALL keeps every correct trajectory.

Training. Full-parameter supervised fine-tuning of Qwen3 4B in `bfloat16` with a global batch size of 16. Learning rate is set to 1×10^{-5} with cosine schedule and 3% warmup, weight decay of 0.01, sequence length of 16,384 with example packing with DeepSpeed ZeRO-2, using LlamaFactory (Zheng et al., 2024). With packing, every optimizer step consumes exactly $16 \times 16{,}384 = 262{,}144$ tokens, so step 40 is 10.5M tokens for both methods. This is the token-matched checkpoint reported as "Token". The "Epoch" rows are the end of one epoch and are *not* compute-matched: the FLEx corpus needs 123 steps against independent sampling's 89 in the ONE setting, because FLEx solutions are longer. Both are reported for exactly that reason, and the token-matched pair is the controlled comparison.

Evaluation. Identical for the base model and every checkpoint: eight independent samples per problem at temperature 0.7 under the math settings of Tab. 5, from which pass@1 (mean per-sample accuracy) and pass@8 (solved by any of the eight) are computed. No checkpoint is evaluated with FLEx, as the question is whether FLEx-collected *data* yields a better model, so the decoding method must not differ at evaluation time.

A.9 WHY BASELINE LEVELS DIFFER ACROSS SETTINGS

Absolute scores should not be compared across the three experimental settings, because they use different context configurations. The only differences are the RoPE scaling configuration and the per-turn token budget.

Setting	RoPE	`max_tokens`	Qwen3 4B pass@8, IMO-AB
Main experiments (Tab. 1)	YaRN, factor 4	16,384	42.0
History ablation (Fig. 4)	YaRN, factor 2	8,192	—
Rejection sampling (Tab. 2)	native	32,768	35.0

We deliberately disable YaRN in the rejection-sampling study because static YaRN affects the compared models differently when their output-length distributions differ. In our measurements, enabling YaRN leaves the base model essentially unchanged ($17.63 \rightarrow 17.69$ pass@1), while improving the FLEx-trained student from 18.33 to 20.22 as its truncation rate falls from 12.74% to 3.71%. Since the FLEx-trained student also tends to generate longer outputs, retaining YaRN would introduce a configuration-dependent advantage for the model whose performance is being evaluated.

The resulting shift in absolute performance is systematic rather than sampling noise. Holding the checkpoint, benchmark, and prompt fixed while varying only the context configuration, Qwen3 4B obtains 42.0 versus 35.0 pass@8 on IMO-AnswerBench, a paired difference of $+7.00\%$ (95% CI $[+3.25, +10.75]$ over the same 400 problems). On the smaller 30–33 problem competition benchmarks, the two configurations are statistically indistinguishable, so this difference is only reliably resolved on the larger benchmark.

These configuration-dependent shifts do not affect the validity of our within-setting comparisons. Within each experimental setting, all methods, including the corresponding baseline, use the same RoPE configuration and per-turn token budget, and the baseline is re-evaluated under that configuration rather than transferred from another setting. Accordingly, every reported method delta is a within-setting contrast between methods evaluated under the same context configuration. Absolute scores, however, should not be compared across settings.

B QUALITATIVE EXAMPLES

The mechanism FLEx targets is easiest to see on a single problem. This appendix shows two, then the raw failure-signal text for each domain.

B.1 THE FAILURE TRAP AND THE ESCAPE

Table 8 shows the eight answers produced by each method on one AIME 2025 problem (Qwen3 4B, seed 42), a plane-geometry question whose reference answer is 588.

Method	Answer at attempt 1 . . . 8								Solved
Independent sampling	1274	1684	986	1078	1896	390	1028	1274	no
Sequential revision	1274	1274	1274	1274	1274	1274	1274	1274	no
FLEx	924	**588**	—	—	—	—	—	—	**yes**

Table 8: Answer streams on AIME 2025 problem 2 (reference answer 588), Qwen3 4B. Sequential revision is told after every attempt that 1274 is wrong and re-derives 1274 seven more times. FLEx stops after two attempts because it succeeds; dashes are attempts never spent.

This is the pathology the paper is about in a clear example. Sequential revision is explicitly told, seven times, that 1274 is incorrect, and each time it reproduces the same coordinate computation and returns 1274 again. The failed attempt in context does not merely fail to help; instead, it pins the model to the mode that produced it. Independent sampling avoids the pin, spending its eight attempts on seven distinct answers, but has no mechanism to use the information that 1274 was rejected, and it never finds 588 either. FLEx starts from a different wrong answer (924) and, conditioned negatively on it, lands on the correct one at the second attempt.

The two revision transcripts also differ in a way that the answer table hides. Sequential revision's second attempt is 3,029 characters and restates the same shoelace-formula derivation as the first. FLEx's second attempt is 14,493 characters, which is roughly $4.8\times$ longer and abandons the coordinate computation for a different argument about the reflected points.

A second example. On an AIME 2025 problem about a product of logarithms (reference answer 106), independent sampling produces $802, 6241, 6113, 1336, 6017, 109, 31, 4139$ and sequential revision produces $673, 22499, 45431, 45431, 45431, 12419, 36989, 12419$ (note the repeated 45431) while FLEx produces $215, 521, \mathbf{106}$ and stops. Here, sequential revision is not stuck on a single value, but it drifts into increasingly large wrong answers; FLEx's three attempts stay in the right order of magnitude and converge.

B.2 What the model actually sees

For the problem in Tab. 8, the two contexts at the second attempt are as follows. The clean context is unchanged from the first attempt:

```
system: You are a helpful assistant.
user: In $\triangle ABC$ points $D$ and $E$ lie on ...
Find the area of heptagon $AFNBCEM$. Please reason step by
step, and put your final answer within \boxed{}.
```

The failure-conditioned context appends the rejected attempt and the verifier's verdict:

```
system: You are a helpful assistant.
user: In $\triangle ABC$ ... within \boxed{}.
assistant: ... $\frac{1}{2} \cdot 2548 = \boxed{1274}$
user: Your previous answer of '1274' is incorrect. Please
try again, reasoning step by step, and put your final answer
in \boxed{}.
```

Every token of the next attempt is sampled from the extrapolation of the two logit vectors these contexts induce, and is appended to both.

B.3 Failure signals by domain

For reference, a realized feedback turn from each domain:

- **Math.** `Your previous answer of '1274' is incorrect. Please try again, reasoning step by step, and put your final answer in \boxed{}.`
- **Instruction following.** `Your response did not follow the following constraint(s): 'count:keywords_multiple'. Please revise the response to satisfy all constraints and try again.`
- **Coding.** `The above code is incorrect and got a wrong answer.\nInput: 5\n1 2 3 4 5\nGenerated Output: 12\nExpected: 15`

The math signal is a single bit plus the rejected answer; the other two localize the failure. FLEx improves on all three, which suggests the contrast is driven by conditioning on the failed trajectory rather than by the informativeness of the feedback attached to it.

B.4 Instruction following example

IFBench example 20, Qwen3 8B at the canonical instruction-following setting. The prompt carries two constraints, one of which is machine-checkable:

```
Include exactly 3 numbers in the response. Make this
short: An alignment of resources and education goals
```

```
within each community is needed to support the education
ecosystem of students, teachers, and parents and assist
in the adjustment to the new normal---homeschooling,
parents-teachers training, community internet centers, a
Citizen Watch for Education, and establishing LGU leaders as
education champions.
```

The first attempt reads the instruction literally and answers *about* the constraint rather than satisfying it:

```
3 numbers:  100% alignment, 3 key areas, 5 initiatives.
```

This contains four numbers (3, 100, 3, 5), not three, so the `count:numbers` checker rejects it and the verifier generates:

```
Your response did not follow the following constraint(s):
`count:numbers`.  Please revise the response to satisfy all
constraints and try again.
```

The second attempt, decoded with the first in the failure-conditioned context, abandons the meta-answer entirely and satisfies the constraint with an enumeration:

```
To strengthen the education ecosystem, communities need
alignment of resources and goals.  Key initiatives include:
1.  **Homeschooling support** through accessible materials.
2.  **Community internet centers** for digital learning.
3.  **LGU leaders as education champions** to drive change.
```

Exactly three numbers, and the checker accepts. Note that the feedback names the violated constraint but not how to satisfy it; the change in strategy is not something the feedback asks for.

B.5 CODING EXAMPLE

LiveCodeBench v6 example 17, using Qwen3 8B at the canonical coding setting. The task is to report, after each query in a sequence of coloring operations, how many adjacent array positions share the same color. Uncolored positions hold 0 and must not count as a match.

The first attempt maintains a running count but forgets this exclusion:

```
for index, color in queries:
    prev_color = nums[index]
    if index > 0:
        if prev_color == nums[index - 1]:  # <-- counts 0 == 0
            count -= 1
        if nums[index - 1] == color:
            count += 1
    if index < n - 1:
        if prev_color == nums[index + 1]:  # <-- counts 0 == 0
            count -= 1
        if nums[index + 1] == color:
            count += 1
    nums[index] = color
    result.append(count)
```

Two uncolored neighbors are therefore treated as a matching pair, causing the counter to be decremented for pairs that were never counted in the first place and allowing the answer to become negative. Executing the program against the public test produces:

```
The above code is incorrect and got a wrong answer.
Input: [4, [[0, 2], [1, 2], [3, 1], [1, 1], [2, 1]]]
Generated Output: [-1, -1, -2, -3, -1]
Expected: [0, 1, 1, 0, 2]
```

The negative outputs are a direct fingerprint of the bug, and the second attempt repairs exactly this issue: it adds two guards, growing from 854 to 894 characters, while leaving the rest of the program unchanged:

```
if prev_color == nums[index - 1] and prev_color != 0:
    count -= 1
...
if prev_color == nums[index + 1] and prev_color != 0:
    count -= 1
```

This attempt passes. The contrast between this case and the math example in §B.1 is instructive and helps explain why coding behaves differently throughout the paper. Here, the verifier feedback includes a concrete counterexample that localizes the defect, so conditioning directly on the failure can be productive, and sequential revision is already a strong baseline (Table 7). In the math setting, by contrast, the verifier provides only a correctness signal; conditioning on the failed attempt tends to reproduce its behavior, while using that failure contrastively recovers the gain.